risti
Revista Ibérica de Sistemas e Tecnologias de Informação
Revista Ibérica de Sistemas y Tecnologías de Información

Recebido/Submission: 22/12/2020
Aceitação/Acceptance: 14/02/2021

Herramienta para la Enseñanza de la Programación usando Elementos Gráficos

Iván Picie-Alcaraz[1], Beatriz Alejandra Olivares-Zepahua[1], Ignacio López-Martínez[1], Celia Romero-Torres[1], Luis Ángel Reyes-Hernández[1]

{msc.jpicie, bolivares, ilopez, cromero, lreyes}@ito-depi.edu.mx

[1] Tecnológico Nacional de México, Campus Orizaba, División de Estudios de Posgrado e Investigación. Orizaba, Veracruz, México.

DOI: 10.17013/risti.41.50–62

Resumen: El aprendizaje de la programación es difícil y requiere un arduo trabajo por parte de los estudiantes. Necesitan hacer muchos ejercicios y crear muchos programas para mejorar sus competencias en programación, y más aún cuando las herramientas con las que se cuentan no están en su idioma nativo. En este artículo se propone una herramienta, también llamada ambiente de desarrollo en conjunto de un lenguaje de programación, en idioma español, que permita programar la generación de patrones geométricos, como un apoyo a la enseñanza de STEM en México.

Palabras-clave: ambiente de desarrollo; lenguaje de programación; patrones geométricos.

Tool for Teaching Programming Using Graphic Elements

Abstract: Learning programming is difficult and requires hard work on the part of students. They need to do a lot of exercises and create a lot of programs to improve their programming skills, and even more so when the tools they have are not in their native language. This article proposes a tool, also called a development environment matching with a programming language, in Spanish, that allows programming the generation of geometric patterns, as a support to the teaching of STEM in Mexico.

Keywords: development environment; programming language; geometric patterns.

1. Introducción

En años recientes, alrededor del mundo, se ha hecho hincapié en la enseñanza de STEM (Science, Technology, Engineering and Mathematics - ciencia, tecnología, ingeniería y matemáticas) con los objetivos de innovar en el conocimiento, promover el desarrollo de países y personas y disminuir la brecha de género y/o pobreza entre otros. Una de las aproximaciones a STEM es el manejo de la programación, bien sea mediante elementos

físicos como robots, o elementos intangibles como la solución de problemas; los lenguajes para programación en ese caso retoman lenguajes como Logo (Logo History, s/f) o se generan nuevos como el sucesor de Logo: Scratch (Scratch - About, s/f). En México se cuenta con varias iniciativas asociadas a STEM (Movimiento STEM, s/f), (STEM (Science, Technology, Engineering and Mathematics) | British Council México, s/f) y, específicamente en las Ciencias Computacionales, a través de la enseñanza de la programación (Cuantrix, s/f).

Movimiento STEM México asegura en (Movimiento STEM, s/f) que la enseñanza de STEM debe a) basarse en un problema social real, b) utilizar tecnologías y arte para crear una solución, c) diseñarse, construirse, probarse y mejorarse. Otra iniciativa mexicana relacionada con la enseñanza de la computación es CuantriX (Cuantrix, s/f) que busca generar equidad de oportunidades a través de las Ciencias de la Computación para que cada año en México un millón de niñas, niños y jóvenes aprendan a programar; loable como es su causa, su solución presenta el problema del idioma porque su plataforma contiene pocas herramientas en español que ayuden a programar, ya que aproximadamente el 80% de sus aplicaciones se encuentra en inglés y, de acuerdo a un informe del 2015 de "Mexicanos Primero" citado en (Gómez et al., 2017), sólo el 3% de los egresados de secundaria cuenta con un nivel aceptable del idioma inglés.

En México existe una larga tradición artesanal, la cual se basa en motivos geométricos como las grecas, dibujos relativamente simples que siguen un patrón repetitivo y su estudio incluso forma parte del temario de Matemáticas de tercero de primaria (Ma. Teresa Rojano Ceballos, 2005), (2005). Dada su simplicidad, las grecas se pueden describir utilizando conceptos básicos de matemáticas (puntos, líneas, ángulos, unidades de medida) y también conceptos simples de programación (inicio, fin, avance, repetición). Por todo lo anterior, en este trabajo se aborda el diseño de un IDE que permita a niñas, niños y adolescentes aprender conceptos sobre programación mientras crean patrones geométricos.

