EDUCACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Orizaba

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

OPCIÓN I.- TESIS

TRABAJO PROFESIONAL

"DESARROLLO DE UNA APLICACIÓN PARA APOYAR EN LA ENSEÑANZA DE LA PROGRAMACIÓN BASADA EN UN LENGUAJE PARA CREAR PATRONES GEOMÉTRICOS"

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA:

I.T.I. Josué Iván Picie Alcaraz

DIRECTOR DE TESIS:

M.C.E. Beatriz Alejandra Olivares Zepahua

CODIRECTOR DE TESIS:

M.R.T. Ignacio López Martínez

ORIZABA, VERACRUZ, MÉXICO. MARZO 2022

Orizaba, Veracruz, 11/marzo/2022
Dependencia: **División de Estudios de Posgrado e Investigación**
Asunto: **Autorización de Impresión**
OPCION: **I**

C. JOSUÉ IVÁN PICIE ALCARAZ
Candidato a Grado de Maestro en:
SISTEMAS COMPUTACIONALES
P R E S E N T E.-

De acuerdo con el Reglamento de Titulación vigente de los Centros de Enseñanza Técnica Superior, dependiente de la Dirección General de Institutos Tecnológicos de la Secretaría de Educación Pública y habiendo cumplido con todas las indicaciones que la Comisión Revisora le hizo respecto a su Trabajo Profesional titulado:

" Desarrollo de una aplicación para apoyar en la enseñanza de la programación basada en un lenguaje para crear patrones geométricos"

comunico a Usted que este Departamento concede su autorización para que proceda a la impresión del mismo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
CIENCIA – TÉCNICA - CULTURA®

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE ORIZABA
DIV. DE EST. DE POSGRADO E INVEST.

DR. MARIO LEONCIO ARRIOJA RODRÍGUEZ
JEFE DE LA DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN

OG-13-F06

"2021: Año de la Independencia"

Orizaba, Veracruz, 28/octubre/2021
Asunto: **Revisión de trabajo escrito**

C. MARIO LEONCIO ARRIOJA RODRÍGUEZ
JEFE DE LA DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
P R E S E N T E.-

Los que suscriben, miembros del Jurado, han realizado la revisión de la Tesis del (la) C.

JOSUÉ IVÁN PICIE ALCARAZ

la cual lleva el título de:

"Desarrollo de una aplicación para apoyar en la enseñanza de la programación basada en un lenguaje para crear patrones geométricos"

y concluyen que se acepta.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
CIENCIA – TÉCNICA - CULTURA®

PRESIDENTE:	M.C.E. BEATRIZ A. OLIVARES ZEPAHUA	FIRMA
SECRETARIO:	M.R.T. IGNACIO LÓPEZ MARTÍNEZ	FIRMA
VOCAL:	M.S.C. LUIS ÁNGEL REYES HERNÁNDEZ	FIRMA
VOCAL SUP.:	M.C. CELIA ROMERO TORRES	FIRMA

TA-09-26

CARTA DE ORIGINALIDAD

En la ciudad de Orizaba, Veracruz, el día 01 del mes de Septiembre del año 2021, el (la) que suscribe Josué Iván Picie Alcaraz, alumno (a) del programa de Maestría en Sistemas Computacionales con número de control M19011615, manifiesta que es autor (a) del trabajo de tesis titulado "Desarrollo de una aplicación para apoyar en la enseñanza de la programación basada en un lenguaje para crear patrones geométricos." y declaro que el trabajo es original, ya que sus contenidos son producto de mi directa contribución intelectual. Todos los datos y las referencias a materiales ya publicados están debidamente identificados con su respectivo crédito e incluidos en las notas bibliográficas y en las citas que se destacan como tal y, en los casos que así lo requieran, cuento con las debidas autorizaciones de quienes poseen los derechos patrimoniales. Por lo tanto, me hago responsable de cualquier litigio o reclamación relacionada con derechos de propiedad intelectual, exonerando de toda responsabilidad al Tecnológico Nacional de México / Instituto Tecnológico de Orizaba.

Josué Iván Picie Alcaraz

Nombre y firma

CARTA DE CESIÓN DE DERECHOS

En la ciudad de Orizaba, Veracruz, el día 01 del mes de Septiembre del año 2021, el (la) que suscribe Josué Iván Picie Alcaraz, alumno (a) del programa de Maestría en Sistemas Computacionales con número de control M19011615, manifiesta que es autor (a) del trabajo de tesis bajo la dirección de Beatriz Alejandra Olivares Zepahua y cede los derechos del trabajo de tesis titulado "Desarrollo de una aplicación para apoyar en la enseñanza de la programación basada en un lenguaje para crear patrones geométricos." al TecNM/Instituto Tecnológico de Orizaba para su difusión y divulgación, con fines académicos y de investigación.

Queda estrictamente prohibido reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del Tecnológico Nacional de México/Instituto Tecnológico de Orizaba. Este puede obtenerse escribiendo a la siguiente dirección: msc@orizaba.tecnm.mx . Si el permiso se otorga, cualquier usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Josué Iván Picie Alcaraz

Nombre y firma

Agradecimientos

Agradezco al Consejo Nacional de Ciencia Y Tecnología (CONACYT) por el soporte financiero y al Tecnológico Nacional de México (TecNM) por permitir el desarrollo de este proyecto.

Agradezco al Instituto Tecnológico de Orizaba por brindarme la oportunidad de seguir aprendiendo, le doy las gracias a los docentes por haber compartido sus conocimientos en el transcurso de la obtención del grado, para mí fue muy valioso aprender de cada uno de ellos, especialmente al maestro Ignacio López Martínez, el cual tiene una forma extraordinaria de enseñar y siempre hacia ver todo de una manera más sencilla o fácil de entender.

Agradezco profundamente a mi directora de tesis, la maestra Beatriz Alejandra Olivares Zepahua, por su paciencia y atención prestada durante el transcurso de esta etapa, por su compromiso y ayuda brindada en la investigación de este proyecto y por orientarme para lograr los objetivos de este.

Índice general

Índice de figuras

Índice de tablas

Índice de listas

Resumen

Para lograr que los habitantes de un país mejoren su calidad de vida es necesario enfocarse en la educación de estos, para que así ellos mismos puedan impulsar a sus países con las habilidades y conocimientos adquiridos a través de los años. Es por esto por lo que México y en otros países de Latinoamérica se han enfocado en instruir a sus habitantes en Ciencia, Tecnología, Ingeniería y Matemáticas, mejor conocidas como STEM de acuerdo con sus siglas en inglés. Alrededor del mundo al igual que en México se han creado asociaciones para la enseñanza de STEM con la meta de que los individuos obtengan habilidades enfocadas al uso y creación de tecnologías. Existe en México una asociación enfocada en la Tecnología particularmente en la computación, llamada Cuantrix, la cual tiene el objetivo de instruir a un millón de niños, niñas y jóvenes en la programación al año, lo malo de su causa es que los recursos con los que cuenta son pocos en el idioma español, aproximadamente el 80% de las herramientas se encuentran en inglés.

Por el motivo expuesto en el párrafo anterior, el objetivo de la solución propuesta es la generación de un lenguaje de programación en idioma español junto con un ambiente de desarrollo integrado que conceda generar patrones geométricos como un soporte a la enseñanza de las STEM en México.

Introducción

STEM es una abreviatura que menciona las siguientes áreas de conocimiento, *Science, Technology, Engineering and Mathematics* (Ciencia, Tecnología, Ingeniería y Matemáticas). Alrededor del mundo estas áreas están ganando un alto impacto para su estudio ya que brindan conocimientos necesarios para el desarrollo de los países. Gracias al estudio de las STEM, México se posiciono como el octavo país en la clasificación de "Digital Nations & Super Cities" en el 2020 de acuerdo con [1].

La importancia que ofrecen las personas que se desempeñan es estas áreas es muy importante dentro del país ya que su presencia amplia la economía y el mercado laboral. Su presencia tiene asociada también la existencia de conocimientos y experiencias en tecnologías avanzadas, las cuales brindan un gran impacto en el país en donde se desarrollan, muy relacionados con la mejora de los negocias del mismo país.

Con base en lo anterior, la presente tesis tiene como objetivo principal el desarrollo de una aplicación que permita crear patrones geométricos como una forma de apoyo a la enseñanza de la programación.

Con la intención de brindar al lector un claro entendimiento de la tesis, el presente documento se encuentra compuesto por tres capítulos:

Dentro del capítulo uno "Antecedentes" se describen el marco teórico, planteamiento del problema, objetivo general y específicos y la justificación que llevó a la elaboración del presente trabajo. En el capítulo dos, "Estado de la práctica", se exponen resúmenes de investigaciones ya desarrolladas que cuentan con un grado de similitud a la propuesta ofrecida dentro de este documento y se muestra un análisis comparativo que asiste al lector para hacer una clara distinción entre las investigaciones mostradas y el presente trabajo. El tercer capítulo, "Aplicación de la Metodología", se aborda todo lo relacionado con el desarrollo del trabajo, en el cuarto capítulo, "Resultados", se llevan a cabo dos casos de estudio para comprobar la eficacia del proyecto y finalmente en el quinto capítulo, "Conclusiones y Recomendaciones", se mencionan las consecuencias que son fruto del estudio y desarrollo del proyecto, así como consejos a considerarse.

Capítulo 1. Antecedentes

En este capítulo se presentan los conceptos más relevantes relacionados con el trabajo presentado, como lo son: el problema a resolver, los objetivos generales y específicos, así como la justificación de lo que se realizará.

1.1 Marco teórico

En esta sección se presentan los conceptos más relevantes relacionados con el tema de investigación.

1.1.1 Lenguaje de programación

En [2] se hace mención a un lenguaje de programación como el lenguaje que una computadora puede entender, el cual los desarrolladores de software utilizan para crear programas de software, aplicaciones u otros conjuntos de instrucciones para que las computadoras los ejecuten. Un lenguaje de programación también es visto como un idioma en conjunto de reglas gramaticales las cuales instruyen a las computadoras para realizar tareas específicas, cada lenguaje de programación cuenta sus propias palabras reservadas y una sintaxis para ordenar las instrucciones. Antes de poder accionar un programa, lo principal es traducirlo en un formato entendible para una computadora. Compilador, es el nombre que se les da a los sistemas que se encargan de esta acción.

1.1.2 Compilador

Un compilador es descrito como un programa de software el cual se encarga de procesar frases escritas en un lenguaje de programación especifico y transformarlo en lenguaje maquina o código que una computadora pueda entender. A este proceso de convertir lo estructurado con un lenguaje de alto nivel a un lenguaje máquina se le conoce como compilación.

En [3] se mencionó que la compilación es usualmente implementada como una secuencia de transformaciones (LF, L1), (L1, L2), …, (Ln, LO), donde LF es el Lenguaje Fuente y LO es el Lenguaje objetivo. Cada lenguaje Li es llamado como un lenguaje intermediario. Los

lenguajes intermediarios son herramientas conceptuales utilizadas en descomponer la tarea de compilar desde un LF a un LO. El diseño de un compilador en particular determina cuál lenguaje intermediario será utilizado para estructurar la información durante la compilación.

1.1.2.1 Análisis lexicográfico

Análisis léxico, es como se le conoce a la primera fase de un compilador. El analizador léxico obtiene el flujo de caracteres que componen el producto inicial y los agrupa en secuencias significativas, conocidas como lexemas. Es un proceso mediante el cual una secuencia de caracteres es convertida a una secuencia de "*tokens*" (unidades léxicas mínimas), es decir, es un proceso que ayuda a reconocer si las "palabras" son válidas en un lenguaje, pero sin verificar si se forma una "oración" válida en el lenguaje o no. Por ejemplo, el siguiente texto:

2 + 5 * 10

Es lo que se conoce como una expresión aritmética y se podría traducir en la siguiente secuencia de *tokens*:

(Número, 2) (Operador, +) (Número, 5) (Operador, *) (Número, 10)

Un *token* es conocido como un par que asocia un tipo léxico y un valor literal opcional. Sin embargo, esta estructura depende finalmente de la implementación del analizador léxico [4].

1.1.2.2 Análisis gramatical (parser)

Análisis gramatical o *parsing,* es el nombre que se le asigna a la segunda fase de un compilador. El analizador gramatical usa los componentes de los *tokens* generados por el analizador léxico para producir un árbol de sintaxis abstracta (AST), el cual sirve para visualizar y manejar la estructura gramatical del flujo de los *tokens*. El análisis gramatical también conocido como sintáctico corresponde al proceso de verificar una secuencia de *tokens*. En esta etapa se decide si un texto de entrada cumple con las reglas gramaticales del lenguaje, es decir, si los elementos están en el orden correcto. Por ejemplo, las siguientes expresiones matemáticas utilizan símbolos válidos, no obstante, sólo dos de ellas son válidas sintácticamente:

a) $(2 + 2) * 2$

b) $7*+10$

c) $241 + 4$

d) $+*+(7$

e) $4/0$

La expresión (d) aplica incorrectamente los operadores de adición y multiplicación, además de tener mal posicionados los paréntesis. Nótese que la expresión (e) es válida sintácticamente, pues es la división entre dos números, sin embargo, produciría un error al ser evaluada (división entre cero).

El análisis sintáctico suele ocurrir como una etapa posterior al análisis lexicográfico, operando sobre la secuencia de *tokens* generada por el analizador léxico. El objetivo de un analizador gramatical es construir una estructura, por lo general algún tipo de árbol, que represente los elementos sintácticos encontrados [4].

1.1.2.3 Analizador semántico

El analizador semántico es la tercera fase de un compilador, la cual hace uso del Árbol de Sintaxis Abstracta, así como de la información en la tabla de símbolos para constatar que la definición del lenguaje de programación tenga congruencia semántica con la información obtenida. De igual manera obtiene información que el programador crea necesaria y la almacena, tanto en el árbol sintáctico como en la tabla de símbolos, para que emplearla más tarde en caso de ser necesario durante la generación de código intermedio.

Otro punto clave del análisis semántico es la validación de tipos, para esto durante el análisis se examina que cada elemento posea operandos que coincidan. Como ejemplo, si un lenguaje de programación requiere que para un identificador solo se puedan utilizar números enteros; se debe reportar un error si se utilizan letras o números con punto decimal al momento de intentar generarlo [5].

1.1.2.4 Generación de código intermedio

Durante el procedimiento de transformar código fuente a código destino, un compilador puede generar una o más representaciones intermedias, existen diferentes tipos de representación intermedia los cuales pueden ser usados según convenga. Para generar una representación intermedia es necesario utilizar algún tipo de mecanismos para obtener los datos en el tiempo de ejecución.

Una vez realizados el análisis gramatical y el semántico del código fuente, la mayoría de los compiladores transforman el código fuente en una representación intermedia que es mayor mente independiente del lenguaje de origen y del código máquina. Una representación intermedia debe tener dos propiedades importantes: debe ser fácil de producir y fácil de traducir en la máquina destino [5].

1.1.2.5 Generación de código

Para la última fase del compilador se encuentra el generador de código, el cual parte de la información obtenida de una representación intermedia del código fuente y la designa al lenguaje destino. Dependiendo del lenguaje destino, se seleccionan los valores importantes del código fuente para después con la información obtenida en la Representación Intermedia se traduzcan en instrucciones que realizan la misma tarea para el lenguaje destino esperado. [5].

1.1.3 Herramienta

Según el diccionario de Cambridge [6], se conoce como herramienta a aquel instrumento que se usa con las manos para crear o reparar algo. Así mismo como algo que ayuda en una actividad en particular.

Hoy en día con el avance en tecnología las personas suelen utilizar herramientas digitales, las cuales son recursos que los ayudan a completar tareas o actividades de una manera más rápida.

1.1.4 Patrones Geométricos

Los patrones geométricos en [7] se definen como figuras geométricas de una misma forma, que se repiten en una serie, visto como las exactitudes que constituyen a las figuras geométricas, círculos, triángulos, cuadrados, entre otros, los patrones geométricos pueden ser flexibles y con buen efecto visual. Muchos de los patrones geométricos fueron inspirados en la naturaleza, donde se pueden observar patrones con solo mirar. Las rayas de las cebras, los hexágonos en los caparazones de las tortugas y las líneas de las abejas son algunos de estos ejemplos. Es frecuente que el ser humano reproduzca estos patrones en algunas artesanías como alfarería o bordado.

1.1.5 IDE

Un Entorno de Desarrollo Integrado es un producto de software que combina todas las características y herramientas que necesita un desarrollador de software. Es de naturaleza gráfica, lo que significa que utiliza ventanas y controles como botones para mostrar información y aceptar aportes del usuario.

Según Yoshiaki [8] el IDE (*Integrated Development Environment*, Entorno de Desarrollo Integrado) se compone de un editor de código fuente, un analizador de impacto, un iniciador de casos de prueba, una interfaz gráfica de usuario (GUI) y una base de datos.

1.1.6 Nodo

Es cualquier punto terminal de un segmento [].

1.1.7 Segmento

Es cualquier parte de una línea a las que pertenecen dos puntos terminales diferentes [].

1.1.8 Grafo

"Es cualquier conjunto de segmentos cuyos únicos puntos en común son llamados nodos o vértices, en donde llegan o salen una o varias líneas denominadas aristas o fronteras" [9]. Un grafo está compuesto de vértices y aristas.

"Representación simbólica de los elementos constituidos de un sistema o conjunto, mediante esquemas gráficos, puede ser representado mediante una cuadricula" [10].

1.1.9 Grafo impar

"Es aquel en donde llegan un número impar de aristas a los nodos" [9].

1.1.10 Grafo par

"Es aquel que tiene sus nodos con aristas pares" [9].

1.1.11 Multígrafo

"En un grafo se admiten aristas múltiples" [9].

1.1.12 Teorema de Euler

"Fórmula de Euler: Sea G = (V, E) un grafo o multígrafo plano conexo con |E| = e (número de aristas) y |V| = v (número de nodos).

Sea r el número de regiones en el plano determinadas por una inmersión (o representación) plana de G; una de estas regiones tiene un área no acotada y se conoce como región infinita, entonces:

r = 2-v+e " [9]

1.2 Situación tecnológica, económica y operativa de la empresa

El Instituto Tecnológico de Orizaba es una institución de educación superior orientada al desarrollo de la ciencia y la técnica mediante la formación de recursos humanos y la investigación en dichas áreas, para el fortalecimiento económico y social de México. Es un establecimiento público que depende de la Secretaría de Educación Pública SEP y pertenece al Tecnológico Nacional de México que agrupa varios Institutos Tecnológicos Federales y descentralizados en el país. Se encuentra en la ciudad de Orizaba, estado de Veracruz en México ubicado en Oriente 9, Colonia Emiliano Zapata, en la ciudad de Orizaba, Veracruz. Esta institución ofrece programas de licenciatura, maestría y doctorado. En el área de maestrías ofrece Maestría en Ingeniería Electrónica, Maestría en Ingeniería Industrial, Maestría en Ciencias en Ingeniería Química, Maestría en Ingeniería Administrativa y Maestría en Sistemas Computacionales.

1.3 Planteamiento del problema

En los últimos años, a nivel mundial, se está haciendo hincapié en brindar conocimiento en las siguientes áreas: Ciencia, Tecnología, Ingeniería y Matemáticas (STEM por sus siglas en inglés) para así modernizar el conocimiento de las personas, impulsar el crecimiento de los países y personas que lo habitan, reducir la abertura de género y/o pobreza, entre otros. En México se crearon diferentes proyectos dedicados al tema [11], [12], [13], [14] y especialmente en la Computación brindando conocimientos de programación [15].

A nivel internacional existen varias investigaciones relacionadas con la enseñanza de la programación a estudiantes de distintas edades como una forma de acercamiento a STEM [16], en algunos casos se relacionan con el uso de la robótica [17] o se focalizan en la solución de problemas y pensamiento lógico distanciándose de las bases matemáticas que propone STEM [18]. Otros estudios se centran en características de edad y/o género para identificar elementos que faciliten el aprendizaje de la programación [19].

Para abordar este problema se busca desarrollar un lenguaje en conjunto de un IDE que permita al usuario familiarizarse con la programación y conceptos matemáticos mientras crea patrones geométricos, los cuales podrá ver reflejados en una en otro apartado de la herramienta al ejecutar el programa. Se evaluará la posibilidad de reproducir estos patrones mediante una máquina de coser con ayuda un intermediario, como podría ser Arduino.

1.4 Objetivo general y objetivos específicos

A continuación, se muestran el objetivo general y los objetivos específicos.

1.4.1 Objetivo general

Desarrollar una aplicación que permita crear patrones geométricos como una forma de apoyo a la enseñanza de la programación.

1.4.2 Objetivos específicos

- Analizar aplicaciones existentes de asistencia al aprendizaje de la programación para detectar las mejores prácticas.
- Delimitar las características del lenguaje de programación para patrones geométricos tomando como referencia el telar de Jacquard.
- Crear la gramática del lenguaje de programación para patrones geométricos.
- Diseñar la arquitectura de la aplicación.
- Construir la aplicación.

1.5 Justificación

La mayoría de los estudios coincide que el acercamiento a la programación en una edad corta le brinda al usuario una ventaja a diferencia de los que no conocen acerca de esta, que los entornos visuales resultan más llamativos para los infantes y adolescentes y que se evocan mayor interés en un tema cuando se obtiene algo físico tangible. Desafortunadamente, la mayoría de las investigaciones que se han llevado a cabo se refieren a experiencias en países desarrollados y muchas de las herramientas disponibles, incluso las referenciadas específicamente para México [15], se encuentran en inglés. Es común también que las herramientas tengan predefinidos una serie de ejercicios que facilitan tanto el aprendizaje de la programación como la verificación de dicho aprendizaje, pero no permiten crear nuevos ejercicios.

Por otra parte, una de las primeras experiencias de programación que se tienen registradas, es el telar de Jacquard, que usaba una serie de tarjetas perforadas para describir los diseños a crear en el telar. Dada la amplia tradición de bordados textiles en México, muchos de ellos basados en diseños geométricos como grecas que son fácilmente representables en instrucciones simples, se considera factible crear un lenguaje para describir dichos diseños; la aplicación para trabajar el lenguaje tendría tanto elementos predefinidos (uso de instrucciones específicas para replicar un diseño y con ello aprender conceptos básicos de programación) como los medios para crear diseños propios.

Capítulo 2. Estado de la práctica

En este capítulo se dan a conocer algunos trabajos relacionados directa o indirectamente con el proyecto de tesis.