2. Trabajos relacionados

Tisza et al. (Tisza et al., 2019) evaluaron tres formas de aprendizaje aplicadas a niños, relacionadas a STEM en Europa, las cuales son: a) aprendizaje formal que sigue un programa de estudios estructurado y es obligatorio; b) aprendizaje informal, fuera de un ambiente formal de aprendizaje, que sigue un programa de estudios estructurado, usualmente voluntario y c) aprendizaje no formal, 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 estudiantes en Europa utilizando las diferentes formas de aprendizaje, este estudio determinó que hay una diferencia entre las actividades que las chicas frecuentemente realizan y aquellas actividades en las que los chicos participan. Las chicas participan más frecuentemente en varias actividades destinadas a involucrar a los participantes en temas científicos como arte, biología, química y física. Los chicos participan en actividades que mejoran 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.

No es 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 (Zhu et al., 2016) se presentaron cuatro talleres de codificación: 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 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.

Es común que los IDEs comerciales se diseñen pensando más en el rendimiento que en los principios de usabilidad. Karvelas (Ioannis Karvelas, 2019) demostró cómo las diferentes formas de indicar los errores afectan a los novatos al aprender programación; se tomaron en cuenta dos formas de presentar los mensaje de error: 1) mostrar sólo el primer error encontrado durante la compilación, excluyendo a los demás si los hubiera, esta forma hizo que los estudiantes se desmotivaran ya que tenían que seguir presionando el botón de compilar para encontrar todos los errores; 2) mostrar los mensajes de error de todos los errores a medida que se escribe, de esta manera los estudiantes tienen la oportunidad de revisar todo el código antes de presionar compilar, lo que tuvo un efecto positivo en los estudiantes. De esta forma, obtuvieron como resultado que el brindar mensajes de error de una manera no amigable para los usuarios hace que estos tengan dificultad al momento de aprender a programar, por lo que es de suma importancia tener en cuenta la experiencia de usuario.

Las dificultades en la comprensión de la programación por los programadores ha sido estudiada por mucho tiempo y se han sugerido diferentes modelos cognitivos para su comprensión; Zayour (Zayour & Hajjdiab, 2013) llevó a cabo el estudio de las prácticas de los programadores en una sola empresa para determinar el efecto que los IDEs tienen en diferentes proyectos en un mismo contexto, se determinó que los mensajes de error la mayoría de las veces representan lo que ha ido mal dentro de la herramienta, son muy generales y algunas veces parecen no tener sentido, lo cual aleja al programador de la causa real y prolonga el tiempo de corrección de errores teniendo que probar soluciones que se vengan a la mente o buscar en Internet soluciones a problemas similares para eventualmente solventar el error, también se obtuvo que las herramientas para depurar los errores a tiempo de ejecución son cada vez menos útiles ya que muestran errores en partes internas del propio IDE, impidiendo al programador revisar lo que él desarrolló.

En (Department of Computer Science, Interlink Polytechnic, Ijebu Jesa, Osun State, 233114, Nigeria et al., 2016) se investigó el efecto de los IDEs en los programadores

principiantes, en la investigación se tomaron muestras de estudiantes y profesores de disciplinas relacionadas con la programación; al analizar los datos se obtuvo que un IDE cuyo propósito es ayudar a los programadores a desarrollar aplicaciones de software efectivas y rápidas, podría causar lo contrario si no se elige correctamente; como resultado se recomendó que un programador principiante utilice un IDE con pocas características técnicas, para que no se sature con características que desconoce y se centre en aprender lo necesario para su nivel, así posteriormente podrá abordar IDEs con características más avanzadas e ir aprendiendo escalonadamente lo que cada uno de ellos puede brindarle. Otro elemento que se remarca es la experiencia previa del usuario en los diferentes IDEs ya que dependiendo de la comodidad que haya sentido se sentirá motivado o no a continuar su aprendizaje.

2.1. Análisis de los trabajos relacionados

Después de revisar los artículos descritos anteriormente se encontró que hay suficiente información respecto a las modalidades en las que los niñas, niños y adolescentes se desenvuelven mejor y aprenden más como se presenta en (Zhu et al., 2016), (Tisza et al., 2019); así también hay información relevante respecto al manejo de ambientes de desarrollo integrado y de las características que deben cumplir para realmente ser un apoyo para programadores novatos como se refleja en los artículos (Ioannis Karvelas, 2019), (Zayour & Hajjdiab, 2013) y (Department of Computer Science, Interlink Polytechnic, Ijebu Jesa, Osun State, 233114, Nigeria et al., 2016). De lo anterior se desprende que el presente trabajo es relevante y es posible determinar muchos de los requisitos funcionales y no funcionales para el IDE a partir de los artículos analizados; por otro lado, si bien existen otras herramientas que persiguen objetivos parecidos a la presente investigación, como los mencionados en los trabajos (*Scratch - About*, s/f) y (*Turtlestitch - Coded Embroidery / TurtleStitch - Coded Embroidery*, s/f), estos no se encuentran en idioma español.

3. Desarrollo

Tras el análisis de la problemática descrita en la Introducción y el análisis de trabajos relacionados, se determinó crear una herramienta pensada para un lenguaje de programación simple y reducido, que permita al estudiante comprender los conceptos básicos y constatar fácilmente sus resultados. A este tipo de herramientas se les conoce generalmente como Ambiente de Desarrollo Integrado o Ambiente de Desarrollo Interactivo (IDE por sus siglas en inglés).

El lenguaje de programación que se pretende implementar se basa en conceptos matemáticos simples que se cubren en los temarios de matemáticas en México como parte de la educación básica puesto que se tienen en mente en todo momento los postulados de STEM aunque con mayor énfasis en la computación. La mayoría de las instrucciones en lenguajes con resultados gráficos son semejantes y se basan en el idioma inglés como es el caso de Logo y Scratch, la diferencia del lenguaje planteado es que sus instrucciones se basan en el idioma español, considerando instrucciones muy simples como <iniciar>, <terminar>, <avanzar n unidades>, <girar n grados>, <repetir> y <elegir color> entre otros.

3.1. Identificación de requerimientos

Para el desarrollo de una aplicación de software de calidad uno de los puntos más importantes es la utilización de una metodología de software que sirva de guía durante el proceso; la mayoría de las metodologías de desarrollo de software consideran en la primera fase reunir información para tener claro lo que se espera que la aplicación realice, dicha información se recolecta mediante reuniones con el cliente y con usuarios que van a hacer uso de la aplicación, o bien aplicando encuestas a los usuarios potenciales identificados; la información obtenida de reuniones o encuestas se transforma en los requerimientos funcionales y no funcionales a cubrir en la aplicación. En el presente proyecto no existe un cliente y los usuarios potenciales son niñas, niños y adolescentes de habla hispana, por lo que en caso de seguir alguna de las técnicas anteriores se habría optado por aplicación de encuestas a un universo amplio de niñas, niños y adolescentes.

Sin embargo, para la obtención de requisitos del IDE para enseñanza de la programación no se siguió ninguna de las dos técnicas mencionadas en el párrafo anterior debido a que casi todos los IDEs, tanto comerciales como de uso libre, dirigidos a la enseñanza o al desarrollo de software, tienen elementos parecidos. En ese sentido, el análisis de herramientas semejantes como alternativa a la recolección de requerimientos (técnica de análisis comparativo) se muestra en "The Deadline" (DeMarco, 1997), de Tom DeMarco, donde se presentan de forma novelada las ideas respecto a administración de proyectos de software y en general sobre ingeniería de software de uno de los autores más renombrados en el área; con esto en mente, en este caso se optó por recurrir a técnicas de análisis comparativo sobre ambientes de desarrollo integrado y herramientas de apoyo a la enseñanza de programación para identificar los requerimientos funcionales y no funcionales de la herramienta; además, se tomaron en consideración buenas prácticas para la enseñanza de la programación descritas en la literatura.

3.2. Análisis de ambientes de desarrollo integrado (IDEs)

1. Eclipse:
   Fue desarrollado por Eclipse Foundation bajo una licencia de software libre y es multiplataforma, este IDE está pensado para trabajar con Java, aunque también permite trabajar otros lenguajes al añadirle *plugins* por ejemplo para C++, JavaScript, Python o PHP entre otros. Entre las características de este IDE se encuentran autocompletar código, acceso eficiente a archivos del proyecto, facilidad para probar el código y corrección de sintaxis entre otros (Guindon, s/f).
2. Visual Studio:
   Fue desarrollado por Microsoft en febrero de 1997, su tipo de licencia es considerada Freemium ya que cuenta con versiones gratuitas y de pago. Es multiplataforma y cuenta con una amplia colección de *plugins* para aumentar su funcionalidad, por ejemplo, para relacionar el código con controladores de versiones. Entre las características que ofrece a sus usuarios se encuentran (*Visual Studio 2019*, s/f): sugerencia de código, facilidad en la navegación de archivos y sugerencia de acciones, como cambiar nombre de una función o agregar un parámetro, entre otros.