2.1 Trabajos relacionados

La innovación textil se asocia con la creación de ensamblajes de materiales, capaces de responder a condiciones cambiantes a través de su material y composición estructural. Sin embargo, las herramientas digitales fallan en asistencia para simulación y predicción de formas en la creación de tejidos. En [20] se desarrolló una herramienta digital para innovar en el sector industrial textil, con capacidad de realizar múltiples combinaciones de tejidos en un mismo telar, usando las siguientes tecnologías: Software CAD (*Computer Aided Design*; Diseño asistido por computadora) 3D Rhinoceros, un entorno de programación visual Grasshopper para diseño paramétrico y como interfaz gráfica de usuario, lenguaje de programación Python y una máquina industrial de tejido "*Stoll*". Como resultado de esta herramienta se destaca la mejora en la flexibilidad, precisión y velocidad de fabricación de los tejidos.

Tisza et al. [19] evaluaron las formas de aprendizaje formal, informal y no-formal aplicadas a niños y enlazadas con STEM (*Science, Technology, Engineering, Mathematics*; Ciencia, Tecnología, Ingeniería, Matemáticas) en Europa; las diferencias entre estas formas de aprendizaje son, respectivamente: a) un espacio de aprendizaje formal, sigue un programa de estudios estructurado y es obligatorio; b) fuera de un ambiente formal de aprendizaje, sigue un programa de estudios estructurado, usualmente voluntario; c) donde sea, no sigue un programa de estudios estructurado y es voluntario. Se llevó a cabo un estudio de encuestas y se tomó un muestreo de 128 instructores expertos en el diseño e implementación de actividades relacionadas con STEM utilizando las diferentes formas de aprendizaje, provenientes de nueve países europeos, este estudio determinó que hay una diferencia entre las actividades en las que chicas frecuentemente participan y aquellas actividades en las que los chicos participan. Se encontró que las chicas participan más frecuentemente en varias actividades destinadas a

involucrar a los participantes en temas como arte, biología, química y física. Mientras que los chicos participan en actividades que mejoraban sus habilidades en ciencias de la computación. Esta información brindó una manera más rápida para determinar qué tipo de actividades se adapta mejor para enseñar algún tema relacionado con STEM, dependiendo de la edad y género de los usuarios.

Las máquinas de tejido controladas por computadora son ampliamente utilizadas en la industria textil para la producción de accesorios, como sombreros y guantes, así como prendas completas. Las máquinas de tejido más avanzadas moldean hilo en un complejo artículo tridimensional listo para usar con un procesamiento mínimo. Sin embargo, estos artículos actualmente son laboriosamente diseñados a mano utilizando herramientas específicas de máquina de tejido que sólo son accesibles a los expertos. En [21] se brindó un algoritmo que automáticamente convierte formas 3D (tridimensionales) en instrucciones de máquina de tejido, el método tomó como entrada una malla triangular 3D que produjo instrucciones para máquinas de tejer en tres pasos. Primero, creó un gráfico de tejido especialmente estructurado en la superficie, después creó instrucciones de tejido a partir de este gráfico y finalmente estas instrucciones fueron programadas para ubicar las agujas en la máquina de tejer, utilizando una máquina de tejer Shima Seiki SWG091N2 calibre 15 con cama en V e hilo acrílico *Tamm Petit*, como resultado se lograron generar objetos a partir del modelo, sin embargo, se obtuvo que los objetos creados fueron inherentemente elásticos, por lo cual las formas generadas por las salidas del sistema no fueron exactamente la forma especificada en la entrada del sistema.

Las máquinas industriales de tejido producen detalles finos, superficies y formas tridimensionales (3D) rápidamente y sin intervención humana, sin embargo, las herramientas utilizadas para programarlas requieren una manipulación y comprensión detallada de operaciones de tejido de bajo nivel. McCann et al. [22] presentaron un compilador que convirtió automáticamente conjuntos de formas primitivas de alto nivel (tubos o formas) en instrucciones de bajo nivel para las máquinas de tejido, estas formas de alto nivel permitieron programar objetos tejidos que de otra manera requerirían miles de ediciones a instrucciones de bajo nivel. En el núcleo del compilador existe un algoritmo de planificación de transferencia para ciclos de punto, el cual primero elige un rol para los valores de cada puntada y luego

iterativamente hace una elección entre “Colapsándose/Expandiéndose” hacia la cama trasera o frontal, eligiendo entre: arriba, la fase de colapso mueve las puntadas a las agujas de la cama trasera; arriba a la derecha, el ciclo colapsado se mueve a la cama delantera; abajo, la fase de expansión mueve las puntadas desde la cama delantera hacia la cama trasera. Este algoritmo permitió la traducción de operaciones de modelado y programación de alto nivel a nivel de las agujas con la ayuda de una máquina industrial Shima Seiki SWG091N2 calibre 15 con cama en V.

Las máquinas de tejido industriales se usan comúnmente para fabricar piezas de hilos con formas complicadas; sin embargo, diseñar patrones para estas máquinas requiere entrenamiento extensivo. Narayanan et al. [23] presentaron la primera interfaz de programación visual general para crear objetos 3D con acabados superficiales complejos en máquinas de tejido industrial. En el núcleo de la interfaz diseñaron una versión, aumentada, de una estructura de datos de malla de puntada. La malla de puntada aumentada almacena operaciones de tejido de bajo nivel por cara y codifica las dependencias entre caras usando etiquetas de borde dirigido, el sistema generó tejido de punto aumentado de modelos 3D, permitió a los usuarios editar estas mallas de manera que conservaron su capacidad de tejido y programaron la orden de ejecución y ubicación de cada cara para producción de una máquina de tejido. El sistema demostró que sus operaciones de edición preservaron la capacidad de tejido suficiente para combinar hasta dos patrones de puntada en las máquinas de tejido con la misma orientación y sobre la misma superficie, utilizando máquinas industriales Shima Seiki SWG091N2 calibre 15 con cama en V.

La industria de tejido fabrica artículos complejos, suaves y con propiedades de textura únicas, pero diseñar estas formas o texturas toma un tiempo largo y es necesaria la ayuda de expertos en el tema para su creación. En [24] se abordaron los desafíos de diseñar y utilizar texturas tejidas digitalmente, se contribuyó con un conjunto de datos medidos y fotografiados de 300 texturas tejidas. Basado en los hallazgos de este conjunto de datos, se aportaron dos algoritmos para manipular KnitGraphs. KnitCarving, el cual da forma a un gráfico mientras respeta una textura, y KnitPatchin, que combina gráficos con texturas dispares mientras mantiene una forma consistente. KnitGraph es una estructura que mantiene cuatro propiedades: (1) cada

bucle se estabiliza haciendo pasar otro bucle a través de él; (2) cada bucle se extrae a través del bucle [s] que se construyó previamente; (3) los cables tienen una profundidad de cruce explícita, y (4) todos los bucles se realizan a una distancia limitada, evitando roturas de hilo. KnitPick presentó una nueva técnica para diseñar, usar y fabricar texturas tejidas, es adecuado principalmente para crear láminas compuestas de texturas tejidas y no para objetos con forma, es efectivo en una variedad de texturas, incluyendo patrones de encaje con dependencias entre repeticiones por patrón.

Hoy en día generalmente las mujeres no son alentadas a seguir la programación o alguna otra práctica técnica con la misma frecuencia que los hombres, tanto recreativa como formalmente, debido a suposiciones basadas en el género o sesgos inconscientes. La sociedad termina orillándolas a actividades más artísticas como tejer, bordar o cocinar. En [25] se tuvo como objetivo el resaltar la relación entre las actividades artísticas y la computación, este visualizador fue enfocado a aquellos tejedores *amateurs* que se quieren familiarizar con la programación y que les gustaría desarrollar sus propios patrones de tejido. La versión generada del prototipo del visualizador de tejido permitió la traducción bidireccional entre las especificaciones del código JavaScript para un patrón de tejido, así como un patrón gráfico simbólico que comunique las especificaciones equivalentes brindadas por el usuario. La vista del patrón permitió una vista previa visual aproximada de cómo se vería un producto tejido de acuerdo con estas especificaciones.

Utilizar la codificación en la educación para promover el pensamiento computacional y nutrir las habilidades de resolución de problemas en los niños se convirtió en una tendencia global emergente. Sin embargo, sigue sin ser claro cómo las diferentes modalidades de entrada y salida en la codificación apoyan la práctica de diferentes habilidades de resolución de problemas para los niños. En [16] se presentaron cuatro talleres de codificación que correspondieron a las cuatro combinaciones de modalidades de entrada y salida: GIGO (*Graphical Input Graphical Output*, Entrada Gráfica Salida Gráfica), TITO (*Tangible Input Tangible Output*, Entrada Tangible Salida Tangible), GITO (*Graphical Input Tangible Output*, Entrada Gráfica Salida Tangible), TIGO (*Tangible Input Graphical Output*, Entrada Tangible Salida Gráfica). Fueron en total tres ejercicios realizados por cada participante durante los

talleres. Se contaron el número de intentos de cada niño para cada ejercicio, basados en los registros del sistema y los videos capturados. Los resultados de la prueba posterior al taller resaltaron que los niños en TITO se calificaron a sí mismos con una mejor comprensión de la codificación (4.67 / 5), mayor confianza (5/5) y mayor disfrute (5/5), los mismos resultados denotaron que el método de entrada gráfica distrajo menos que el enfoque de entrada tangible, pero también fue menos provocativo durante la etapa de argumentación. En comparación con la contraparte gráfica, el resultado tangible podría atraer mejor la atención de los estudiantes, apoyar un razonamiento causal y promover una argumentación más activa.

2.2 Análisis Comparativo

La Tabla 2.1 muestra la información del análisis comparativo realizado a los artículos que tienen relación con el proyecto de tesis.

Tabla 2. 1 - Análisis comparativo de trabajos relacionados

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
Karmon et al. [18]	Escasez de innovación digital en el área de fabricación industrial con tejido de punto.	Crearon una herramienta digital para la fabricación industrial de tejidos de punto, que contiene múltiples combinaciones de puntos de tejer dentro de la misma tela, llamadas estructuras de tejido de punto.	• Software CAD 3D Rhinoceros (Rhino). • Entorno de programación visual Grasshopper. • Lenguaje de programación Python. • Máquina industrial de tejido *Stoll*.	La herramienta brindó a las fábricas del sector textil, mayor flexibilidad de fabricación, precisión y velocidad.	En desarrollo
Tisza et al. [17]	¿Cómo es el grupo de edad y el género de los participantes, y el género de los expertos	Proporcionó información acerca del grupo de edad, género, objetivo de	• Estudio de encuestas • Muestreo de 128	El género y la edad están asociados con el objetivo y el contenido de la actividad.	Finalizado

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
	relacionados con diferentes actividades de aprendizaje en Europa?	actividades de aprendizaje relacionadas a STEM en Europa.	instructores altamente experimentados en diseño y seguimiento de actividades de aprendizaje informal y no-formal de nueve países de Europa.	El género de los líderes de la actividad está asociado con el contenido y el objetivo de la actividad.	
Narayanan et al. [19]	Las máquinas de tejido más avanzadas en el sector industrial son capaces de transformar hilo en patrones 3D, sin embargo, necesitan de patrones que deben ser laboriosamente diseñados a mano, utilizando herramientas	Generaron un algoritmo que automáticamente convierte formas 3D en instrucciones de máquina de tejer.	• Máquina de tejer Shima Seiki SWG091N2 calibre 15 con cama en V. • Hilo acrílico Tamm Petit.	Los objetos creados son inherentemente elásticos y el relleno no provee una presión uniforme, por esto, las formas generadas por las salidas del sistema no son exactamente la forma especificada en la entrada del sistema.	Finalizado

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
	específicas de tejido de máquina que solo son accesibles a los expertos.				
McCann et al. [20]	Las máquinas de tejer industriales pueden generar patrones 3D rápido y sin intervención humana, pero las herramientas usadas para programarlas requieren manipulación detallada y un gran conocimiento de operaciones de tejido a bajo nivel.	Generaron un compilador que convierte automáticamente un conjunto de formas de alto nivel primitivas (tubos o formas) en instrucciones de bajo nivel de máquina de tejer.	• Máquina de tejer Shima Seiki SWG091N2 calibre 15 con cama en V. • Hilo de acrílico ("Supersheen 1-ply" de Hilos Yeoman) y lana merina ("Polo 1-ply" de Hilos Yeoman).	Crearon un algoritmo en el núcleo de su compilador para la planificación de transferencia de ciclos de punto, no logra producir artículos en tiempos óptimos, posee un tratamiento de las operaciones básicas de una máquina de tejer.	Finalizado
Narayanan et al. [21]	Las máquinas de tejer industriales se usan comúnmente para fabricar piezas	Desarrollaron la primera interfaz de programación visual general para crear	• Máquina industrial de tejido Shima Seiki SWG091N2 con 15 agujas por	Existe en el núcleo de la interfaz una nueva versión aumentada de estructura de datos de	Finalizado.

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
	complicadas de formas con hilos; sin embargo, diseñar patrones para estas máquinas requiere entrenamiento extensivo.	objetos 3D con acabados superficiales complejos en máquinas de tejido industrial.	pulgada con cama en V.	malla de puntada. La malla de puntada aumentada almacena bajo nivel de operaciones de tejido por cara y codifica las dependencias entre caras usando etiquetas de borde dirigido. El sistema genera tejido de punto aumentado, las mallas de puntada de modelos 3D permiten a los usuarios editar estas mallas de una manera que conserva su capacidad de tejido y programa la orden de ejecución y ubicación de cada cara para la	

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
				producción en una máquina de tejer.	
Hofmann et al. [22]	El tejido fabrica artículos complejos, suaves y con propiedades de textura únicas, pero diseñar estas formas o texturas toma un tiempo largo y es necesaria la ayuda de expertos en el tema para su creación.	Se presentó un compilador que interpreta patrones de textura de tejido a mano como gráficos de tejido para máquinas de tejido.	• *Grammar-Kit parser generator*. • JFlex lexer *generator*. • Máquina industrial de tejido Shima Seiki SWG091N2 con 15 agujas por pulgada con cama en V.	KnitPick presentó una nueva técnica para diseñar, usar y fabricar texturas tejidas, es principalmente adecuado para crear láminas compuestas de texturas tejidas y no para objetos con forma, es efectivo en una variedad de texturas, incluyendo patrones de encaje con dependencias entre repeticiones por patrón.	Finalizado
Yang [23]	Hoy en día generalmente las mujeres no son	Se desarrolló un visualizador de tejido en el que los usuarios	• JavaScript	El visualizador permite la traducción bidireccional fluida	Finalizado

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
	alentadas a seguir la programación o alguna otra práctica técnica con la misma frecuencia que los hombres, tanto recreativa como formalmente, debido a suposiciones basadas en el género o sesgos inconscientes. La sociedad termina orillándolas a actividades más artísticas como tejer, bordar o cocinar.	jugaron a construir formas de tejido mientras aprendieron código JavaScript para formalizar y estructurar su trabajo.		entre especificaciones de tejido escritas y el patrón de tejido, alentando a sus usuarios a jugar con los patrones.	
Zhu et al. [14]	Utilizar la codificación en la educación para promover el pensamiento	Se presentaron cuatro talleres de codificación que correspondieron a las	• Fueron en total tres ejercicios realizados por cada participante durante	Los resultados denotaron que el método de entrada gráfica distrajo menos que el	Finalizado

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
	computacional y nutrir las habilidades de resolución de problemas en los niños se convirtió en una tendencia global emergente. Sin embargo, ¿Cómo las diferentes modalidades de entrada y salida en la codificación apoyan la práctica de diferentes habilidades de resolución de problemas para los niños? sigue sin ser claro.	cuatro combinaciones de modalidades de entrada y salida: GIGO, TITO, GITO y TIGO.	los talleres. Se contaron el número de intentos de cada niño para cada ejercicio, basados en los registros del sistema y los videos capturados.	enfoque de entrada tangible, pero también fue menos provocativo durante la etapa de argumentación. En comparación con la contraparte gráfica, el resultado tangible podría atraer mejor la atención de los estudiantes, apoyar un razonamiento casual, y promover una argumentación más activa.	

Después de revisar los artículos mencionados anteriormente se puede decir que el área textil necesita de una interfaz que permita diseñar patrones de forma amigable, la mayoría de artículos se enfocaron en diseñar y crear formas en 3D como sombreros y guantes, de igual manera brindan información acerca de las mejores modalidades en las que los niños se desenvuelven mejor y aprenden más, pero este proyecto se enfoca en guiar a las personas, con ayuda de una interfaz gráfica, a crear patrones geométricos mientras aprende a programarlos, de esta manera le resultará más sencillo el diseñar sus propios patrones, lo que les brindará conocimiento acerca de la programación para que se vayan adentrando en el tema sin tanta dificultad y les resulte fácil entender conceptos más técnicos.

2.3 Propuesta de solución

Este capítulo tiene como objetivo presentar una descripción general de la propuesta de solución al problema plateado en este proyecto de tesis, así como diversos enfoques y tecnologías analizadas durante el diseño de la propuesta de solución.

2.3.1 Descripción de la solución

Como propuesta de solución se pretende crear un lenguaje de programación muy simple, con instrucciones en español, para crear patrones geométricos basados en líneas y ángulos. Para dar soporte al lenguaje, se propone también desarrollar una aplicación tipo IDE en donde los niños y jóvenes tengan la opción de arrastrar las instrucciones para crear patrones geométricos y colocarlas ordenadamente siguiendo una gramática muy básica pero capaz de comunicar las ideas que sus usuarios necesitan, contando con la opción de visualizar el resultado de la combinación de tales instrucciones.

En la figura 2.3.1 se muestra un esquema de módulos para la aplicación tipo IDE.

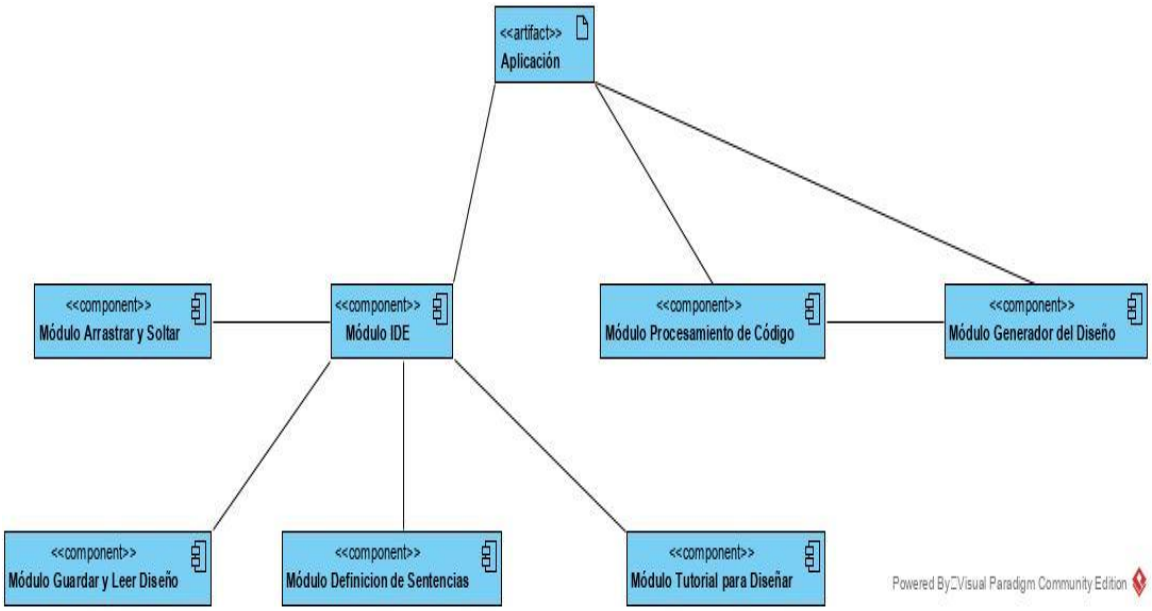

Figura 2. 3. 1 - Esquema de solución propuesto

A continuación, se realiza una breve explicación de cada uno de los módulos que la conforman:

1. Módulo IDE:

   Este módulo será el encargado de generar la vista para el ambiente de desarrollo.

2. Módulo Definición de Sentencias:

   Este módulo será en donde se delimitarán las posibles sentencias a emplear por el usuario para producir un diseño.

3. Módulo Arrastrar y Soltar:

   Este módulo se ocupará de dar el funcionamiento de arrastrar y soltar adentro del IDE para la estructuración de código.

4. Módulo de Procesamiento de Código:

   Este módulo se encargará de interpretar el código estructurado por el usuario para brindarle funcionalidad.

5. Módulo Generador del Diseño:
   Este módulo tendrá como objetivo acoger las instrucciones procesadas para producir el diseño y mostrarlo visualmente.

6. Módulo Tutorial para Diseñar:
   Este módulo mostrará videos explicando cómo generar diversos patrones geométricos para que el usuario comprenda la operatividad de la aplicación al mismo tiempo en el que genera diseños sencillos, para que posteriormente obtenga la habilidad de crear sus propios diseños.

7. Módulo Guardar y Leer Diseño:
   Esté módulo será el encargado de almacenar el diseño en el que se está trabajando para seguir utilizándolo posteriormente si así lo desea el usuario.

Del esquema mostrado se derivan ciertos requerimientos tecnológicos: una herramienta para apoyar en la construcción del módulo de Procesamiento de Código, un lenguaje de programación compatible con el lenguaje de salida de la herramienta mencionada y que además sea capaz de soportar la funcionalidad de “arrastrar y soltar”, un ambiente de desarrollo integrado para crear la aplicación y una metodología para apoyar y dirigir, respectivamente, la construcción de la aplicación motivo de este proyecto.

2.3.2 Solución propuesta

Tomando en cuenta las ventajas y desventajas de las tecnologías, se determinó como solución para desarrollar la propuesta planteada el uso de ANTLR4 como herramienta para la generación del lenguaje, el IDE IntelliJ IDEA y el uso del marco de trabajo JavaFx, así como la metodología Scrum.

2.3.3 Justificación de la solución propuesta

Como herramienta para la generación del lenguaje se optó por ANTLR4, que utiliza el algoritmo ALL(*), el cual se basa en el uso de gramáticas libres de contexto y no solo en expresiones regulares, por lo que se reconocen como *tokens* sin contexto, como lo son comentarios anidados, adicionalmente tiene la capacidad de introducir o extraer *tokens* de acuerdo al contexto semántico, además de también ser adecuado para el análisis sin el uso de un escáner debido a su poder de reconocimiento, lo cual es útil para solventar problemas léxicos sensibles al contexto.

En cuanto al ambiente de desarrollo integrado se seleccionó IntellJ IDEA que, aunque la licencia para la edición *Ultimate* solo es por 1 año para estudiante, puede ser readquirida mientras se cuente con un correo institucional, además de eso cuenta con asistencia a la codificación con finalización de código inteligente, análisis estático profundo, refactorizaciones inteligentes, inspecciones, entre otros. IntelliJ IDEA es compatible con una gran variedad de marcos de trabajo: tanto en el lado del servidor como en *front-end* además permite revisar/realizar cambios, explorar el historial y trabajar con ramas.

Para el marco de trabajo para la generación de la aplicación se decidió utilizar JavaFX, ya que es una herramienta muy completa para crear componentes enriquecidos con vista y sensación mejorada y sigue siendo actualizada.

Por último, como metodología se eligió Scrum, ya que brinda transparencia, facilidad de cambios y un claro manejo de las iteraciones tomando en consideración qué funcionalidad se espera obtener al concluir cada una.

Capítulo 3: Aplicación de la Metodología

En este capítulo se presenta todo lo relacionado con el análisis, diseño y desarrollo del proyecto. Después de realizar la investigación de la problemática expuesta en los antecedentes y la investigación de trabajos asociados, se optó por desarrollar un ambiente de desarrollo integrado en conjunto de un lenguaje de programación simple y reducido en idioma español, el cual ayude a los estudiantes a repasar conceptos matemáticos, así como asimilar conceptos básicos de programación y comprobar fácilmente sus resultados.

Dado lo anterior, para el lenguaje de programación que se aspira a desarrollar se apoyó en conceptos matemáticos simples que se llevan a estudio en matemáticas de primaria en México como parte de la educación básica puesto que se mantienen en mente en todo momento las metas de STEM, pero con un mayor enfoque en la computación. Las instrucciones de los lenguajes que utilizan una salida gráfica, como lo son Logo y Scratch, son muy parecidas ya que se basan en el idioma inglés, utilizar el idioma español en las instrucciones del lenguaje a abordar define una diferencia a sus semejantes. Tomando en cuenta instrucciones muy simples como <inicio>, <fin>, <pintar n unidades>, <avanzar n unidades>, <girar n grados>, <repetir n veces> y <cambiar color>; también se hará uso de una cuadricula para generar la salida, además de considerar el uso de la teoría de grafos para el recorrido dentro de la misma, en donde se utiliza geometría de posición la cual no depende de ninguna medida, el desplazamiento se hace de nodo a nodo en forma horizontal, vertical o diagonal, entonces cuando se habla de una unidad se hace referencia a una arista o camino.

3.1 Identificación de requerimientos

El hacer uso de una metodología de software que ayude como orientación durante el proceso de desarrollar productos de software es uno de los puntos más importantes para obtener software de calidad; las diferentes metodologías para la generación de software tienen como primer punto el recabar datos para tener una idea concisa de lo que el producto debe realizar, esta información se logra obtener mediante entrevistas con el cliente o con los usuarios finales de dicho producto, o de igual manera generando encuestas que se llevaran a cabo mediante los usuarios finales; los datos obtenidos mediante alguna de estas

formas se analizan para así generar los requerimientos funcionales y no funcionales necesarios del producto de software. En el caso de esta propuesta no se tiene un cliente y los usuarios finales identificados son jóvenes entre primaria y secundaria, dado esto si se hubiera optado por alguna de las técnicas descritas anteriormente la mejor opción sería la generación de encuestas a un número considerable de jóvenes en el país.

En cambio, dado que la mayoría de *IDEs*, ya sea para generar productos de software o enfocados en el aprendizaje, cuentan con componentes semejantes, no se llevó a cabo alguna de las técnicas descritas en el párrafo anterior. Ya que existe una opción diferente haciendo uso de los componentes parecidos, llamada: análisis de herramientas semejantes como alternativa a la recolección de requerimientos (técnica de análisis comparativo) la cual se puede apreciar en el libro "*The Deadline*" [26], de Tom DeMarco, en donde narra diferentes ideas que tienen que ver con dirigir proyectos de software y en general sobre ingeniería de software de la mano de uno de los autores más importantes en el área; dicho lo anterior, para la obtención de los requerimientos funcionales y no funcionales del producto a generar, se planteó la utilización de la técnica de análisis comparativo analizando diferentes *IDEs* enfocados a la generación de productos de software al igual que algunos enfocados al aprendizaje de programación.

3.2 Estudio de IDEs enfocados a generar productos de software

1) Eclipse:

El IDE Eclipse es un ambiente de desarrollo rico en funciones, el cual proporciona varias funciones de programación para los desarrolladores y facilita su trabajo al crear productos de software, su uso principal es para la generación de software basado en Java, aunque es compatible con otros lenguajes de programación lo que hace que la plataforma sea favorable para el que los programadores la utilices para generar diferentes productos de software [27].

2) Visual Studio:

El ambiente de desarrollo integrado Visual Studio es una plataforma utilizada para editar, depurar y compilar código, para después publicar una aplicación [28]: Es un programa rico en funciones que se utiliza para el desarrollo de software, Visual Studio incluye herramientas de finalización de código, generador de diseños gráficos, entre otras funciones para facilitar el desarrollo de software.

3) NetBeans:

NetBeans IDE es un ambiente de desarrollo integrado de código abierto y gratuito, el cual permite desarrollar aplicaciones de escritorio, móviles y webs. El IDE permite el desarrollo de aplicaciones en diferentes lenguajes, entre ellos Java, PHP y C++. Proporciona soporte para todo el ciclo de desarrollo, desde la creación del proyecto hasta la depuración e implementación.

3.3 Estudio de herramientas para el aprendizaje de programación

1) Scratch

De acuerdo con [29] Scratch cuenta con un sitio web oficial en donde su comunidad tiene la opción de subir sus proyectos además de ver proyectos desarrollados por otros usuarios, la interfaz de usuario de Scratch fue diseñada para hacer del codificar una tarea fácil para los programadores principiantes, en la pantalla principal del lado izquierdo cuenta con una paleta de bloques el cual contiene nueve secciones de bloques diferentes para su utilización en el área derecha en donde los bloques pueden ser arrastrados para generar código, denominada Área de Scripts.

Scratch es utilizado por personas de todas las edades, actualmente es muy utilizado para enseñar diferentes lenguajes de programación, de igual manera sirve para crear animaciones, juegos, entre otros. Hay miles de personas utilizando Scratch para sus proyectos, actualmente más de 150 países lo utilizan ya sea para enseñar matemáticas, computación, lenguajes, entre muchas otras opciones.

Scratch fue diseñado por Mitchel Resnick el 9 de mayo del 2012, su utilización es de manera gratuita y se dice que su programación está dirigida por eventos.

2) Snap!

Snap! También conocido como BYOB (*Build Your Own Blocks*, Construye tus propios bloques), es un lenguaje de programación el cual usa bloques que pueden ser arrastrados y soltados para crear funciones, es una reimplementación de Scratch, la primera versión de BYOB fue una modificación del código fuente de Scratch 1.4, hoy en día son dos programas separados completamente, aunque sigue utilizando una interfaz parecida a Scratch 1.4, sirve para introducir a jóvenes a la programación.

Originalmente fue desarrollado por Scratcher Jens, en la versión BYOB 3.0 se unió como codesarrollador Scrathcer bharvey, quien contribuyo al diseño, creación de bibliotecas, documentación y también brindo mentorías en línea para los nuevos usuarios [29].

3) Turtlestitch

Turtlestitch es una plataforma de bordado programable creada por Adrea Mayr-Stalder y Michael Aschauer. Es un micro mundo de programación el cual permite a sus usuarios aprender a programar mediante la exploración [30].

Turtlestitch se basó en un lenguaje de programación educativo denominado Snap! Su uso principal es generar patrones personalizados para máquinas de bordar. Es útil ya que los diseñadores tienen la oportunidad de experimentar con la decoración en los bordados, así como para introducir a sus usuarios a la programación con una salida tangible.

3.4 Requerimientos del IDE

Se presentó en [31] que un ambiente de desarrollo integrado se divide en dos fragmentos importantes: una parte encargada de estructurar el código y la segunda para compilar/ejecutar dicho código. Es común que los editores compartan conductas parecidas y de igual manera sucede con las partes encargadas de compilar el código. Una diferencia entre el *IDE* a realizar y los mostrados en el apartado 3.2 es que el propuesto está enfocado a enseñar a programar, de igual manera le proporcionara al usuario videos de tutoriales para generar figuras sencillas, al ser interactivo, contara con bloques de instrucciones las cuales

tendrán funcionalidad de "*drag and drop*" (arrastrar y soltar) para estructurar el código, el cual es un elemento encontrado en el estudio de la sección 3.3.

Ya que la *IDE* propuesto tiene el objetivo de enseñar a jóvenes, se consideraron las opciones de desarrollarse en un entorno Web o en un entorno de escritorio, al final se optó que la opción de escritorio era la ideal ya que es menos probable que una escuela pública tenga una buena conexión a Internet a que se tenga a la mano al menos una computadora de escritorio, de igual manera se planteó que en un futuro al enlazar la herramienta con una máquina de coser para generar los diseños desarrollados será más sencilla la interacción con un ambiente de escritorio que con un ambiente web, debido a que la herramienta será llevada a cabo en un ambiente de escritorio, se optó por utilizar la plataforma de aplicaciones cliente de código abierto JavaFx, ya que esta tecnología le brinda a la herramienta la opción de ser multiplataforma, lo que significa que podrá ser ejecutada en cualquier computadora sin importar su sistema operativo, además de brindar elementos gráficos enriquecidos.

En la siguiente sección se enlistan los requisitos funcionales y los no funcionales, marcados con las letras (AS) los obtenidos de la investigación del "Estudio de *IDEs* enfocados a generar productos de software" y con las letras (AA) los que provienen del "Estudio de herramientas para el aprendizaje de programación":

1) *Requisitos funcionales*

- Seleccionar archivo para su utilización (AS)
- Dar por terminado el programa (AS)
- Generar un respaldo del archivo en uso (AS)
- Generar un nuevo archivo (AS)
- Arrastrar y Soltar instrucciones (AA)
- Cancelar último movimiento (AS)
- Analizar código estructurado (AS)
- Generar diseño del código estructurado (AA)

- Generar diseño del código estructurado creando bordado (a futuro) (AA)
- Visualizar tutoriales (AA)

2) Requisitos no funcionales

- Desarrollo con ayuda de JavaFX
- Se le brindara al usuario una lista de acciones que se podrán llevar a cabo con la herramienta, las cuales podrán ser arrastradas y soltadas en un área de edición. (AA)
- Se mostrará el diseño resultante en otra sección de la herramienta (AA)
- Conectividad con una máquina de coser o bordar (a futuro) (AA)

Ya que el ambiente de desarrollo integrado será interactivo y se planea arrastrar y soltar las instrucciones para estructurarlas en un área de edición, se descartaron las funcionalidades de copiar y pegar que brindan los ambientes de desarrollo integrado destinados a la generación de productos de software.

Para el requisito funcional nombrado “Visualizar tutoriales” se tomaron en cuanta la generación de figuras y patrones sencillas (un cuadrado, un hexágono, una greca cuadrada, una greca en zigzag) para que el usuario aprenda a utilizar la herramienta y posteriormente pueda realizar sus propios diseños.

3.5 Diagrama de casos de uso

Una vez obtenidos los requisitos funcionales y no funcionales de la herramienta se procedió a realizar el diagrama de casos de uso, el cual se puede apreciar en Fig.3.5.1 obtenida del artículo [32]; en el diagrama se muestra “usuario” como nombre por defecto haciendo referencia a los niños, niñas y jóvenes que harán uso de la herramienta.

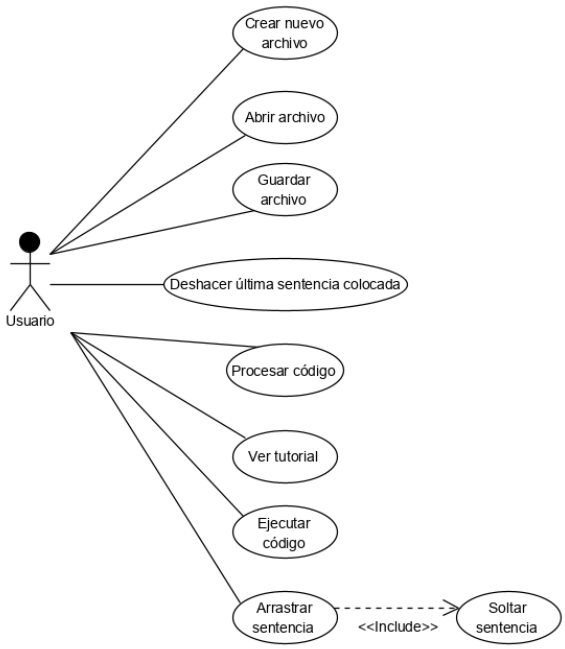

Figura 3. 5. 1 - Diagrama de casos de uso [30]

3.6 Diseño del IDE

Una vez teniendo en mente lo que la herramienta debe realizar se procedió a realizar el diseño preliminar del IDE tomando en cuenta los puntos obtenidos en los requisitos funcionales y no funcionales. El diseño se puede apreciar en la Fig. 3.6.1 obtenida del artículo [32].

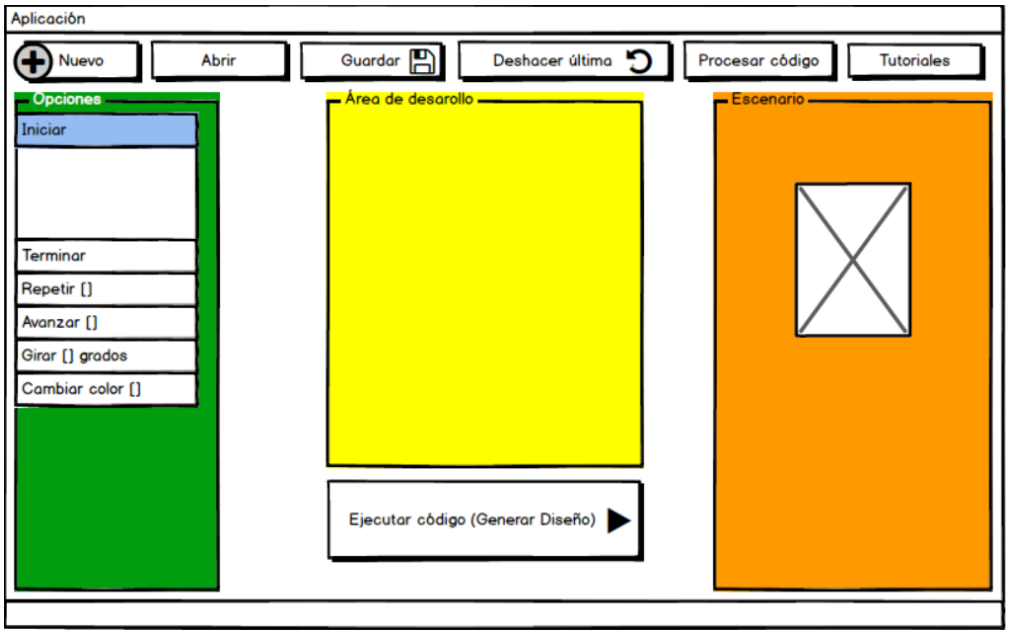

Figura 3. 6. 1 - Diseño del IDE [30]

Barra de herramientas: Se encuentra en la parte superior de color blanco, es en donde se encuentran posicionados la mayoría de los botones para interactuar con la herramienta.

Nuevo: Este botón se encargará de generar un nuevo archivo vacío, en caso de tener código estructurado en el área de desarrollo este se perderá si no se guardó antes de generar un nuevo archivo, al ser presionado le preguntará al usuario si desea continuar para evitar borrar lo estructurado por error.

Abrir: Este botón tendrá la funcionalidad de brindarle al usuario la opción de seleccionar un archivo guardado con anterioridad para volver a estructurar el código en la sección denominada área de desarrollo, al ser presionado abrirá una ventana aparte para seleccionar dicho archivo.

Guardar: Este botón servirá para respaldar lo estructurado en la sección de área de desarrollo, al presionarlo se abrirá una nueva ventana con el explorador de archivos para que el usuario seleccione el lugar en donde desea hacer el respaldo al igual que darle un nombre al archivo a respaldar.

Deshacer Última: Este botón se encargará de cancelar la última acción realizada en el área de desarrollo, ya sea quitar la última instrucción arrastrada a esta sección desde la sección de opciones mostradas o volver un paso atrás las sentencias movidas dentro de la misma área de desarrollo.

Procesar Código: Este botón tendrá la funcionalidad de mandar a analizar el código estructurado por el usuario en el área de desarrollo para determinar si está correcto, de igual manera saldrá una ventana en donde se le indicara al usuario si es correcto o tiene errores.

Tutoriales: Este botón se encargará de abrir una ventana no modal para que el usuario visualice videos para generar figuras o patrones, para ver el funcionamiento de la herramienta.

Menú de opciones: Se encuentra en la sección izquierda de color verde, en ella se encontrarán enlistadas las diferentes instrucciones que el usuario podrá utilizar para generar alguna figura o patrón, arrastrándolas desde esta sección hasta la parte denominada área de desarrollo.

Área de Desarrollo: Se encuentra en la parte central de color amarillo, en este apartado el usuario arrastrará las instrucciones deseadas desde el menú de opciones para generar alguna figura o patrón.

Ejecutar Código (Generar diseño): Una vez que el código ha sido procesado y se muestre que es correcto, se habilitara este botón para así al presionarlo generar el diseño en la parte de Escenario.

Escenario: Posicionado del lado derecho de color anaranjado, este apartado es en donde se mostrará al usuario el diseño resultante.

3.7 Arquitectura

Se le conoce como arquitectura al modelado de un sistema de software, en donde se ven reflejados los diferentes elementos que lo comprenden, sus propiedades y las relaciones que existen entre estos. [33]

Robert Martin [34], indica que una arquitectura de software debe mostrar cual es el objetivo principal del mismo. Pone como ejemplo los planos de una casa, en donde uno ve los diferentes elementos que la conforman como los cuartos, cocheras, sala, baños, patio, entre otros, así uno se da cuenta de lo necesario para construirla, de la misma forma al tener presente una arquitectura de software, se debe dar a entender para que sirve cada elemento y lo necesario para su creación.

Las buenas prácticas para desarrollar productos de software nos indican que es mejor dividir el código en módulos individuales, es más eficaz ya que se tiene el conocimiento de que hace cada cosa en lugar de buscar entre un código que se encargue de todo.

Se le conoce como módulo a cada una de las piezas que conforman el producto de software, el cual brinda una o más funcionalidades al producto final. El número de módulos necesarios para desarrollar el producto final depende de cada programador o equipo de programadores.

De acuerdo con G. Meyers [35], “La modularidad es el único atributo del software que permite gestionar un programa intelectualmente, un software monolítico (programa grande formado por un único modulo) no puede ser entendido fácilmente por el lector”.

3.7.1 Diagrama de componentes

Después de obtener los requisitos funcionales y no funcionales de la herramienta a desarrollar y generar el diseño preliminar de ésta, el siguiente paso fue plantear la arquitectura con los módulos necesarios para echar a andar el producto final, los cuales se pueden apreciar en la Fig. 3.7.1; los módulos sombreados son los que se encargan del funcionamiento principal de la herramienta, refiriéndose al funcionamiento de los botones, las ventanas, imágenes, entre otros, y de los módulos sin sombrear son los encargados del análisis del código abarcando el lenguaje de programación a generar así como el de desarrollar el diseño resultante o pintado en el escenario.

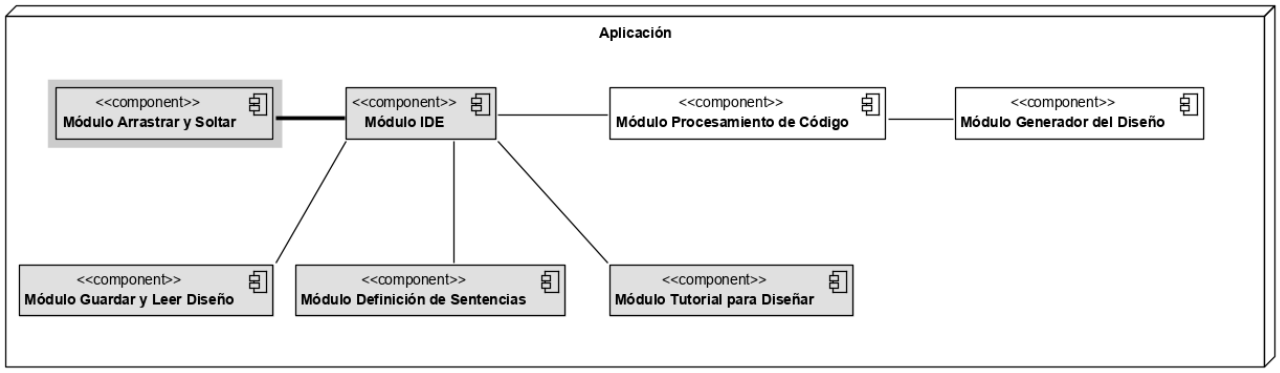

Figura 3. 7. 1 - Arquitectura de la aplicación

- *IDE*. El *IDE*, visto como la pantalla principal, está conformado por diferentes módulos que le brindan la funcionalidad total, los cuales se describen a continuación:
- *Definición de Sentencias.* Es el módulo en donde se declararán las instrucciones que el usuario utilizara para estructurar el código y así generar un diseño, se utilizaran imágenes que le brinden al usuario lo que hace cada instrucción.

- *Arrastrar y Soltar.* Es el módulo que le proporcionara la habilidad de arrastrar y soltar a las instrucciones dentro del *IDE*, lo que le brindara al usuario la habilidad de estructurar el código, así como reposicionarlo en caso de que lo haya puesto en un orden no valido.

- *Tutorial para Diseñar*. Es el módulo encargado del funcionamiento de la ventana de tutoriales en donde el usuario podrá visualizar diferentes videos para observar el funcionamiento de la herramienta para así más adelante generar sus propios diseños.

- *Guardar y Leer Diseño*. Es el módulo encargado de respaldar el código estructurado por el usuario en el área de desarrollo para poder seguir utilizando en otra ocasión, de igual manera será el encargado de abrir un archivo ya respaldado y reestructurarlo en el área de desarrollo para seguir trabajando en él.

- *Procesamiento de Código*. Es el módulo encargado de analizar que el código estructurado por el usuario se encuentre correcto para así poder ejecutarlo para generar el diseño en el escenario, en caso contrario le mostrara al usuario los errores generados en el código.

- *Generador del Diseño*. Es el módulo encargado de transformar el código estructurado por el usuario en el diseño y plasmarlo en el área de escenario.

3.7.2 Diagrama de Paquetes

En UML se le conoce como paquete al elemento encargado de agrupar los componentes que conforman el producto de software, lo que les brinda a los programadores la opción de organizar los componentes modelados, lo que facilita la lectura de estos.

Permiten dividir un proyecto para estructurar sus elementos de forma individual y encapsular sus atributos, los paquetes tienen la opción de contener más paquetes dentro de ellos, así como unos depender de otros, usualmente se utilizan para estructurar la

arquitectura del producto en una posición macro [36], los paquetes facilitan la visibilidad de los elementos en proyectos amplios. Generalmente se utilizan cuando un diagrama modelado con UML ya no puede visualizarse en una hoja tamaño carta.