3. NetBeans:
   Es un IDE escrito en el lenguaje de programación Java y es de código abierto (*NetBeans IDE 8.2 Release Information*, s/f). Como en el caso de Eclipse, permite trabajar en Java, aunque soporta otros lenguajes de programación mediante configuración como PHP y JavaScript. Entre las características destacadas con las que cuenta se encuentran: sintaxis resaltada, autocompletado de código, soluciones rápidas y soporte para diferentes marcos de trabajo mediante *plugins*.

3.3. Análisis de herramientas de enseñanza de programación

1. Scratch
   En (*Scratch - About*, s/f) se menciona que Scratch sirve para programar historias interactivas, juegos y animaciones, además de compartir estas creaciones con otros miembros en la comunidad en línea. Scratch ayuda a los jóvenes a aprender a pensar de forma creativa, a razonar sistemáticamente, y a trabajar de forma colaborativa, habilidades que son esenciales para la vida en el siglo XXI.
2. Snap!
   Snap! es un lenguaje de programación visual basado en el modelo de arrastrar y soltar que permite a los estudiantes crear historias interactivas, animaciones, juegos y más, mientras aprenden ideas matemáticas y computacionales. Snap! se inspiró en Scratch, pero también se dirige a estudiantes novatos y más avanzados al incluir y expandir las características de Scratch.
   El diseño de Snap! está influenciado e inspirado por Scratch, del grupo Lifelong Kindergarten en el MIT Media Lab (*Scratch - About*, s/f).
3. Turtlestitch
   Turtlestitch está basado en un lenguaje de programación educativo que se desarrolló a partir de Snap!, para generar patrones para máquinas de bordar. Es fácil de usar, no requiere conocimientos previos en programación y es poderoso para crear patrones de bordado, aunque no está pensado como herramienta para aprender a programar. Es útil para los diseñadores experimentar con la estética generativa y el bordado de precisión, así como una herramienta innovadora para talleres, que combina una introducción a la programación con un resultado háptico (*Turtlestitch - Coded Embroidery / TurtleStitch - Coded Embroidery*, s/f).

3.4. Requerimientos del IDE

De acuerdo a (Ask John: Which IDE should I use with my students? – WeTeach_CS Blog Archive, s/f) un IDE tiene dos partes básicas: un editor y un elemento para compilar/ejecutar el programa editado. La mayoría de los editores tienen comportamientos semejantes y lo mismo puede decirse de los elementos para ejecutar programas. Sin embargo, el IDE propuesto está pensado para la enseñanza de la programación que no se considera en las herramientas descritas en el apartado 3.2, por lo que debe contar también con elementos particulares como son ejemplos dirigidos (tutoriales) y, al ser gráfico, soporte para "arrastrar y soltar" (drag and drop) las instrucciones del programa que son algunos elementos identificados en el análisis del apartado 3.3.

En la Tabla 1 se muestran las listas de requisitos funcionales y no funcionales del IDE obtenidas a partir del análisis comparativo de los apartados 3.2 y 3.3 se encuentran marcados con un asterisco (*) o un signo más (+) respectivamente:

Requisitos funcionales	Requisitos no funcionales
Abrir programa *	Implementación mediante JavaFX
Cerrar programa *	Las instrucciones del lenguaje se presentarán en una lista de la que podrá elegirse un elemento y arrastrarlo hacia el área de edición del programa +
Guardar programa *	La ejecución del código presentará el gráfico obtenido en otra área del IDE +
Crear nuevo programa *	Relación con una máquina de coser/bordar mediante Arduino (a futuro) +
Colocar instrucción +	
Deshacer última acción *	
Procesar código *	
Ejecutar código mostrando diseño resultante +	
Ejecutar código creando bordado (a futuro) +	