A continuación, se presenta el diagrama de paquetes que encapsula los paquetes que componen la aplicación, en la Figura 3.7.2 Así como los diagramas de paquetes con sus componentes y asociaciones en las Figuras 3.7.2.1, 3.7.2.2, 3.7.2.3, 3.7.2.4, 3.7.2.5 y 3.7.2.6.

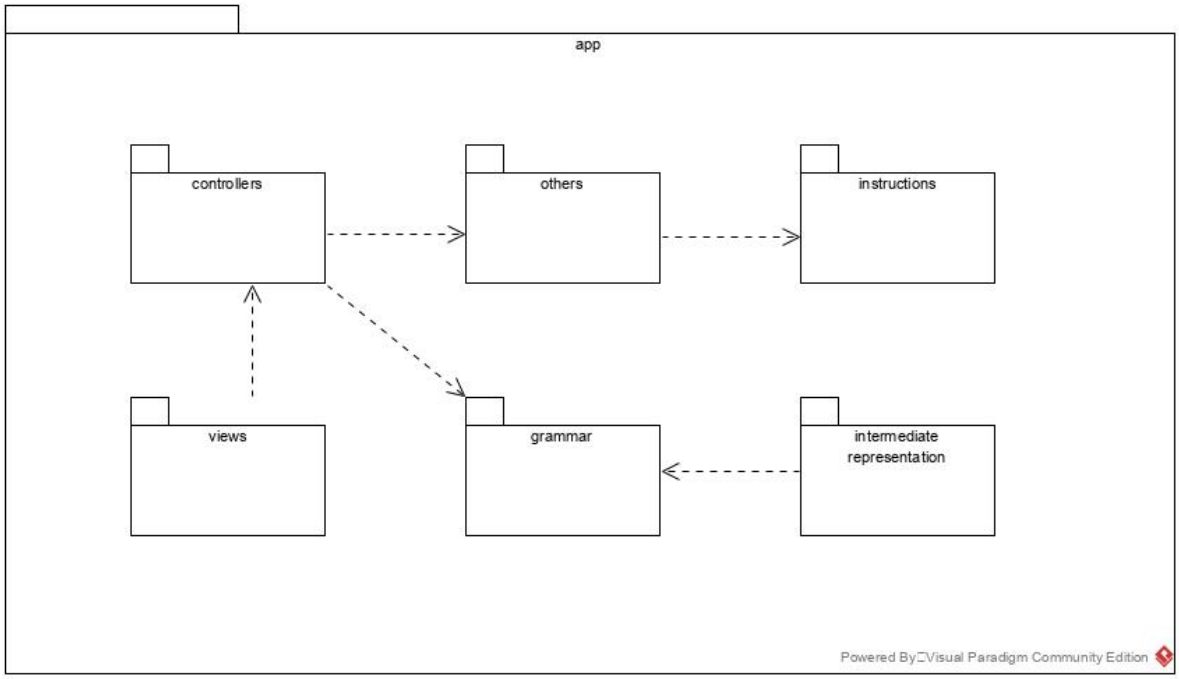

Figura 3. 7. 2 - Diagrama de paquetes

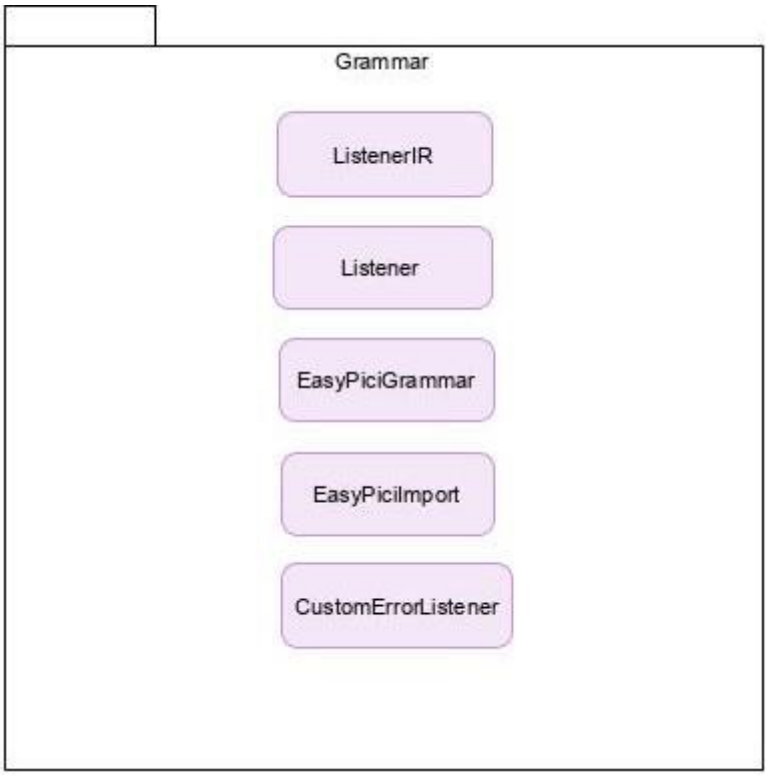

Figura 3. 7. 2. 2 - Paquete Grammar

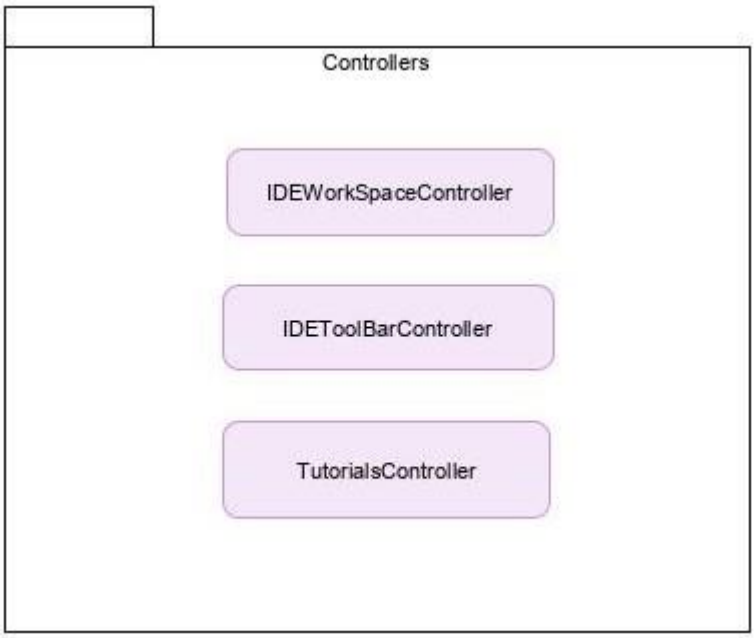

Figura 3. 7. 2. 1 - Paquete Controllers

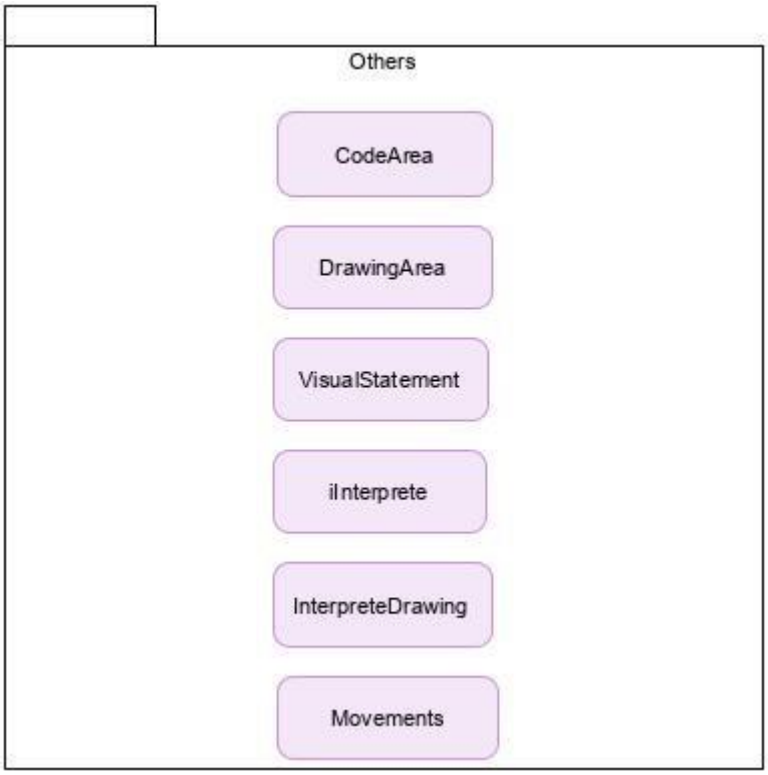

Figura 3. 7. 2. 3 - Paquete Others

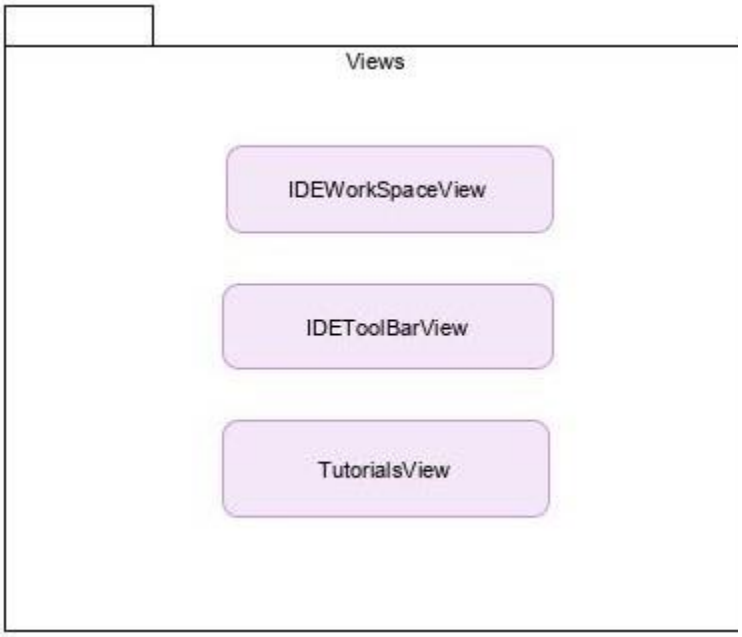

Figura 3. 7. 2. 5 - Paquete Views

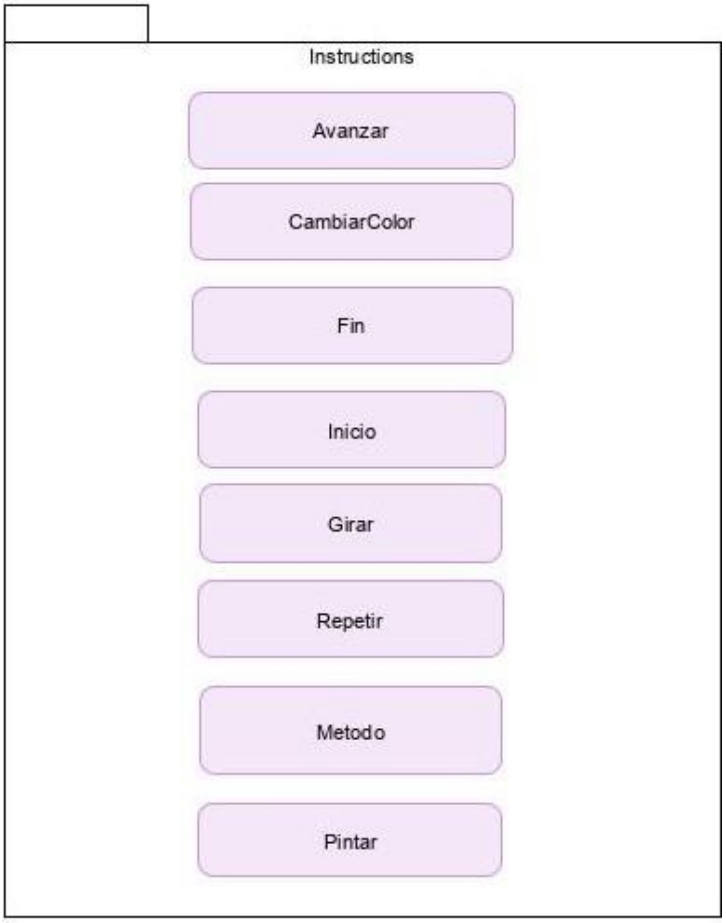

Figura 3. 7. 2. 4 - Instructions

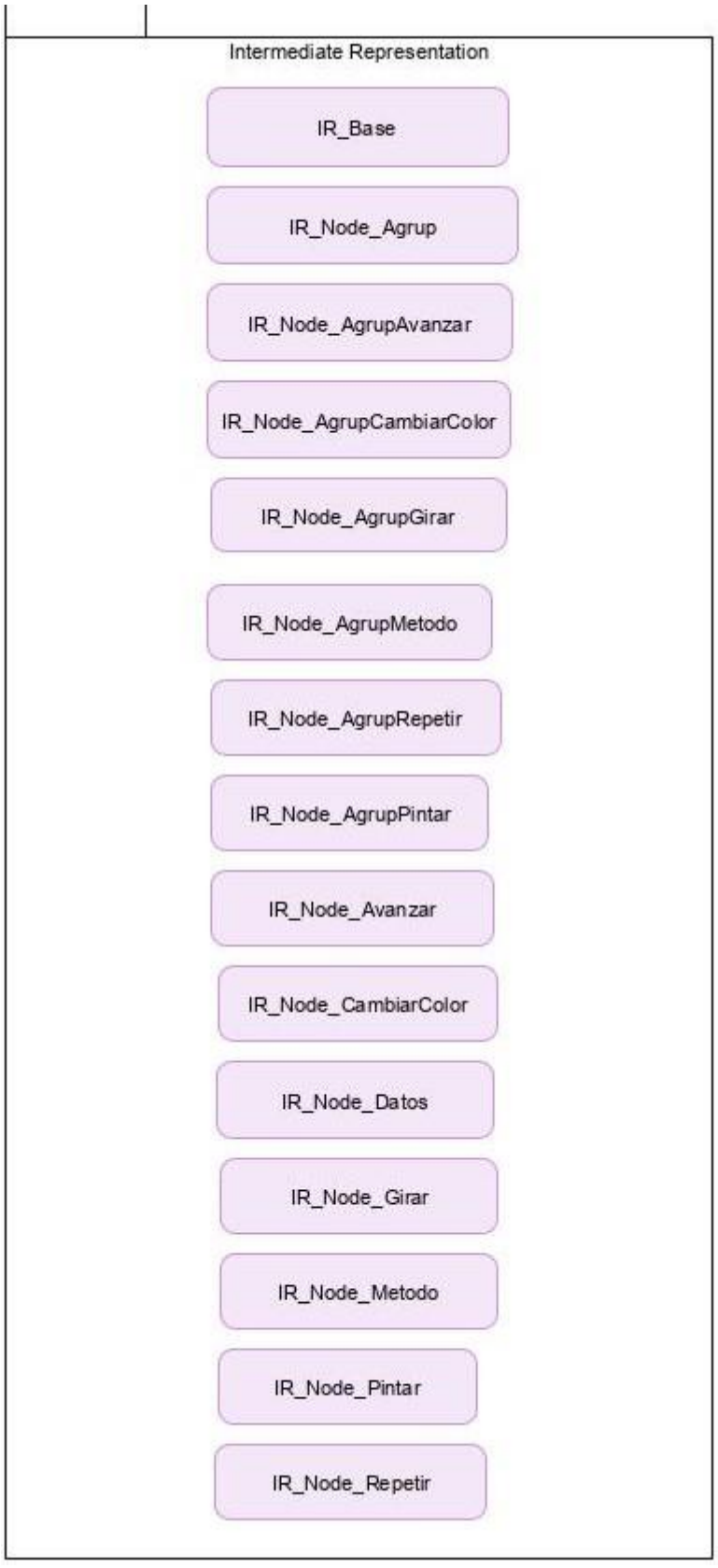

Figura 3. 7. 2. 6 - Paquete Intermediate Representation

3.8 Diagrama de clases para el IDE

Haciendo énfasis en los elementos sombreados de la arquitectura de la aplicación planteada, aquellos encargados del funcionamiento principal del ambiente de desarrollo integrado, se generó el diagrama de clases preliminar del *IDE*, el cual se puede apreciar en la Fig. 3.8.1 el cual fue compartido en el artículo [37]. Dado que se optó por la utilización de JavaFx para el desarrollo de la herramienta, se marca con un estereotipo <<JavaFx>> a aquellos elementos particulares de plataforma, para las vistas utilizadas en el *IDE* se usa el estereotipo <<fxml>> las cuales son codificadas en FXML (JavaFX *Markup Language*), aquellas clases que carecen de un estereotipo son aquellas que deben ser programadas en Java respectivamente.

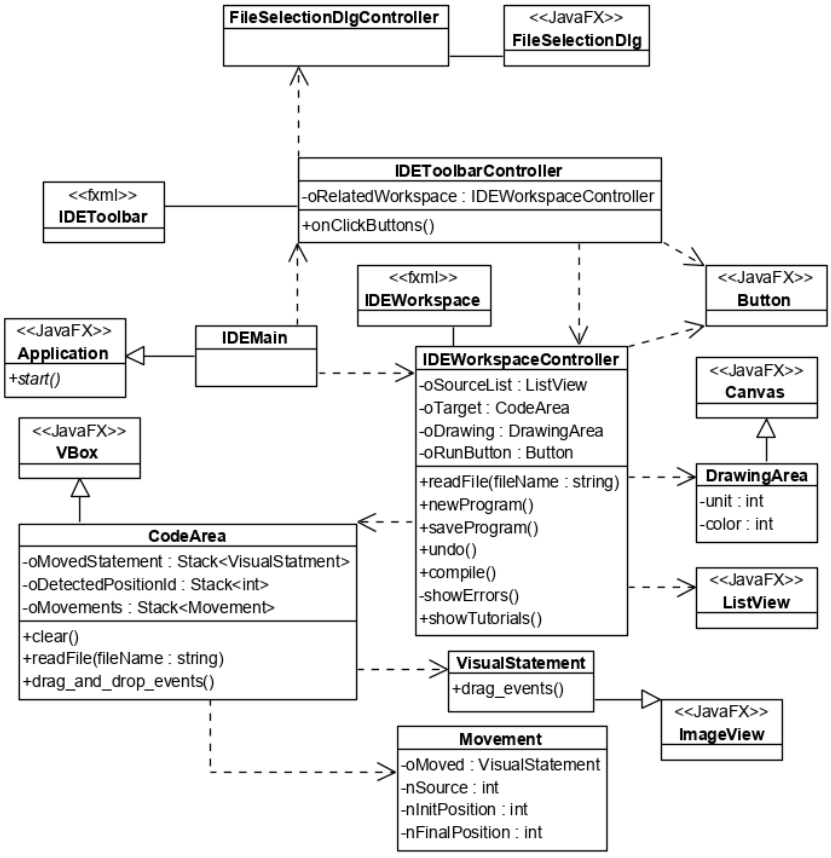

Figura 3. 8. 1 - Diagrama de clases del IDE [35]

- *IDEMain*: Es la clase de inicio del *IDE*, hereda de la clase `Application`, propia de JavaFx, para hacer uso del método `start` he inicializar tanto la vista como el controlador del `IDEWorkSpace` y del `IDEToolbar`.
- *IDEToolbar*: Es la vista de la barra de herramientas.
- *IDEToolbarController*: Esta clase se encarga de brindarle funcionalidad a los componentes que se encuentran dentro de la barra de herramientas.
- *FileSelectionDlg*: Es la vista para la exploración de archivos, utilizada para respaldar y seleccionar archivos.
- *FileSelectionDlgController*: Esta clase es la encargada del funcionamiento de la vista para respaldar y leer archivos.
- *IDEWorkspace*: Es la vista del espacio de trabajo del *IDE*, tanto el menú de opciones, el área de codificación y el escenario.
- *IDEWorkspaceController*: Esta clase es responsable de brindarle funcionalidad a las diferentes secciones del espacio de trabajo, como lo son: el menú de opciones en donde se incluyen las instrucciones a utilizar por el usuario, el área de codificación en donde se estructurará el código, el escenario en donde se visualizará el diseño generado y el botón de ejecutar para que se lleve a cabo el proceso de generar el diseño.
- *CodeArea*: Es la clase que representa el "programa" creado a partir de colocar sentencias (`VisualStatement`), arrastrándolas de una lista y soltándolas en este espacio o bien cambiando la posición de estas sentencias ahí mismo, soportando una distribución vertical de elementos (`VBox`); por lo tanto, esta clase reacciona a los eventos de arrastrar y soltar del ratón. Se auxilia de una serie de pilas para conocer en qué orden se han colocado las sentencias, qué sentencia se está moviendo y en qué posición se desea colocarla.
- *VisualStatement*: Esta clase hereda de `ImageView`, es la encargada de reproducir cada una de las instrucciones a utilizar por el usuario, posee la

funcionalidad de *drag and drop*, cada instrucción hereda de ella para asignar las características particulares del lenguaje a desarrollar.

- *Movement*: Esta clase se encarga de guardar los movimientos de posición de una instrucción, de igual manera almacena información para conocer si se movió desde el menú de opciones o si se reposicionó en la sección del área de codificación; es una clase que ayuda a brindar la funcionalidad del requisito "Cancelar último movimiento".

3.9 Prototipos Primera Versión del IDE

Para desarrollar la herramienta, se empezó por crear prototipos que ayudaron a entender mejor el funcionamiento de cada tecnología que se utilizaría para el desarrollo de esta:

3.9.1 Prototipo 1

Como primer prototipo se desarrolló un proyecto en donde se utilizaron los elementos gráficos que ofrece JavaFX para implementar la distribución descrita en el maquetado y darles funcionalidad posteriormente.

Se asignaron imágenes a la lista y se implementó el *drag and drop* a partir de ella; si bien el movimiento se realiza, las imágenes no se ordenan en el espacio destino (Figura 3.9.1).

Figura 3. 9. 1 - Prototipo uno implementación de drag and drop

3.9.2 Prototipo 2

En el segundo prototipo se incrementó el uso de la funcionalidad *drag and drop*, básicamente fue el ordenamiento en el espacio de destino, con la ayuda de una clase de distribución tipo `VBox` como se visualiza en la figura 3.9.2:

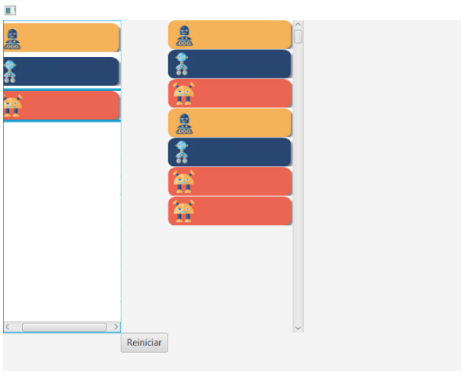

Figura 3. 9. 2 - Prototipo dos, segunda prueba de drag and drop

3.9.3 Prototipo 3

En este prototipo se implementó la funcionalidad de `Canvas` (espacio de pintado) con JavaFX, se generó una cuadricula y se probó el uso de líneas para pintarlas sobre la cuadricula (figura 3.9.3), esta misma será integrada dentro de la herramienta.

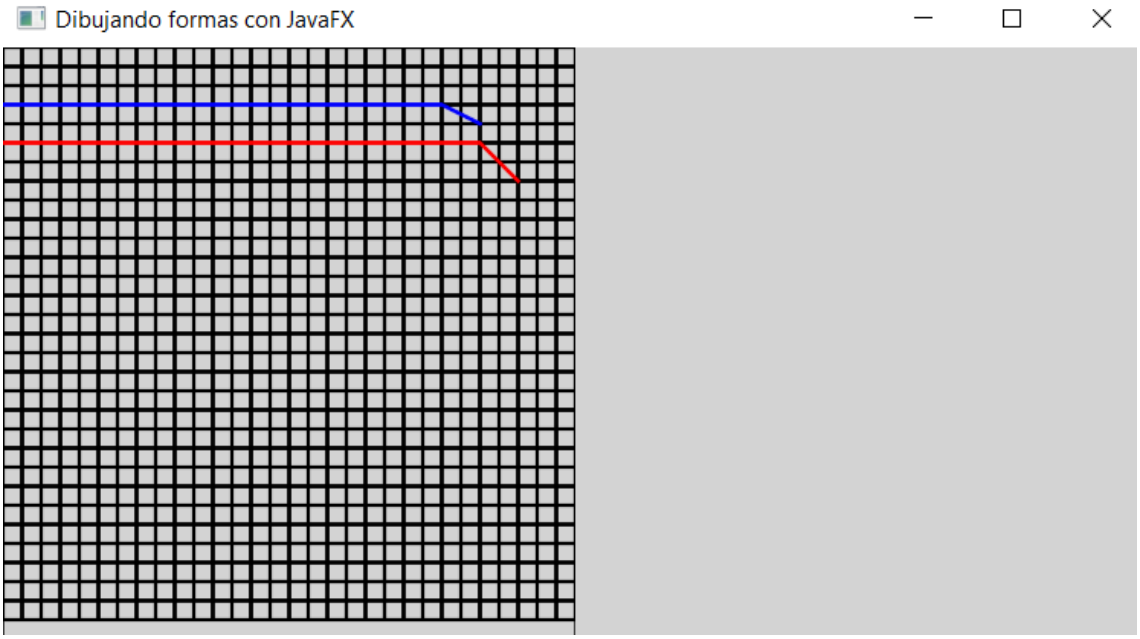

Figura 3. 9. 3 - Prototipo tres con primera prueba del pintado en canvas

3.9.4 Prototipo 4

Se creó la pantalla del *IDE* en JavaFX (figura 3.9.4) siguiendo el maquetado descrito en la sección Diseño del *IDE* para posteriormente darle funcionalidad; también se creó la clase `VisualStatement` que hereda de la clase `ImageView`, propia de JavaFX, para permitir que las instrucciones del lenguaje tengan una representación visual en el *IDE* (lado izquierdo de la figura).

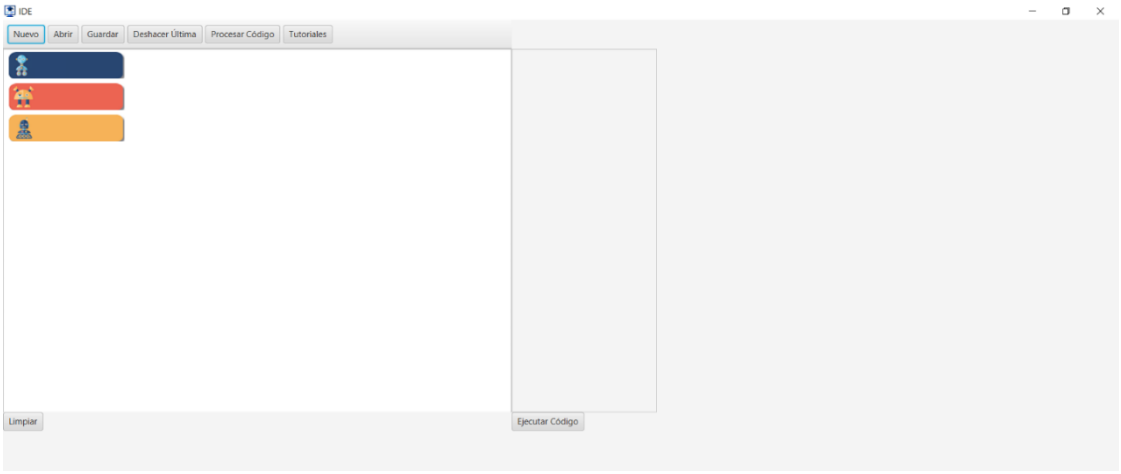

Figura 3. 9. 4 - Prototipo cuatro con primer diseño del FXML y las imágenes creadas con la clase visualStatement

Tabla 3. 9. 4. 1 - Tabla de clases para el prototipo 4

Clases construidas	Objetivo
VisualStatement	Clase que hereda de `ImageView` para asignarle una imagen de instrucción y brindarle funcionalidad *drag and drop*

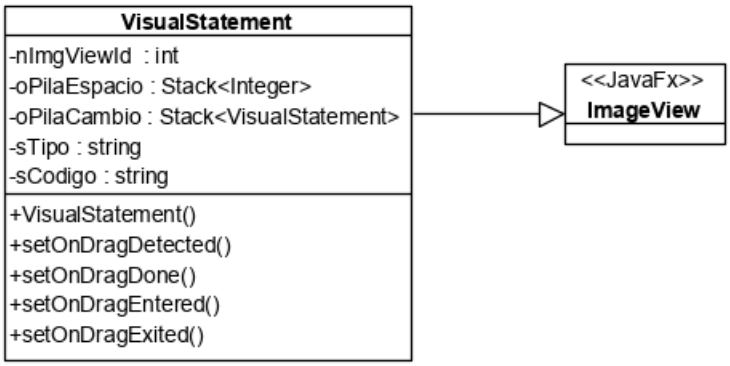

Figura 3. 9. 4. 1 - Diagrama de la clase VisualStatement

3.9.5 Prototipo 5

Se integró la primera versión funcional del IDE con ayuda de los prototipos desarrollados anteriormente, en donde se muestra la pantalla con la distribución esperada incluyendo botones e íconos, se inicializa la lista de sentencias con imágenes de prueba, mismas que se pueden arrastrar hacia la parte central (área de desarrollo) como se visualiza en la Figura 3.9.5.

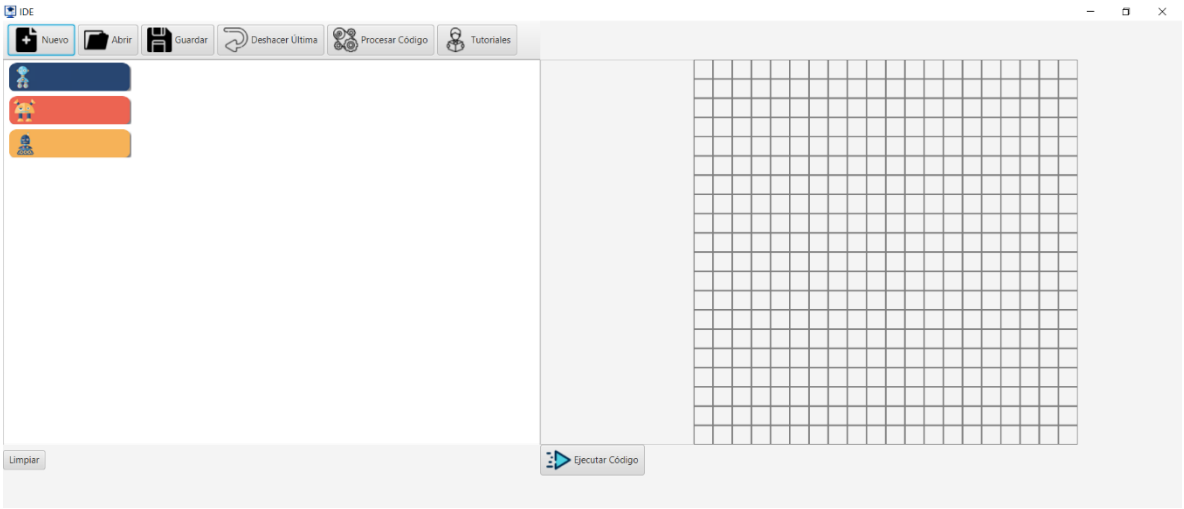

Figura 3. 9. 5 - Prototipo cinco con la primera versión del IDE (funcionalidad drag and drop, y pintado en canvas)

Tabla 3. 9. 5. 1 - Tabla de las clases para el prototipo 5

Clases construidas	Objetivo
Movement	**Esta clase fue creada para guardar la información de la imagen o instrucción movida, guarda la imagen, en donde estaba y en donde terminó.**
CodeArea	**Esta clase tiene como objetivo el contener las instrucciones arrastradas dentro del área de desarrollo.**
DrawingArea	**Esta clase se desarrolló para dibujar la cuadricula con ayudas de `Canvas`.**

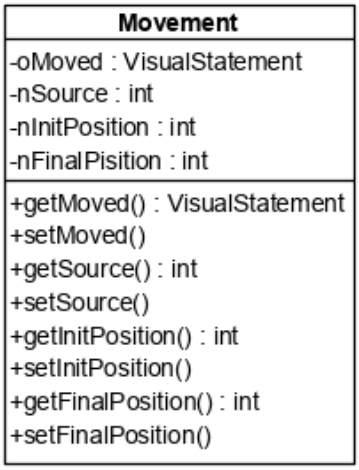

Figura 3. 9. 5. 3 - Diagrama de la clase Movement

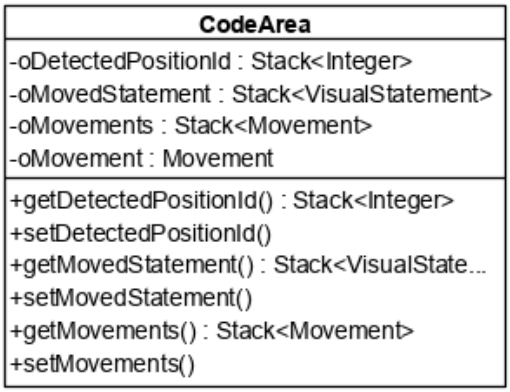

Figura 3. 9. 5. 2 - Diagrama de la clase CodeArea

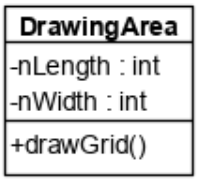

Figura 3. 9. 5. 1 - Diagrama de la clase DrawingArea

3.9.6 Prototipo 6

En este punto se le dio funcionalidad al botón de Ejecutar, el cual dibuja unas líneas en la cuadricula como primera prueba; también se le dio funcionalidad al botón <Deshacer última>, el cual deshace el último movimiento hecho con las imágenes de instrucciones en el área de desarrollo; finalmente, se asignaron botones para cambiar el tamaño de la cuadricula en cm, mm y pulgadas (figura 3.9.6).

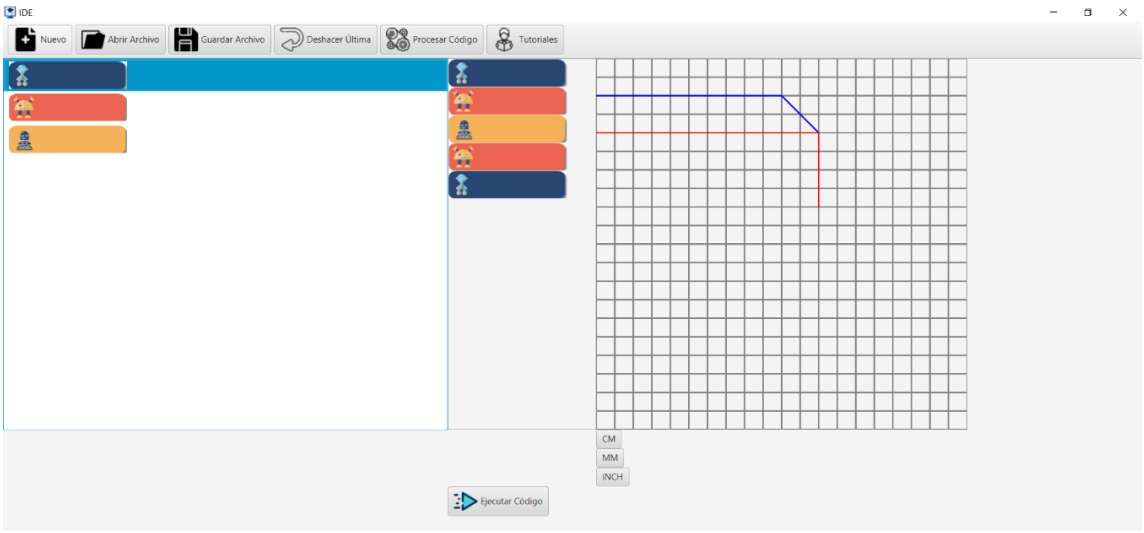

Figura 3. 9. 6 - Prototipo seis con funcionalidad del botón deshacer ultima y botón ejecutar, cambio de tamaños de la cuadricula

Tabla 3. 9. 6. 1 - Tabla de las clases para el prototipo 6

Clases construidas	Objetivo
CodeArea	**Se modifico para agregar el método *undo* el cuál es el encargado de la funcionalidad del botón deshacer ultima.**
IDEWorkSpaceController	**Es la clase encargada de la funcionalidad de la parte central del IDE.**

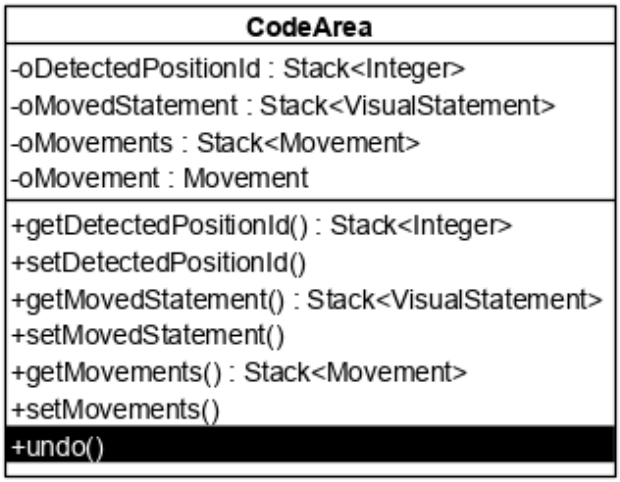

Figura 3. 9. 6. 2 - Diagrama de la clase CodeArea

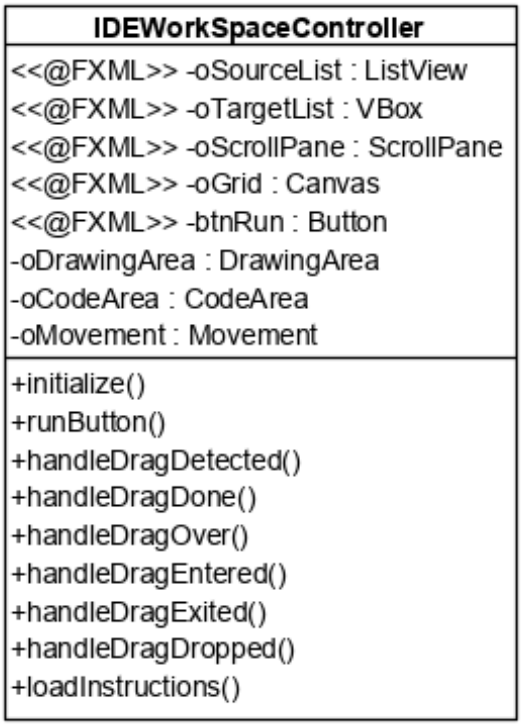

Figura 3. 9. 6. 1 - Diagrama de la clase IDEWorkSpaceController

3.9.7 Prototipo 7

En esta versión del *IDE* se dio funcionalidad a los botones <Nuevo>, el cual por el momento pregunta si se desea crear un nuevo proyecto y limpia lo que hay en pantalla si es que hay algo (figura 3.9.7.2); también se dio funcionalidad al botón <Abrir archivo>, por el momento abre documentos con extensión .txt y muestra el nombre (figura 3.9.7.1); por último, el botón <Guardar archivo> permite almacenar el archivo correspondiente al programa realizado con los estatutos visuales (figura 3.9.7.3).

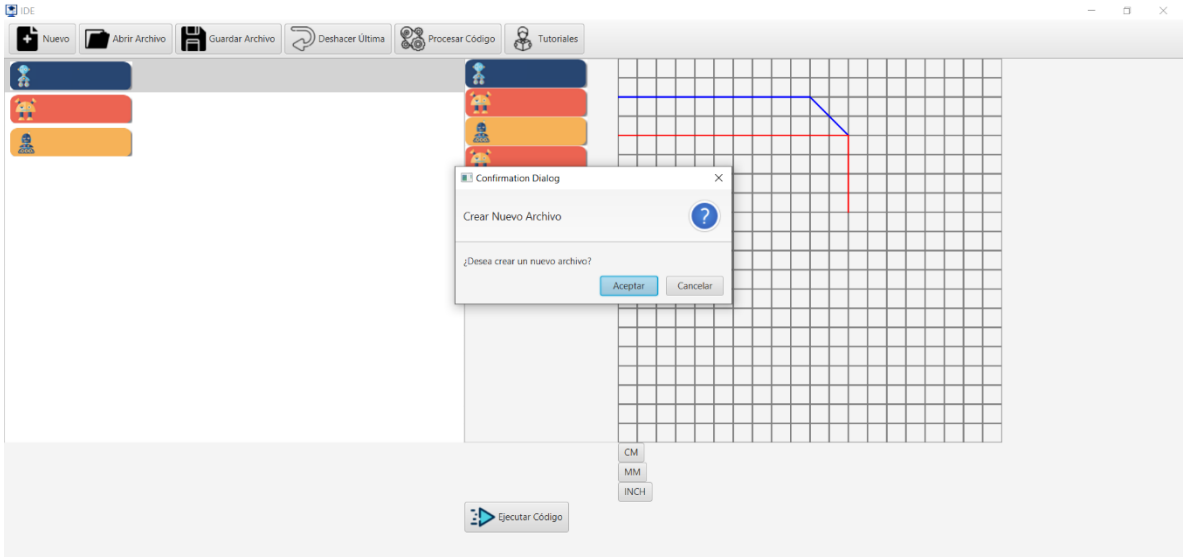

Figura 3. 9. 7. 1 - Prototipo Siete con Funcionalidad botón nuevo

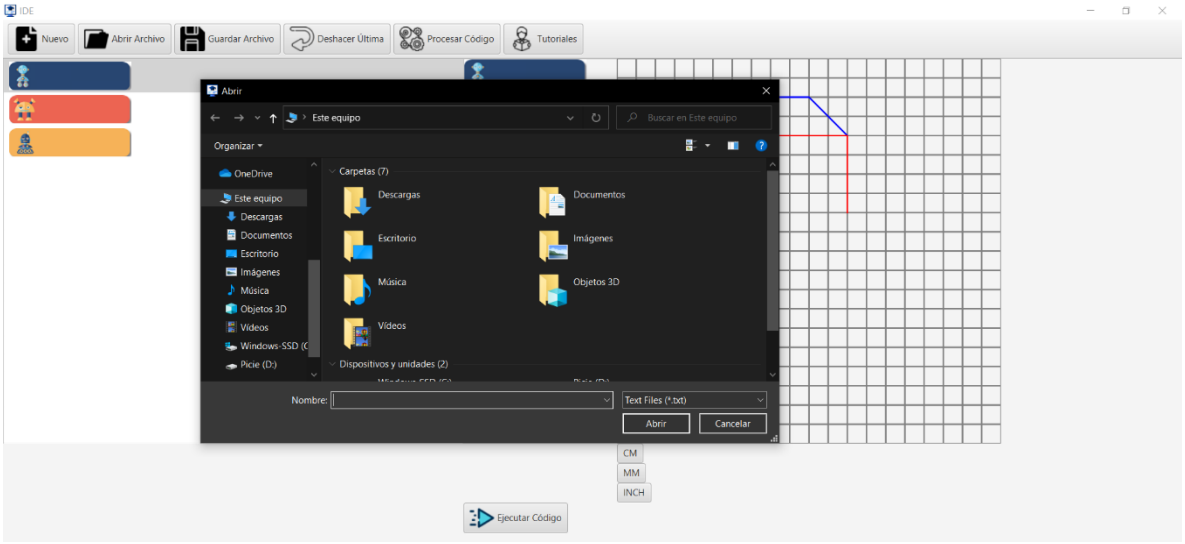

Figura 3. 9. 7. 2 - Prototipo Siete con Funcionalidad botón abrir archivo

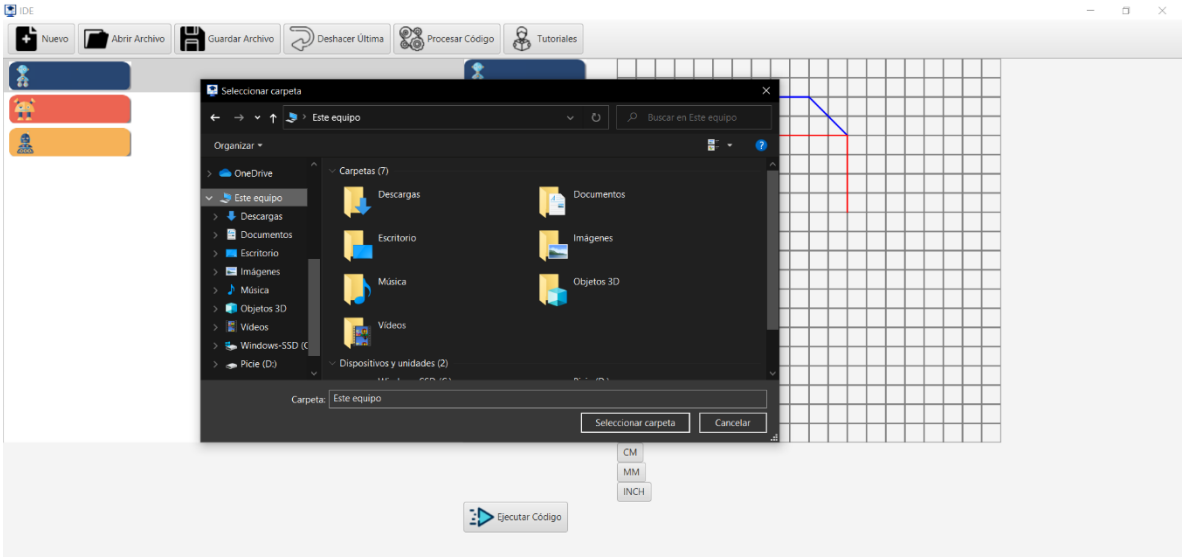

Figura 3. 9. 7. 3 - Prototipo Siete con Funcionalidad botón guardar archivo

Tabla 3. 9. 7. 1 - Tabla de las clases para el prototipo 7

Clases construidas	Objetivo
IDEToolBarController	**Es la clase encargada de brindarle funcionalidad a los botones de la barra de herramientas situada en la sección superior del IDE (Abrir, Guardar, Nuevo, Deshacer Última, Procesar, Tutoriales).**

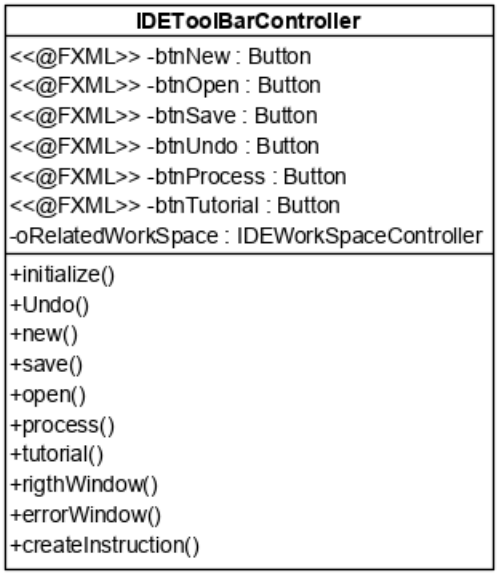

Figura 3. 9. 7. 6 - Diagrama de la clase IDEToolBarController

3.10 Requerimientos del Lenguaje

En paralelo al desarrollo del *IDE*, se creó el lenguaje de programación para crear patrones geométricos. Como se indicó en el marco de referencia, Lenguaje es el conjunto de unidades de léxico, reglas sintácticas y semánticas que permiten la comunicación entre dos entidades.