Tabla 1 – Requisitos Funcionales y No Funcionales

3.5. Diagrama de casos de uso

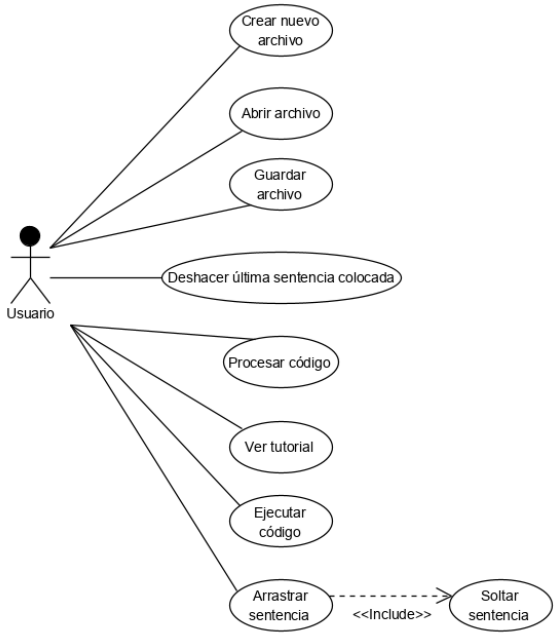

Figura 1 – Diagrama de casos de uso.

Con los requerimientos de la herramienta obtenidos se modeló el diagrama de casos de uso (Fig. 1); como se había mencionado antes, los usuarios potenciales son niñas, niños y adolescentes de habla hispana, mismos que aparecen en el diagrama con el nombre genérico de “Usuario”.

3.6. Arquitectura

Una vez que se contó con los requisitos, se procedió a determinar los componentes de software para la arquitectura, mismos que se muestran en la Fig. 2; en gris se indicaron los componentes más relacionados con el comportamiento del IDE y en blanco los componentes relacionados con la compilación del programa resultante en el lenguaje de diseño gráfico a crear.

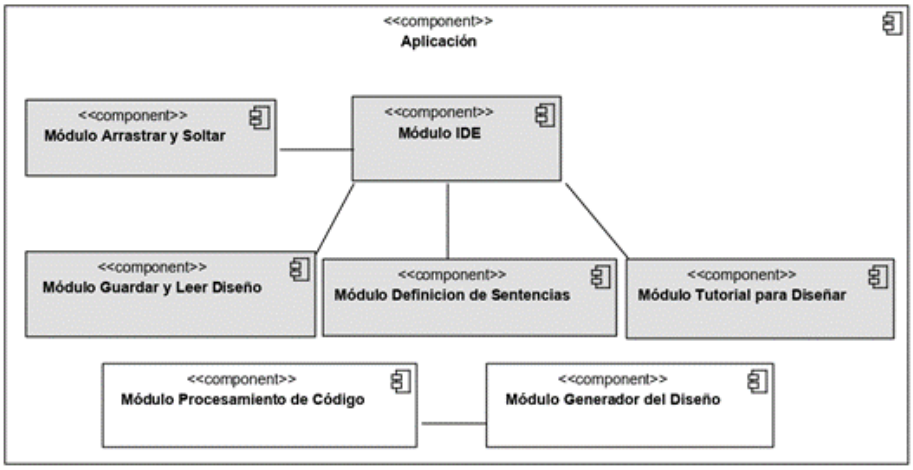

Figura 2 – Componentes de la arquitectura para el ambiente de desarrollo integrado

3.7. Requerimientos del Lenguaje

Para plantear el lenguaje primero se describieron las “frases esperadas”, es decir, cómo serían los programas esperados, posteriormente se describieron los elementos léxicos, gramaticales y semánticas del lenguaje.

Linea	Código
1	Metodo HacerUnaLinea Inicio
2	Girar 90 grados;
3	Avanzar 5 unidades;
4	Girar 270 grados;
5	Pintar 20 unidades;
6	Fin

Tabla 2 – Ejemplo 1 de frases soportadas.

A continuación, se muestran unas posibles frases a considerar en el IDE para obtener los elementos necesarios para la creación del lenguaje.

Linea	**Código**
1	Metodo HacerCuadrado Inicio
2	Repetir 4 Inicio
3	Pintar 5 unidades;
4	Girar 90 grados;
5	Fin
6	Fin

Tabla 3 – Ejemplo 2 de frases soportadas.