Para describir los requerimientos del lenguaje se realizaron varias etapas que se describen en los siguientes puntos.

3.10.1 Ejemplo de frases soportadas por el lenguaje

En primer lugar, se escribieron los tipos de frases esperadas del lenguaje como se muestra en los listados 3.10.1.1, 3.10.1.2 y 3.10.1.3.

Lista 3. 10. 1. 1 - Código esperado para pintar una línea

Línea	Contenido
1.	Metodo HacerUnaLinea
2.	Inicio
3.	Girar 90 grados;
4.	Avanzar 5 unidades;
5.	Girar 270 grados;
6.	Pintar 20 unidades;
7.	Fin

Lista 3. 10. 1. 2 - Código esperado para hacer un rectángulo de colores

Línea	Contenido
1.	Metodo HacerRectanguloColores
2.	Inicio
3.	Cambiar Color rojo;
4.	Coser 10 unidades;
5.	Girar 90 grados;
6.	Cambiar Color azul;
7.	Coser 5 unidades;
8.	Girar 90 grados;
9.	Cambiar Color verde;
10.	Coser 10 unidades;
11.	Girar 90 grados;
12.	Cambiar Color Naranja;
13.	Coser 5 unidades;
14.	Fin

Lista 3. 10. 1. 3 -Código esperado para hacer un cuadrado

Línea	Contenido
1.	Metodo HacerCuadrado
2.	Inicio
3.	Repetir 4 veces;
4.	Inicio
5.	Coser 5 unidades;
6.	Girar 90 grados;
7.	Fin
8.	Fin

3.10.2 Lexemas o *Tokens*

Una vez analizadas las posibles frases a utilizar planteadas en el punto anterior se consideraron los siguientes lexemas o *tokens* para el lenguaje de programación, como se pueden observar en la lista 3.10.2.1:

Lista 3. 10. 2. 1 - Tokens o Lexemas para el lenguaje

Palabra Reservada	Valor Asociado
PR_Avanzar	**Avanzar**
PR_Unidades	**Unidades**
PR_Pintar	**Pintar**
PR_Girar	**Girar**
PR_Grados	**Grados**
PR_Repetir	**Repetir**
PR_Cambiar	**Cambiar**
PR_Color	**Color**
PR_Metodo	**Metodo**
PR_Inicio	**Inicio**
PR_Fin	**Fin**
PR_Azul	**Azul**
PR_Rojo	**Rojo**
PR_Verde	**Verde**
PR_Negro	**Negro**
SG_PC	**;**
ID	**[A-z]+**
Numero	**[0-9]+**

3.10.3 Gramática

Una vez identificados los lexemas, fue necesario describir las reglas gramaticales para el nuevo lenguaje, tomando en consideración que el axioma u objetivo de la gramática es realizar un "método". A continuación, se listan las producciones de la gramática BNF resultante en la lista 3.10.3.1.

Lista 3. 10. 3. 1 - producciones para el lenguaje.

No Terminal	Regla
Metodo	**PR_Metodo ID PR_Inicio Instrucciones PR_Fin**
Instrucciones	**Avanzar | Pintar | Girar | Mientras | CambiarColor**
Avanzar	**PR_Avanzar Numero PR_Unidades SG_PC**
Pintar	**PR_Pintar Numero PR_Unidades SG_PC**
Girar	**PR_Girar Numero PR_Grados SG_PC**
Repetir	**PR_Repetir Numero PR_Inicio instrucciones PR_Fin**
CambiarColor	**PR_Cambiar PR_Color color SG_PC**
Color	**PR_Azul | PR_Rojo | PR_Verde | PR_Negro**

3.10.4 Semántica.

Las reglas gramaticales de un lenguaje no aseguran que se interpreten todas las frases correctamente, pues el contexto juega un papel importante y por eso se requieren validaciones semánticas. En el caso del lenguaje para programar patrones gráficos, se tomaron en consideración las siguientes validaciones semánticas:

-Verificar que se haya introducido ángulo correcto.

-Validar que el numero introducido sea correcto.

-Verificar que el color escogido sea correcto.

-Validar la correcta iniciación y finalización de instrucciones.

-Validar la correcta iniciación y finalización del método.

3.10.5 Características de la salida a obtener

Para la salida lo deseable es generar una greca o figura a partir de las instrucciones colocadas por el usuario, las cuales serán arrastradas al apartado llamado área de trabajo, las cuales se procesarán con el botón "procesar código", en caso de ser procesado sin errores se habilitará el botón de ejecutar, el cual al ser presionado se procesará el código de entrada y se mandará una instrucción de Java específica para cada caso, para generar un `Canvas` y dibujar en él.

3.10.6 Desarrollo de la aplicación de lenguaje

A continuación, se explican los pasos llevados a cabo para el desarrollo de la aplicación del lenguaje.

3.10.6.1 Analizador Léxico

Después de realizar los análisis de las frases soportadas en la sección 3.10.1 y la sección 3.10.2 se procedió a realizar el analizador léxico con la ayuda de ANTLR4, se creó como primera instancia el analizador léxico, el cual se muestra en la lista 3.10.6.1 , para esto en la primera línea se debe colocar "`Lexer grammar`", para indicar que se trata de un analizador léxico, seguido del nombre que se le desea asignar, después se van colocando en una línea diferente cada uno de los Lexemas que se definieron antes en el apartado 3.10.1 de la siguiente manera: "Palabra Reservada" seguido de dos puntos ":", espacio, entre comillas su valor asociado y punto y coma ";" para finalizar el *token*.

Lista 3. 10. 6. 1- Código para el Analizador Léxico

Línea	Contenido
1.	Lexer grammar EasyPiciGramatica;
2.	PR_Avanzar: 'Avanzar';
3.	PR_Unidades: 'Unidades';
4.	PR_Pintar: 'Pintar';
5.	PR_Girar: 'Girar';
6.	PR_Grados: 'Grados';
7.	PR_Repetir: 'Repetir';
8.	PR_Cambiar: 'Cambiar';
9.	PR_Color: 'Color';
10.	PR_Metodo: 'Metodo';
11.	PR_Inicio: 'Inicio';

12.	PR_Fin: ‘Fin’;
13.	PR_Azul: ‘Azul’;
14.	PR_Rojo: ‘Rojo’;
15.	PR_Verde: ‘Verde’;
16.	PR_Negro: ‘Negro’;
17.	PR_Cafe: ‘Cafe’;
18.	PR_Naranja: ‘Naranja’;
19.	ID: [A-z]+;
20.	Numero: [0-9]+;
21.	SG_PC: ‘;’;
22.	EB: [\t\r\n]+ -> skip;

3.10.6.2 Analizador Gramatical

Una vez definidos los *tokens* que se utilizarían, se desarrolló el analizador gramatical con ayuda de ANTLR4 y utilizando al analizador léxico generado anteriormente, quedando como se muestra en la lista 3.10.6.2.

Para empezar, se debe declarar que es un analizador gramatical, con la palabra “`grammar`” seguido del nombre que se le quiere poner y se finaliza con punto y coma “;” en la primera línea, en la segunda línea es necesario agregar el analizador léxico con la palabra “`import`” seguido del nombre del analizador léxico. Una vez definidos estos dos puntos se debe explicar la manera en que los componentes del lenguaje se enlazan para generar textos validos en la aplicación del lenguaje, para esto se hace uso de las producciones obtenidas en el apartado 3.10.3, se deben poner los no terminales seguido de dos puntos “:” y después la regla para finalizar con punto y coma “;”. Es importante destacar que no se realizó un analizador semántico ya que el usuario no es libre de escribir como tal en la herramienta, las instrucciones ya se encuentran predefinidas como imagen, y antes de procesarse con ayuda de java se analizan las reglas semánticas seleccionadas en el apartado 3.10.4 y algunas otras con las reglas del analizador gramatical.

Lista 3. 10. 6. 2 - Código para el Analizador Gramatical

Línea	Contenido
1.	grammar EasyPiciImport;
2.	import EasyPiciGramatica;
3.	metodo: PR_Metodo ID PR_Inicio instrucciones PR_Fin;
4.	instrucciones: (avanzar | pintar | girar | repetir | cambiar)+:
5.	avanzar: PR_Avanzar Numero PR_Unidades SG_PC;
6.	pintar: PR_Pintar Numero PR_Unidades SG_PC;

7.	girar: PR_Girar Numero PR_Grados SG_PC;
8.	repetir: PR_Repetir Numero PR_Inicio instrucciones PR_Fin;
9.	cambiar: PR_Cambiar PR_Color color SG_PC;
10.	color: PR_Azul | PR_Rojo | PR_Verde | PR_Negro | PR_Cafe | PR_Naranja;

3.11 Representación intermedia

A las primeras fases del compilador se le denomina *front-end*, como se puede observar en la Figura 3.11.1, esta incluye el analizador léxico y gramatical, una vez realizados estos análisis se puede generar una o más representaciones intermedias, para lograr esto se necesita de un mecanismo que nos ayude a acceder a la información en tiempo de ejecución del producto de software. La mayoría de los mecanismos utilizan los recorridos de los árboles sintácticos para esto. [5]

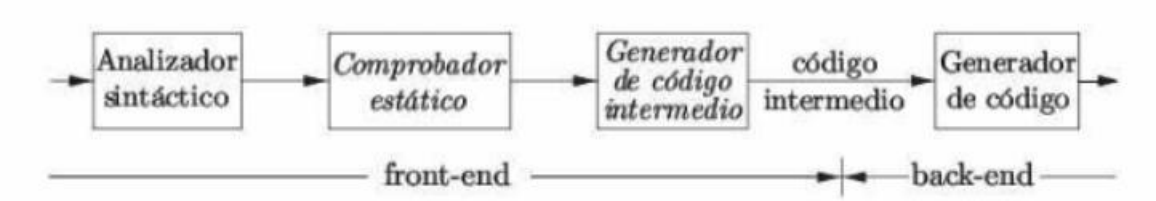

Figura 3. 11. 1 -Estructura lógica del front-end de un compilador [5]

En el transcurso de transformar el código fuente a el lenguaje destino se utilizan n representaciones intermedias como sean necesarias. Existen dos niveles para las representaciones intermedias, se les conoce como de alto nivel cuando se aproximan más al lenguaje del código fuente y de bajo nivel cuando se aproximan al lenguaje destino. Los AST son considerados de alto nivel ya que modelan los elementos en orden jerárquico del programa fuente y es posible utilizarlos para la validación de tipos. [5]

Figura 3. 11. 2 - Secuencia de representaciones intermedias [5]

Se utiliza una representación de bajo nivel cuando se desea inspeccionar las sentencias para obtener información necesaria y asignarla a ciertos registros. Otro de los elementos utilizados para las representaciones intermedias es el código en tres direcciones.

3.11.1 Código de tres direcciones

Básicamente el código de tres direcciones se utiliza en los compiladores para optimizar los procesos de transformación del código fuente y vuelve más sencillo el convertir el código al lenguaje destino. [5]

De acuerdo con la información presentada se puede decir que una representación intermedia es una representación neutra de los elementos relevantes del lenguaje origen, independiente del lenguaje de implementación esperado, que incluye características programables y elimina características que no cumplen o desempeñan una función.

Para la asignación de cada uno de los elementos necesarios por instrucción que serán utilizados para la implementación en java se requirió realizar una representación intermedia que agrupara cada uno de ellos, para esto se realizaron las siguientes clases:

Tabla 3. 11. 1 - Tabla de clases generadas para la representación intermedia

Clases construidas	Objetivo
IR_Node_Data	Clase abstracta de la que heredan las clases de los nodos de instrucciones.
IR_Node_Avanzar	Clase para obtener los datos de la instrucción avanzar.
IR_Node_CambiarColor	Clase para obtener los datos de la instrucción cambiar color.
IR_Node_Girar	Clase para obtener los datos de la instrucción girar.
IR_Node_Metodo	Clase para obtener los datos de la instrucción método.
IR_Node_Repetir	Clase para obtener los datos de la instrucción repetir.
IR_Node_Pintar	Clase para obtener los datos de la

	instrucción pintar.
IR_Node_Agrup	Clase abstracta para agrupar instrucciones.
IR_Node_AgrupAvanzar	Clase para agrupar las instrucciones avanzar.
IR_Node_AgrupCambiarColor	Clase para agrupar las instrucciones cambiar color
IR_Node_AgrupGirar	Clase para agrupar las instrucciones girar.
IR_Node_AgrupMetodo	Clase para agrupar las instrucciones método.
IR_Node_AgrupRepetir	Clase para agrupar las instrucciones repetir.
IR_Node_AgrupPintar	Clase para agrupar las instrucciones pintar
ListenerIR	Clase para obtener los datos en tiempo de ejecución.

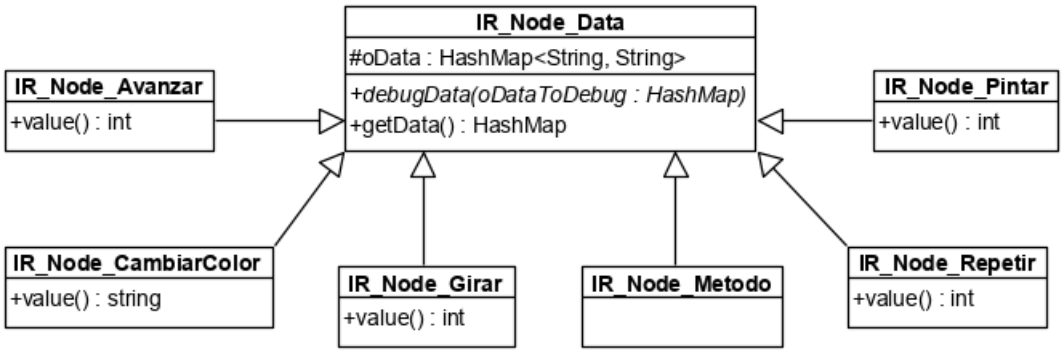

Figura 3. 11. 3 - Diagrama de los nodos para la representación intermedia

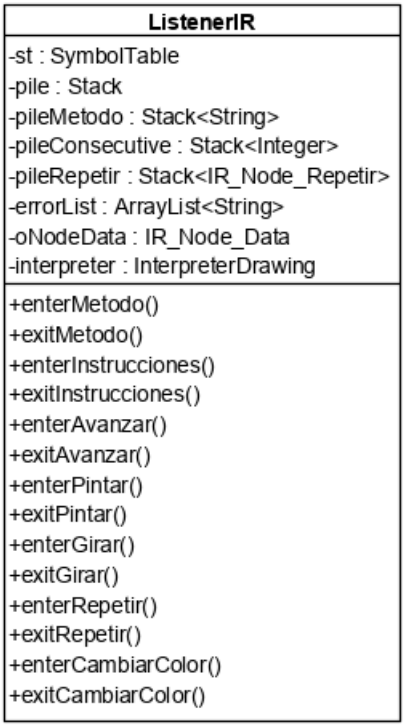

Figura 3. 11. 5 - Diagrama del escucha generado para la representación intermedia

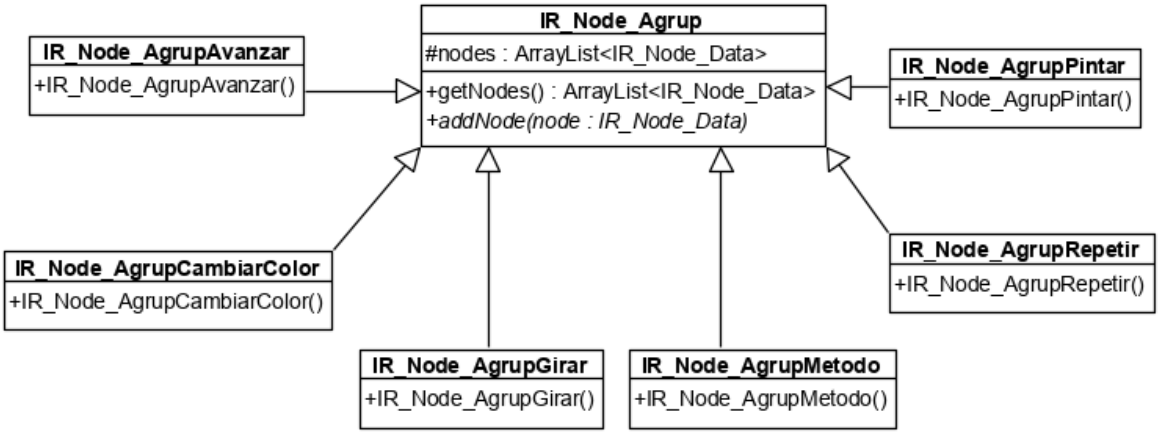

Figura 3. 11. 4 - Diagrama de clases para la agrupación de nodos para la representación intermedia

3.12 Generación de código

Para la última fase se encuentra el generador de código, el cual hace uso de la o las representaciones intermedias generadas para obtener los datos importantes guardados en la tabla de símbolos y utilizarlos para generar una salida.

Un generador de código posee como objetivo decidir qué acciones se van a llevar a cabo dependiendo del código fuente y seleccionar los datos necesarios para generar la salida esperada, así como decidir el orden en que serán efectuadas las acciones. La selección de instrucciones implica la selección de instrucciones adecuadas para llevar a cabo las instrucciones esperadas.[5]

Para este caso se desarrolló un intérprete para determinar que instrucción se tenían que llevar a cabo en java dependiendo de lo que se seleccionó en el área de desarrollo para poder dibujar en el `Canvas` especifico, para esto se desarrollaron las siguientes clases:

Tabla 3. 12. 1 - Tabla de clases para la generación de código

Clases construidas	Objetivo
Interpreter	**Es una clase abstracta en donde se determinaron los métodos necesarios para la implementación de las instrucciones en java, así como asignar algunos atributos estáticos y otros necesarios para el funcionamiento de la parte denominada escenario.**
InterpreterDrawing	**Es la clase que hereda de `interpreter` y les da funcionalidad a los métodos planteados.**

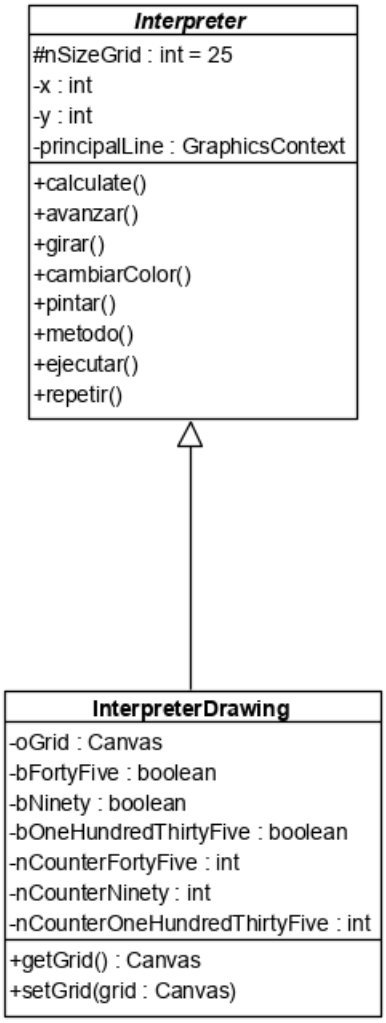

Figura 3. 12. 1 - Diagrama de las clases Interpreter e InterpreterDrawing

3.13 Implementación de la aplicación de lenguaje en el IDE

Después de desarrollar la aplicación de lenguaje se procedió a enlazarla con el prototipo 7 de la herramienta, para obtener una primera versión “completa” del IDE.

3.13.1 Avance Prototipo 8

En este avance se agregaron diálogos de texto para solicitarle al usuario los datos necesarios según la instrucción que desean colocar en el área de desarrollo, como se visualiza en la figura 3.13.1

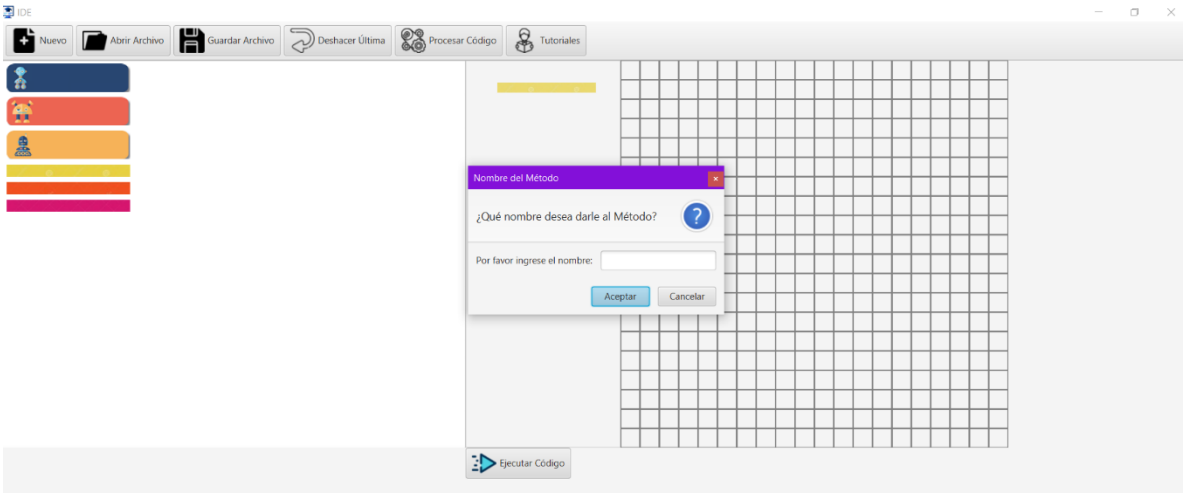

Figura 3. 13. 1. 1 - Diálogo de texto Solicitar Nombre Método

Tabla 3. 13. 1. 1 - Tabla de clases para el prototipo 8

Clases construidas	Objetivo
IDEWorkSpaceController	**Se modificó la clase para incluir el intérprete generado en la sección de generación de código, los `TextInputDialog` para obtener la información necesaria para cada una de las sentencias y el escucha de la representación intermedia.**

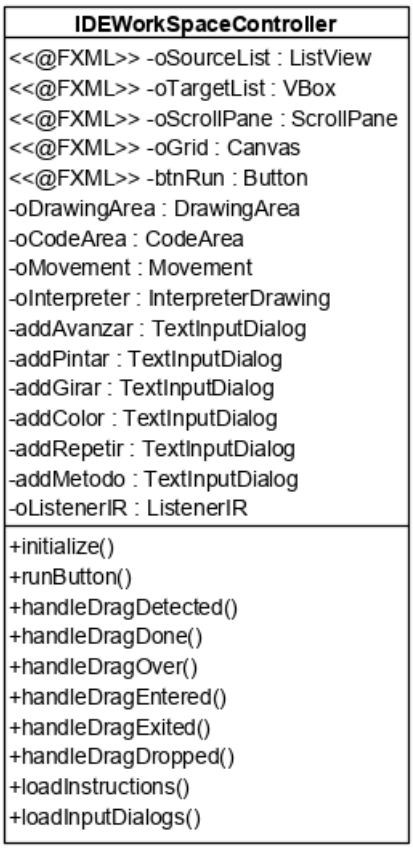

Figura 3. 13. 1. 2 - Diagrama de la clase IDEWorkSpaceController modificado

3.13.2 Avance Prototipo 9

En este avance se enlazó la aplicación de lenguaje mediante un escucha para procesar el método que se encuentra en el área de desarrollo, lo que muestra una ventana de Correcto en caso de que el método no tenga errores o una ventana con la información de los errores detectados, según sea el caso, las cuales se pueden observar en las figuras 3.13.2.1 y 3.13.2.2 respectivamente.

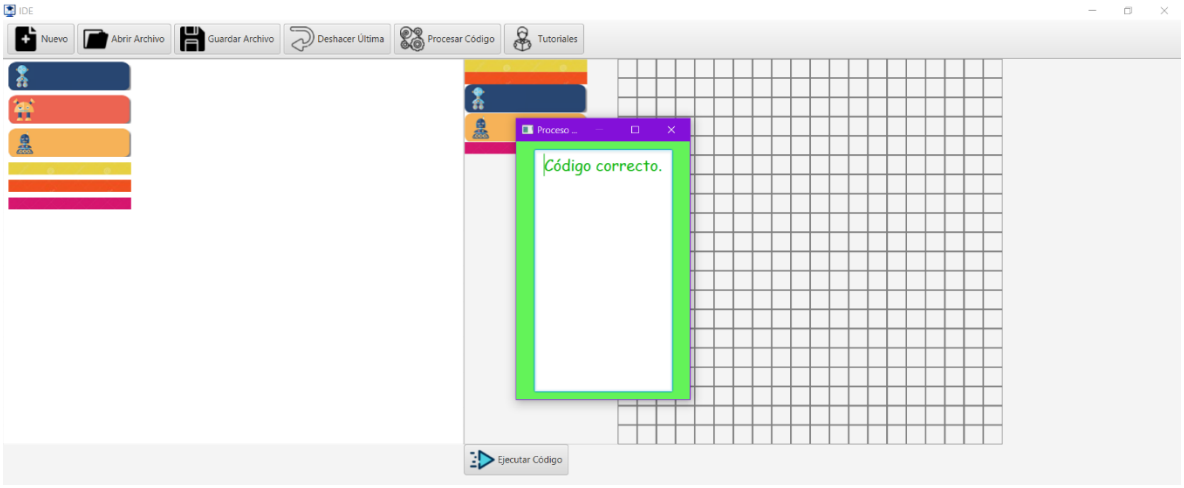

Figura 3. 13. 2. 1 - Ventana de Código Correcto

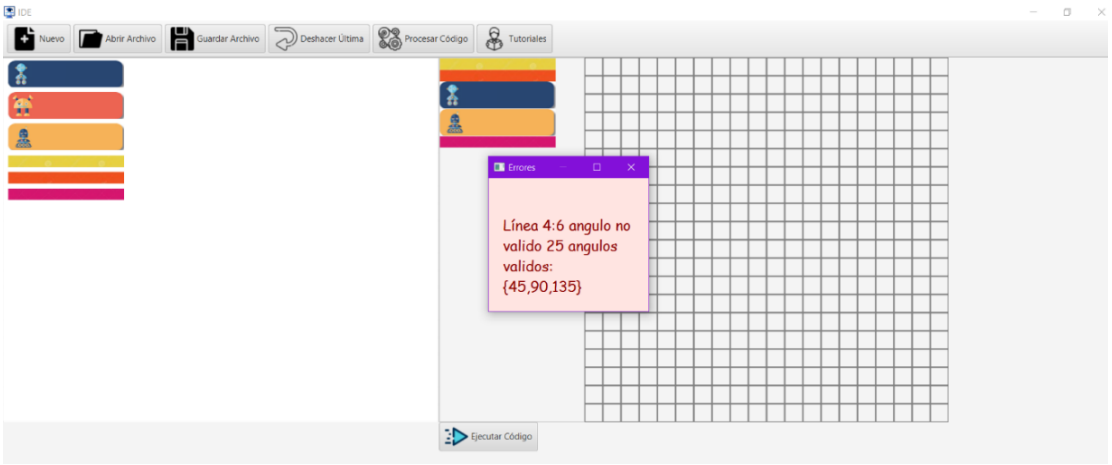

Figura 3. 13. 2. 2 - Ventana de Errores

ANTLR proporciona soporte mediante dos mecanismos para caminar a través de los árboles en su tiempo de ejecución, los escuchas y los visitantes.

Los escuchas son la opción por omisión, en donde presenta métodos asociados al inicio y fin de cada regla, al generar un analizador gramatical, se genera una interfaz que hereda los métodos de inicio y fin de cada regla la cual puede ser heredada para obtener información. Los visitantes deben solicitar su generación al procesar el archivo de la gramática y solo implementan las funciones para visitar o recorrer nodos específicos. [38]

Dado a que se necesita recorrer cada uno de los nodos de las reglas generadas se optó por utilizar un escucha para el recorrido del árbol.

Tabla 3. 13. 2. 1 - Tabla de clases para el prototipo 9

Clases construidas	Objetivo
Listener	**Se desarrolló el Escucha para enlazar la gramática con la aplicación y poder determinar si estaba armada correctamente una instrucción.**

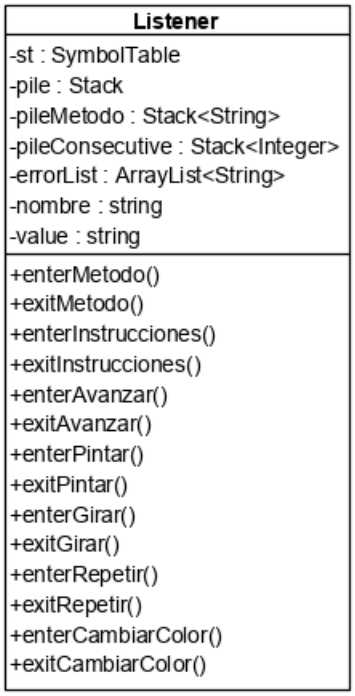

Figura 3. 13. 2. 3 - Diagrama de la clase Listener

3.13.3 Avance Prototipo 10

Para este prototipo se implementó la funcionalidad de cada una de las sentencias a utilizar con ayuda de la gramática desarrollada mediante ANTLR4, de igual manera se agregaron las imágenes generadas para las sentencias, además de modificar los iconos de los botones para que fueran más llamativos para los usuarios finales, lo que se puede apreciar en la Figura 3.13.3.1 y se implementó la primera etapa de los tutoriales, la cual abre una ventana no modal para seleccionar un video y ver la realización de alguna figura sencilla, como se puede observar en la Figura 3.13.3.2.

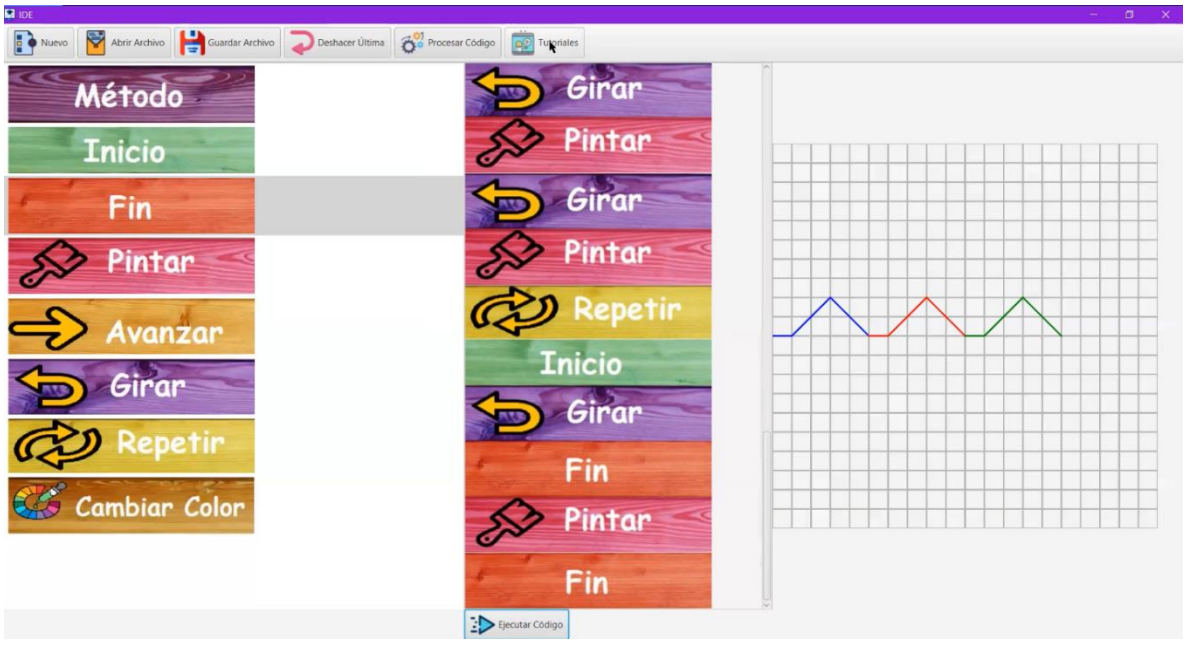

Figura 3. 13. 3. 1 - IDE con funcionalidad para las sentencias y nuevas imágenes.

Tabla 3. 13. 3. 1 - Tabla de clases para el prototipo 10

Clases construidas	Objetivo
TutorialsController	**Se construyó la clase para darle funcionamiento a la parte de los tutoriales que es en una ventana no modal.**

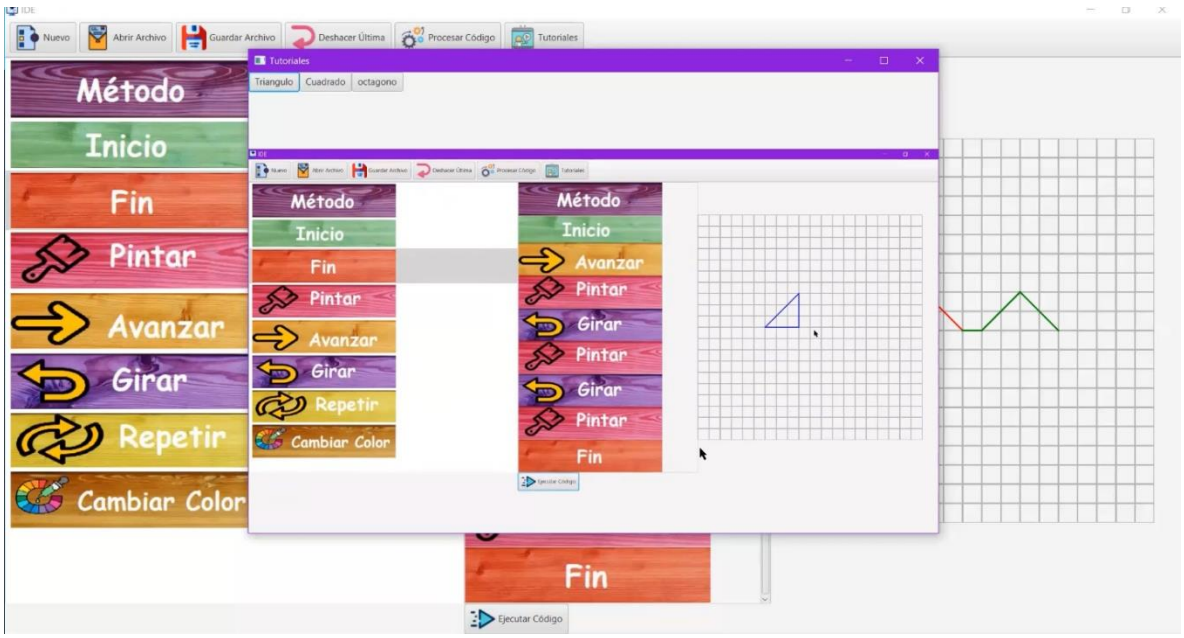

Figura 3. 13. 3. 2 - Primera versión se los tutoriales en el IDE.

TutorialsController
<<@FXML>> -btnFigureOne : Button <<@FXML>> -btnFigureTwo : Button <<@FXML>> -btnGrecaOne : Button <<@FXML>> -btnGrecaTwo : Button <<@FXML>> -oMedia : MediaView <<@FXML>> -oPane : AnchorPane
+initialize() +playFigureOne() +playFigureTwo() +playGrecaOne() +playGrecaTwo()

Figura 3. 13. 3. 3 - Diagrama de la clase TutorialsController

Capítulo 4: Resultados

Como parte del trabajo realizado se creó un *IDE* el cual permite desarrollar patrones geométricos como lo son las grecas, así como diferentes figuras geométricas, entre ellas: triángulo, octágono, cuadrado, entre otros. Esto con el apoyo de un lenguaje de programación en idioma español generado con ANTLR4, el cual ayuda a facilitar el entendimiento de las sentencias utilizadas dentro del *IDE*.

En este capítulo se mide la usabilidad de la herramienta, en base a dos casos de estudio propuestos, en el primero de ellos se mostró un tutorial a dos niñas de cuarto de primaria, explicando cómo podría generarse una greca cuadrada utilizando las sentencias del *IDE*, pero dejando fuera la parte de repetir, y en el segundo se mostró un tutorial, explicando cómo generar una greca en zigzag con ayuda de repetir.

Usabilidad:

La usabilidad se relaciona con la eficiencia, la efectividad y la experiencia subjetiva asociada con los usuarios que interactúan con una aplicación para lograr objetivos. Básicamente la usabilidad de un producto de software está relacionada con la facilidad que tiene un usuario para lograr realizar la tarea para la cual el producto fue diseñado. La usabilidad no puede ser medida de primera mano, ya que es un valor subjetivo.

La ISO 9241-11 nos brinda el siguiente concepto de usabilidad: “usabilidad se describe como el grado con el que un producto puede ser usado por usuarios específicos para alcanzar objetivos específicos con efectividad, eficiencia y satisfacción, en un contexto de uso especifico” [39].

Efectividad: La efectividad es la capacidad de tener éxito y producir los resultados esperados, Para medir la efectividad se utiliza la cantidad de errores obtenidos por el usuario al hacer uso de la herramienta.

Eficiencia: Se conoce a la eficiencia como el buen uso del tiempo utilizado para lograr un objetivo esperado. Para lograr medir la eficiencia de un producto de software se utiliza el

tiempo empleado en finalizar una misión y el tiempo utilizado para aprender a utilizar el producto. Mientras menos tiempo se desperdicie mejor nivel de eficiencia.

Satisfacción: La satisfacción es un sentimiento de agrado cuando se realiza algo con una meta especifica en la forma que se desea. Este elemento es medido con escalas para calificar el agrado que se tuvo con la interacción con la herramienta.

Si se busca medir la usabilidad se requiere desbaratar a las propiedades que la conforman en elementos mensurables y comprobables. En la figura 4.1 del *framework* de usabilidad utilizado en la norma ISO 9241-11 [39], se pueden observar la relación entre el producto de software, el contexto de uso, las medidas de usabilidad y la meta que se desea lograr.

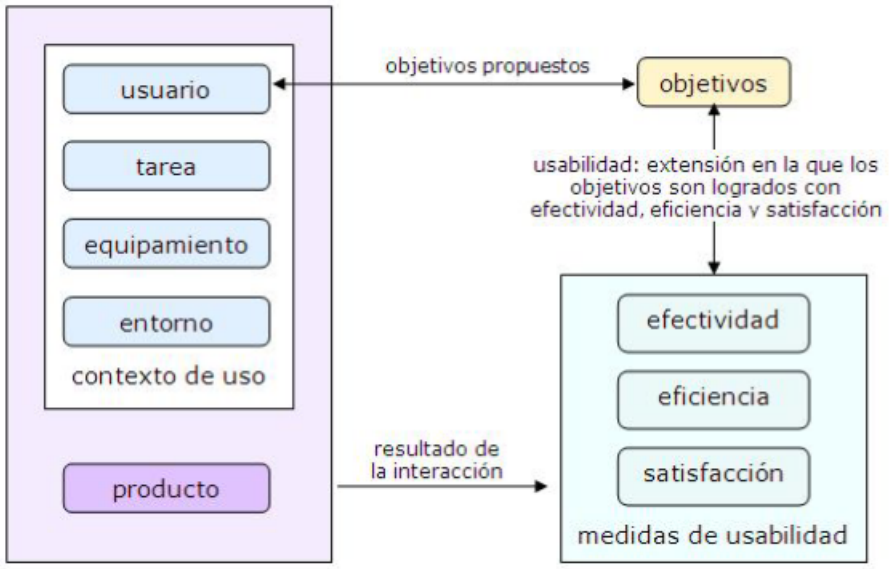

Figura 4. 1- Framework de usabilidad (ISO 9241-11).

A continuación, se destacan algunos elementos empleados para mensurar el nivel de usabilidad de un producto de software:

Facilidad de aprendizaje: La primera interacción que tiene un usuario con un producto de software es la de entender su funcionamiento, se refiere a la sencillez con la que un usuario resuelve una tarea la primera vez que lo intenta.

Memoria mental: Se refiere a que tan fácil es recordar la manera en que se utiliza el producto de software para lograr la meta esperada. Un usuario que ya utilizo la herramienta debe tener una curva de aprendizaje menor a la de un usuario que interacciona con ella por primera vez.

Errores: Se refiere a la cantidad de veces que un usuario se equivoca al realizar una tarea en la herramienta. El producto de software no debería producir errores por si sola, en caso de existir errores se le debe notificar al usuario de manera rápida y concisa, también se le debe brindar la forma de sobreponerse del error.

Métricas de Usabilidad

Dado a que los elementos utilizados para medir el nivel de usabilidad de un producto de software son abstractos, para lograr su medición se deberá asignarles ciertas métricas, como puede ser, para medir el elemento de eficiencia se usa la cantidad de tiempo utilizado por un usuario para finalizar una tarea determinada.

"Una métrica (medida) es un valor numérico o nominal asignado a características o atributos de un objeto computado a partir de un conjunto de datos observables y consistentes con la intuición" [40]. Una métrica cuenta con las siguientes cualidades:

- Contiene elementos matemáticos.
- Si un resultado es positivo el valor de la métrica debe subir, pero si el valor es negativo el valor debe bajar respectivamente.
- Una métrica debe verificarse matemáticamente con un alto número de contextos antes de aplicarse en las mediciones.

De acuerdo con [41] las métricas son clasificadas en estáticas y dinámicas. "Las métricas estáticas se utilizan para medir las características, que no cambian, de una aplicación, como el tamaño del código o la complejidad de este. Las métricas dinámicas permiten medir el comportamiento de la aplicación, se calculan con la aplicación en ejecución". Se debe mencionar que las métricas de usabilidad no brindan un resultado especifico, sino que nos proporciona datos acerca del uso del producto de software por parte del usuario. Dichos datos pueden ser evaluados para tomar acción dentro del producto y mejorarlo, para esto se obtuvo la tabla 4.1 con las métricas para la medición de la usabilidad propuestas.

Tabla 4. 1 - Métricas para la usabilidad

Atributos	Métricas
Efectividad	• Tareas resueltas en un tiempo limitado. • Porcentaje de tareas completadas con éxito al primer intento. • Número de funciones aprendidas.
Eficiencia	• Tiempo empleado en completar una tarea. • Tiempo transcurrido en cada pantalla. • Eficiencia relativa en comparación con un usuario experto.
Satisfacción	• Nivel de dificultad. • Agrada o no agrada. • Preferencias.
Facilidad de aprendizaje	• Tiempo usado para terminar una tarea la primera vez. • Cantidad de entrenamiento. • Curva de aprendizaje.
Memoria mental	• Número de pasos, clics o páginas usadas para terminar una tarea después de no usar la aplicación por un periodo de tiempo.
Errores	• Número de errores.

4.1 Caso de estudio 1

En este caso se les explico a los niños durante 15 min el funcionamiento de la herramienta, después se procedió a mostrarles un tutorial para hacer una greca cuadrada y un triángulo, los niños se notaron atraídos por la herramienta gracias a las sentencias de colores, uno de los puntos que causó confusión en ellos fue el uso de los giros.

Para solventar el problema identificado se les mostró una hoja explicando los tres tipos de giros los cuales eran de 45°, 90° y 135 °, con esta hoja presente les fue más fácil identificar la cantidad de giros necesarios para ir hacia donde ellos querían. Una vez entendido el uso de los giros hacer la greca fue mucho más sencillo. Se puede observar el resultado en las figuras 4.1.1 y 4.1.2.