- Lexemas o Tokens
  El nombre del token es un símbolo abstracto que representa un tipo de unidad léxica; por ejemplo, una palabra clave específica o una secuencia de caracteres que denotan un identificador. Los nombres de los tokens son los símbolos de entrada utilizados para procesar en el analizador léxico.
  A partir del análisis de las frases mostradas anteriormente se obtuvieron los lexemas o tokens del lenguaje (tabla 4)

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]+

Tabla 4 – Tokens o Lexemas para el Lenguaje

- Gramática
  En la gramática se explica la forma en que los elementos del lenguaje se relacionan para formar textos y se analizan los significados de estas combinaciones.
  En la tabla 5 se muestran las reglas del lenguaje, en la primera columna se encuentra el No Terminal y en la segunda columna la regla correspondiente; el No Terminal "Método" corresponde al axioma de la gramática.

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

Tabla 5 –Gramática del lenguaje de programación

- Semántica
  La semántica verifica el "sentido" de la "frase" mediante la comprobación estática de información.
  Las validaciones semánticas a considerar de acuerdo a las frases estudiadas fueron las siguientes:

  1. Verificar que se haya introducido un número.
  2. Verificar que se haya introducido un ángulo correcto (donde solo se consideraron los ángulos de: 45, 90 y 135 grados).
  3. Verificar que Número sea un Número válido.
  4. Validar que el color sea un color válido (en donde se establecieron solo algunos colores, lo cuales se pueden observar en la parte de tokens).
  5. Verificar inicio y fin de instrucciones.
  6. Verificar inicio y fin del método.

3.8. Resultados

Una vez generada nuestra aplicación de lenguaje mediante ANTLR4 partimos a implementarla en la primera versión del IDE, la cual fue programada con ayuda de java y javaFX, para poder generar una greca como se planteó al principio del artículo.

4. Conclusiones

Al realizar la investigación para el presente trabajo, se identificaron varios puntos relevantes a) una aproximación temprana a la programación es una ventaja competitiva, b) los ambientes gráficos resultan más atractivos para los jóvenes y se despierta mayor interés cuando el resultado es algo visible, c) existe una amplia tradición de bordados en

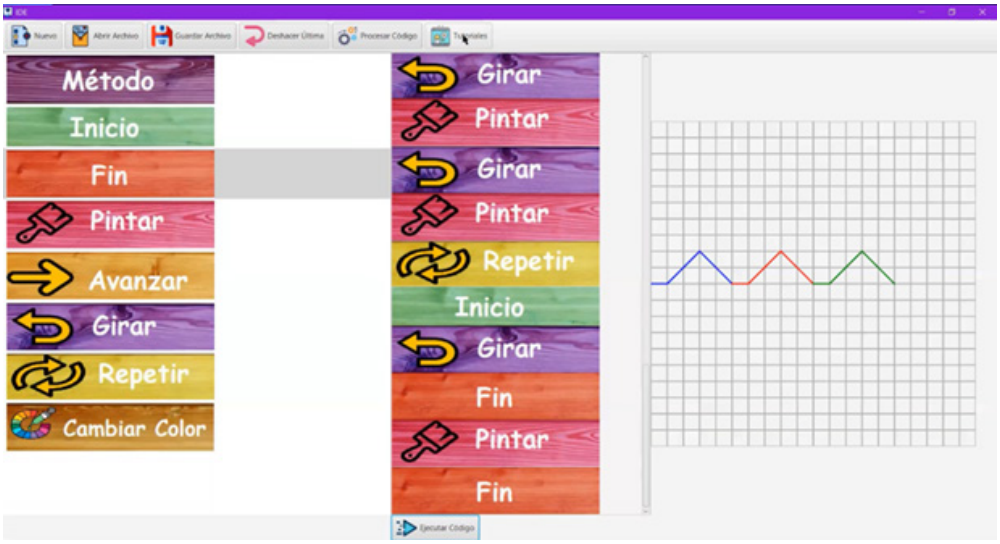

Figura 3 – Resultado de la primera versión de la herramienta.

México, d) el uso de patrones geométricos fortalece el pensamiento lógico matemático y e) en México el porcentaje de estudiantes con conocimientos suficientes del idioma inglés es reducido.