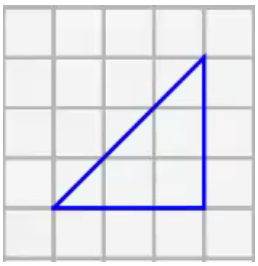

Figura 4. 1. 2 - Triángulo Generado

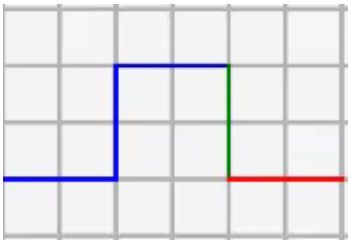

Figura 4. 1. 1 - Primera Greca Generada

La fórmula de Euler nos dice que todo grafo simple se cumple que r = 2 – v + e, entonces v + r = e + 2, el número de nodos + el número de regiones = el número de aristas + 2.

Aplicación de la formula

Triangulo:

r = 2 – v + e -> r = 2 – 9 + 9 -> r = 2

v + r = e +2

9 + 2 = 9 + 2 -> 11=11

Por lo tanto, es un grafo valido.

Greca cuadrada:

r = 2 – v + e -> r = 2 – 11 + 10 -> r= 1

v + r = e + 2

11 + 1 = 10 + 2 -> 12 = 12

Por lo tanto, es un grafo valido.

4.2 Caso de estudio 2

En esta ocasión se procedió a mostrarles a los niños un tutorial para hacer una greca en zigzag y un octágono haciendo uso de la instrucción REPETIR, se pensó que los niños tendrían problemas para utilizar dicha instrucción pero no fue así; una vez entendido el uso

de los giros y las demás sentencias el uso de REPETIR fue relativamente fácil; siguiendo la lógica del caso 1 les resultó más rápido el generar la segunda figura geométrica, la cual se puede observar en la figura 4.2.1, así como la segunda greca, ahora en zigzag, la cual se puede observar en la figura 4.2.2; fue interesante ver que, a medida que avanzaban, los niños iban diciendo las sentencias que se necesitaban utilizar, lo cual brinda información de que el uso de la herramienta ayuda a los jóvenes a entender conceptos tanto matemáticos como de programación al mismo tiempo en el que sienten que están jugando como se muestra en la figura 4.2.3.

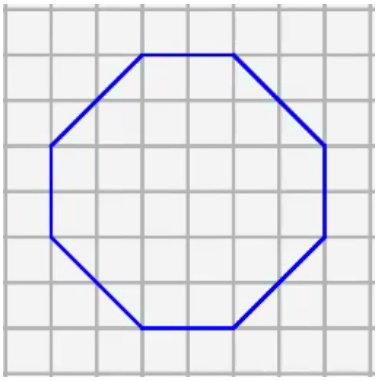

Figura 4. 2. 1 - Octágono Generado

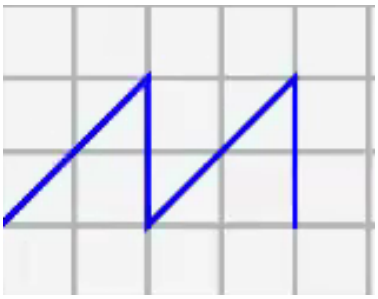

Figura 4. 2. 2 - Segunda Greca generada

La fórmula de Euler nos dice que todo grafo simple se cumple que r = 2 – v + e, entonces v + r = e + 2, el número de nodos + el número de regiones = el número de aristas más 2.

Aplicación de la formula

Octágono:

r = 2 – v + e -> r = 2 – 16 + 16 -> r = 2

v + r = e +2

16 + 2 = 16 + 2 -> 18 = 18

Por lo tanto, es un grafo valido.

Greca Zigzag:

r = 2 – v + e -> r = 2 – 9 + 8 -> r= 1

v + r = e + 2

9 + 1 = 8 + 2 -> 10 = 10

Por lo tanto, es un grafo valido.

Figura 4. 2. 3 - Experiencia de usuario

4.3 Evaluación de Resultados

A continuación, se presentan las tablas de resultados obtenidos de acuerdo con los dos casos de estudio realizados con los infantes.

En la tabla 4.3.1 se presentan las métricas obtenidas con el primer sujeto de estudio y la tabla 4.3.2 las correspondientes al segundo sujeto de estudio.

Tabla 4. 3. 1 - Métricas primer usuario

Métricas	Resultados
Tareas resueltas en un tiempo limitado. (20 minutos)	1/2 (Caso 1) 2/2 (Caso 2)
Porcentaje de tareas completadas con éxito al primer intento.	1/2 (Caso 1) 2/2 (Caso 2)
Número de funciones aprendidas.	5/6
Tiempo empleado en completar una tarea.	(20 min, 25 min) (15 min, 18 min)
Tiempo transcurrido en cada pantalla.	Tutoriales - 40 min, Principal - 38 min
Eficiencia relativa en comparación con un usuario experto.	Usuario Experto (7 min) – Usuario Nuevo (19.5 min) (Media)

Nivel de dificultad.	Media
Agrada o no agrada.	Agrada
Preferencias.	Figuras geométricas
Tiempo usado para terminar una tarea la primera vez.	20 min
Cantidad de entrenamiento.	15 min de explicación sobre la herramienta
Curva de aprendizaje	Se disminuye 1 min en cada ejercicio de acuerdo con la fórmula: $Y_x = Kx^{\log_2 b}$
Tiempo usado para terminar una tarea después de no usar la aplicación por un periodo de tiempo. (2 semanas)	28 min
Número de errores.	1

Tabla 4. 3. 2 - Métricas segundo usuario

Métricas	**Resultados**
Tareas resueltas en un tiempo limitado. (20 minutos)	1/2 (Caso 1) 2/2 (Caso 2)
Porcentaje de tareas completadas con éxito al primer intento.	1/2 (Caso 1) 2/2 (Caso 2)
Número de funciones aprendidas.	6/6
Tiempo empleado en completar una tarea.	(18 min, 23 min) (13 min, 15 min)
Tiempo transcurrido en cada pantalla.	Tutoriales - 34 min, Principal - 35 min
Eficiencia relativa en comparación con un usuario experto.	Usuario Experto (7 min) – Usuario Nuevo (17.5 min) (media)
Nivel de dificultad.	Media
Agrada o no agrada.	Agrada
Preferencias.	Figuras geométricas
Tiempo usado para terminar una tarea la primera vez.	18 min
Cantidad de entrenamiento.	15 min de explicación sobre la herramienta
Curva de aprendizaje	Se disminuye 1 min en cada ejercicio de acuerdo con la fórmula: $Y_x = Kx^{\log_2 b}$
Tiempo usado para terminar una tarea después de no usar la aplicación por un periodo de tiempo. (2 semanas)	28 min
Número de errores.	0

Al evaluar los casos de estudios realizados, se identificó que la falta de información sobre el funcionamiento de la herramienta dentro de la misma hace que los infantes se tomen más tiempo al realizar la actividad ya que no recuerdan todo lo explicado y regresan a ver de nuevo el tutorial para guiarse, además que algunos conceptos como el de los giros les resulta complejo sin alguna explicación visual de su funcionamiento, por tal razón se decidió agregar un botón de ayuda en donde se le brinde la información necesaria sobre las instrucciones y su funcionamiento, como se puede visualizar en la Figura 4.3.1.

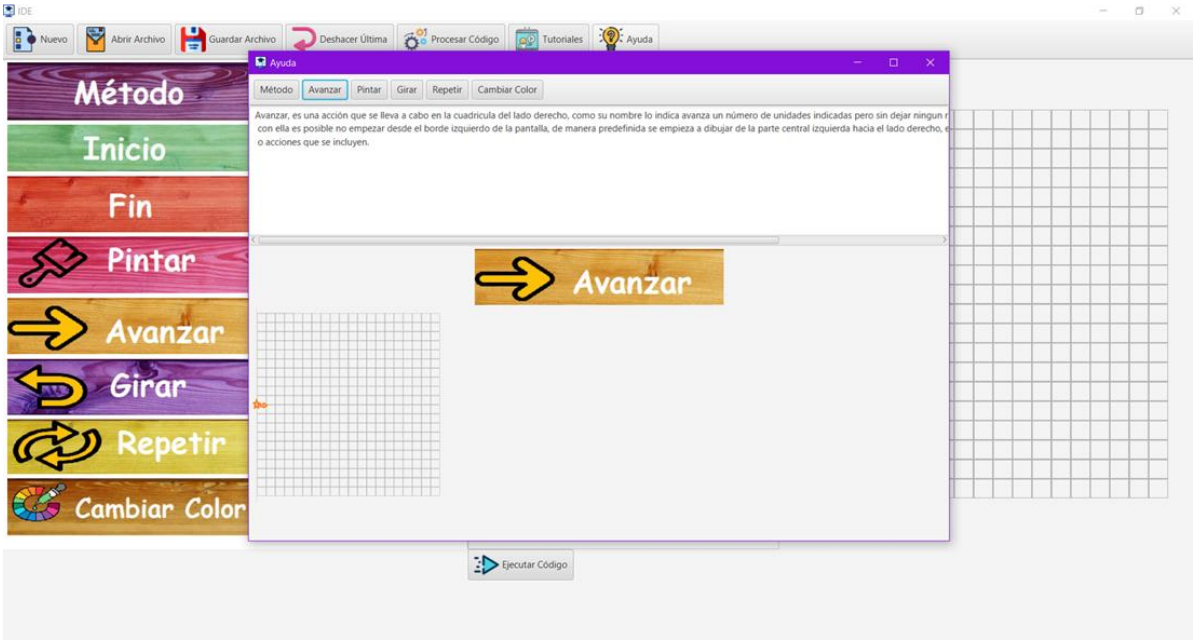

Figura 4. 3. 1 - Sección de Ayuda agregado al IDE

Capítulo 5: Conclusiones y Recomendaciones

Después de realizar el estudio para el desarrollo de la tesis, se obtuvieron algunos puntos importantes, los cuales indicaron que la propuesta era aceptable, como se muestra a continuación: El nivel de inglés en los jóvenes mexicanos es muy bajo lo que le brinda dificultad a aprender conceptos de programación dado a que la mayoría de herramientas o datos acerca de la programación están en inglés, por esta razón la generación de un lenguaje de programación en idioma español es favorable para ellos; Brindarle tanto a niños como niñas conocimientos en programación genera una equidad de oportunidades en el área de trabajo; Según el análisis de estudios relacionados con el tema, el brindarle a los niños, niñas y jóvenes algo con lo que puedan interactuar y además poder visualizar el resultado hace que se interesen más en el tema; Dado a que se busca apoyar a la STEM, abarcar conocimientos de matemáticas, programación y utilizar tecnología para enseñarla hace factible la realización del presente trabajo.

Con el presente trabajo se intenta aportar al desarrollo de habilidades matemáticas en los usuarios, entre las habilidades matemáticas deseadas se encuentran las siguientes: graficar, algorítmica, calcular, modelar, entre otros; los usuarios de igual manera se verán familiarizados con los siguientes conceptos en la rama de las matemáticas: Grafo, arista, nodo, grados, entre otros.

Si se desea desarrollar un producto de software se deben conocer los requisitos con los que debe cumplir el mismo, para lograr esto, se utilizó el análisis comparativo de elementos semejantes, descrita por Tom DeMarco, evaluando ambientes de desarrollo dedicados a la enseñanza de la programación como lo son: Scratch y Snap!, ambientes dedicados al desarrollo de bordados como TurtleStitch y de igual manera ambientes de desarrollo para la producción de software; identificando así las mejores prácticas de cada uno para basarse en ellos pero aplicando algunas diferencias necesarias, por ejemplo, los ambientes de desarrollo dedicados a la enseñanza y al bordado están desarrollados para su uso en forma web, mientras que el presente trabajo se decidió ser desarrollado en un ambiente de escritorio, dado a las carencias de internet que pueden tener algunas escuelas, cinco de las

seis herramientas analizadas utilizan el idioma inglés para los conceptos utilizados en la codificación, mientras que el presente trabajo utiliza un lenguaje propio en español para un mejor entendimiento de los usuarios a los que está dirigido, además de que la herramienta propuesta está enfocada a no solo enseñar programación sino también conceptos matemáticos simples.

Una vez identificados los requisitos con los que debe cumplir el producto de software, se debió modelar en una arquitectura que sirviera de guía en el desarrollo de dicho producto, por lo que fue imprescindible diseñar diferentes diagramas UML los cuales consideraron los requisitos y las tecnologías a utilizar en la elaboración del producto final.

La utilización de la tecnología de ANTLR4 para la generación de la aplicación de lenguaje en idioma español, el cual hace que los usuarios se sientan más confiados a la utilización del ambiente de desarrollo gracias a que los conceptos utilizados se encuentran en su idioma nativo, se volvió una tarea sencilla dado a que nos proporciona clases en Java las cuales pueden ser utilizadas para recobrar datos mientras la herramienta se encuentre en ejecución.

Gracias a utilizar las características de la Ingeniería de Software se desarrolló de forma satisfactoria el *IDE* planteado, recabando información para obtener los requisitos esenciales para que los niños y jóvenes se adentren en el mundo de la programación, achicando las aberturas del lenguaje que se tienen con ambientes de desarrollo semejantes.

La utilización de imágenes en lugar de escribir las instrucciones se decidió gracias al artículo [16], en donde se indica que la utilización de imágenes es más llamativa para los niños, dado esto, las instrucciones se transformaron en imágenes que son arrastradas para estructurarlas en forma de bloques, idea que fue tomada de las herramientas analizadas mencionadas anteriormente.

Es importante destacar que se utilizaron un cierto número de colores ya que esta fue una primera fase de la herramienta, a la cual se le pueden seguir agregándole más colores para que los usuarios tengan más opciones en la utilización de esta, de igual manera en los conceptos de programación se pueden seguir agregando diferentes tipos de instrucciones como lo podrían ser: While, Switch, Do-While, entre otros.

En lo que toca a recomendaciones, es deseable incluir en la herramienta la posibilidad de convertir las imágenes resultantes en bordados dado que los patrones obtenidos son relativamente simples y factibles de obtener con una máquina de coser, que tiene menor costo que una máquina de bordar. También es importante probar la herramienta con un mayor número de niños y, de acuerdo con los resultados que se obtengan, analizar la posibilidad de extender el lenguaje para incluir otras instrucciones y/o el uso de sentencias definidas para el usuario. Otro posible trabajo a futuro es la incorporación del concepto de "Bibliotecas", con la finalidad de reutilizar elementos ya definidos previamente para crear salidas más complejas.

Referencias

[1] Ankita Vashistha, Richard Jones, y Anthony Rajesh, “TGIIndex”, *THOLONS*. http://tholons.com/tsgindex/

[2] Ciencias Holguín, Revista trimestral,Año XX,abril-junio2014Ciencias HolguínISSN 1027-21271El lenguaje de programaciónPython/Theprogramming languagePythonIvet Challenger-Pérez, Yanet Díaz-Ricardo, y Roberto Antonio Becerra-García, “El lenguaje de programaciónPython/Theprogramming languagePython”. Ciencias Holguín, jun. 2014.

[3] W. M. Waite y G. Goos, *Compiler Construction*. New York, NY: Springer New York, 1984. doi: 10.1007/978-1-4612-5192-7.

[4] J. A. U. Urzúa, J. A. P. Urtubia, J. F. Á. Rubio, y M. C. R. Zúñiga, “DESARROLLO DE UN LENGUAJE PARA LA VISUALIZACIÓN DE ESTRUCTURAS DE DATOS”, p. 159.

[5] Alfred V. Aho, “Compiladores principios, técnicas y herramientas”, núm. segunda edición, p. 1040, 2008.

[6] “TOOL | significado, definición en el Cambridge English Dictionary”. https://dictionary.cambridge.org/es-LA/dictionary/english/tool (consultado ene. 26, 2020).

[7] J. A. Pulgarín, “Generalización de patrones geométricos. Proyecto de aula para desarrollar pensamiento variacional en estudiantes de 9 – 12 años.”, p. 91.

[8] Y. Sawano, “(54) ASYNCHRONOUS CODE TESTING IN INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)”, p. 17.

[9] DIANA MARITZA ROJAS, “TEORIA DE GRAFOS Y SUS APLICACIONES CON EL TANGRAM”. dic. 13, 2003.

[10] “GRAFO | Definición de GRAFO por Oxford Dictionary en Lexico.com y también el significado de GRAFO”, *Lexico Dictionaries | Español*. https://www.lexico.com/es/definicion/grafo (consultado sep. 02, 2021).

[11] P. López, “Urge aumentar la investigación en ciencias de la computación.”, Gaceta UNAM - en línea, nº 5088, 2018.

[12] “Movimiento STEM México”. https://www.movimientostem.org/ (consultado may 13, 2020).

[13] “STEM (Science, Technology, Engineering and Mathematics) | British Council México”. https://www.britishcouncil.org.mx/stem-science-technology-engineering-and-mathematics (consultado may 13, 2020).

[14] “Stem México”. //stem-mexico.com/index.html

[15] "Cuantrix". https://cuantrix.mx (consultado may 13, 2020).

[16] K. Zhu, X. Ma, G. K. W. Wong, y J. M. H. Huen, "How Different Input and Output Modalities Support Coding as a Problem-Solving Process for Children", en *Proceedings of the The 15th International Conference on Interaction Design and Children - IDC '16*, Manchester, United Kingdom, 2016, pp. 238–245. doi: 10.1145/2930674.2930697.

[17] M. Mariappan, J. C. Sing, y M. Nadarajan, "Design and Development of Tangible Instruction Set for Educational Robotic System", en *Proceedings of the 4th International Conference on Control, Mechatronics and Automation - ICCMA '16*, Barcelona, Spain, 2016, pp. 12–15. doi: 10.1145/3029610.3029616.

[18] K. DePryck, "From computational thinking to coding and back", en *Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality - TEEM '16*, Salamanca, Spain, 2016, pp. 27–29. doi: 10.1145/3012430.3012492.

[19] G. Tisza *et al.*, "The role of age and gender on implementing informal and non-formal science learning activities for children", en *Proceedings of the FabLearn Europe 2019 conference on ZZZ - FabLearn Europe '19*, Oulu, Finland, 2019, pp. 1–9. doi: 10.1145/3335055.3335065.

[20] A. Karmon, Y. Sterman, T. Shaked, E. Sheffer, y S. Nir, "KNITIT: a computational tool for design, simulation, and fabrication of multiple structured knits", en *Proceedings of the 2nd ACM Symposium on Computational Fabrication - SCF '18*, Cambridge, Massachusetts, 2018, pp. 1–10. doi: 10.1145/3213512.3213516.

[21] V. Narayanan, L. Albaugh, J. Hodgins, S. Coros, y J. Mccann, "Automatic Machine Knitting of 3D Meshes", *ACM Trans. Graph.*, vol. 37, núm. 3, pp. 1–15, ago. 2018, doi: 10.1145/3186265.

[22] J. McCann *et al.*, "A compiler for 3D machine knitting", *ACM Trans. Graph.*, vol. 35, núm. 4, pp. 1–11, jul. 2016, doi: 10.1145/2897824.2925940.

[23] V. Narayanan, K. Wu, C. Yuksel, y J. McCann, "Visual knitting machine programming", *ACM Trans. Graph.*, vol. 38, núm. 4, pp. 1–13, jul. 2019, doi: 10.1145/3306346.3322995.

[24] M. Hofmann *et al.*, "KnitPicking Textures: Programming and Modifying Complex Knitted Textures for Machine and Hand Knitting", en *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, New Orleans LA USA, oct. 2019, pp. 5–16. doi: 10.1145/3332165.3347886.

[25] S. Yang, "Knitting Visualizer: Connecting Craft and Code", en *Proceedings of the 2017 Conference on Interaction Design and Children - IDC '17*, Stanford, California, USA, 2017, pp. 705–708. doi: 10.1145/3078072.3091985.

[26] T. DeMarco, *The Deadline: A Novel about Project Management*. Dorset House Pub., 1997.

[27] C. Guindon, “Eclipse desktop & web IDEs | The Eclipse Foundation”. https://www.eclipse.org/ide/

[28] “Visual Studio 2019”, *Visual Studio*. https://visualstudio.microsoft.com/es/vs/

[29] “Scratch - About”. https://scratch.mit.edu/ (consultado ene. 31, 2020).

[30] “Turtlestitch - Coded Embroidery / TurtleStitch - Coded Embroidery”, *Turtlestitch - Coded Embroidery / TurtleStitch - Coded Embroidery*. https://www.turtlestitch.org/page/about (consultado ene. 31, 2020).

[31] “Ask John: Which IDE should I use with my students? – WeTeach_CS Blog Archive”. https://sites.utexas.edu/weteachcs/ask-john-which-ide-should-i-use-with-my-students/ (consultado jun. 24, 2020).

[32] Iván Picie-Alcaraz, Beatriz Alejandra Olivares-Zepahua, Ignacio López-Martínez, Celia Romero-Torres, y Luis Ángel Reyes-Hernández, “Design of an IDE for Teaching Programming using Graphic Elements”, *2020 9th Int. Conf. Softw. Process Improv. CIMPS*, mar. 2021, doi: 10.1109/CIMPS52057.2020.9390149.

[33] Maximiliano Cristiá, “Introducción a la Arquitectura de Software”, Technical Report, ene. 2008.

[34] Robert C. Martin, *Clean Architecture A CRAFTSMAN’S GUIDE TO SOFTWARE STRUCTURE AND DESIGN*. 2018.

[35] R. S. Pressman, “Ingenieria del Software. Un Enfoque Practico”, p. 810.

[36] M. Fowler y K. Scott, *UML gota a gota*. México: Pearson Educación, 1999.

[37] Iván Picie-Alcaraz, Beatriz Alejandra Olivares-Zepahua, Ignacio López-Martínez, Celia Romero-Torres, y Luis Ángel Reyes-Hernández, “Herramienta para la Enseñanza de la Programación usando Elementos Gráficos”, *RISTI*, mar. 2021, doi: 10.17013/risti.41.50–62.

[38] T. Parr, *The definitive ANTLR 4 reference*. Dallas, Texas: The Pragmatic Bookshelf, 2012.

[39] “ISO 9241-11:2018(en), Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts”. https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en (consultado may 07, 2021).

[40] T. DeMarco, *Controlling Software Projects: Management, Measurement & Estimation*. Yourdon Press, 1982.

[41] J. G. Enriquez y S. I. Casas, “Usabilidad en aplicaciones móviles”, *Inf. Científicos Téc. - UNPA*, vol. 5, núm. 2, pp. 25–47, jun. 2014, doi: 10.22305/ict-unpa.v5i2.71.