Debido a estos puntos, la creación de un IDE para realizar patrones geométricos como una forma de facilitar el aprendizaje de conceptos básicos de programación y matemáticas es altamente recomendable sobre todo si se brinda al usuario un resultado "tangible" de su programa y utiliza el idioma español.

Para realizar una aplicación o herramienta es necesario conocer los requisitos que debe cumplir y existen varias técnicas para lograrlo, una de ellas es el análisis comparativo de elementos con objetivos semejantes, en este caso se siguió dicha técnica y se analizaron herramientas para la enseñanza de la programación como Snap! y Scratch, herramientas que generan bordados (TurtleStitch) y también IDEs para el desarrollo de aplicaciones; de ahí surgieron los requisitos a cubrir, sin ser una copia de ellos sino la identificación de funcionalidades con diferencias como el ambiente de ejecución (escritorio para la nueva aplicación), el idioma (español), el lenguaje a programar (basado en conceptos matemáticos muy simples), el tipo de resultados esperados (grecas) y el objetivo (enseñanza de la programación) entre otros.

Por otra parte, la generación de un lenguaje propio en español para la herramienta ayuda a que los niños no se sientan intimidados de utilizar la herramienta ya que parten de conceptos familiares para ellos, gracias a la ayuda de ANTLR4 la creación del mismo se vuelve relativamente fácil, ya que nos brinda clases java que pueden ser heredadas para obtener información en el tiempo de ejecución.

Al seguir los lineamientos de la Ingeniería de Software fue posible construir un ambiente de desarrollo integrado gráfico, que considere los aspectos necesarios para incorporar

exitosamente a más niños, niñas y adolescentes a la tecnología disminuyendo las brechas por idioma que se presentan con otras herramientas similares.

Referencias

Ask John: Which IDE should I use with my students? – WeTeach_CS Blog Archive. (s/f). Recuperado el 24 de junio de 2020, de https://sites.utexas.edu/weteachcs/ask-john-which-ide-should-i-use-with-my-students/

Cuantrix (s/f). Recuperado el 13 de mayo de 2020, de https://cuantrix.mx

DeMarco, T. (1997). The Deadline: A Novel about Project Management. Dorset House Pub.

Gómez, L. A. R., Maya, C. J. P., & Villanueva, R. S. L. (2017). Panorama del sistema educativo mexicano en la enseñanza del idioma inglés como segunda lengua.

Guindon, C. (s/f). Eclipse desktop & web IDEs. The Eclipse Foundation. https://www.eclipse.org/ide/

Karvelas, I. (2019). Investigating Novice Programmers' Interaction with Programming Environments. Session 6B: Doctoral Consortium Preseentation, 2.

Logo History (s/f). Recuperado el 22 de junio de 2020, de https://el.media.mit.edu/logo-foundation/what_is_logo/history.html

Movimiento STEM (s/f). Recuperado el 10 de octubre de 2019, de https://movimientostem.org/

Narayanan, V., Albaugh, L., Hodgins, J., Coros, S., & Mccann, J. (2018). Automatic Machine Knitting of 3D Meshes. ACM Transactions on Graphics, 37(3), 1–15. https://doi.org/10.1145/3186265

NetBeans IDE 8.2 Release Information. (s/f). https://netbeans.org/community/releases/82/

Owoseni, A., & Akanji, S. A. (2016). Survey on Adverse Effect of Sophisticated Integrated Development Environments on Beginning Programmers' Skillfulness. International Journal of Modern Education and Computer Science, 8(9), 28–34. https://doi.org/10.5815/ijmecs.2016.09.04

Rojano Ceballos, M. T. (2005). Enseñanza de la Física y las Matemáticas con Tecnología: Modelos de transformación de las prácticas y la interacción social en el aula.

Scratch—About. (s/f). Recuperado el 31 de enero de 2020, de https://scratch.mit.edu/

Secretaria de Educacion Publica. (2005). Programación computacional para matemáticas de secundaria, Libro de actividades para el alumno: Enseñanza de las Matemáticas con Tecnología.

STEM (Science, Technology, Engineering and Mathematics) | British Council México. (s/f). Recuperado el 13 de mayo de 2020, de https://www.britishcouncil.org.mx/stem-science-technology-engineering-and-mathematics

Tisza, G., Papavlasopoulou, S., Christidou, D., Voulgari, I., Iivari, N., Giannakos, M. N., Kinnula, M., & Markopoulos, P. (2019). The role of age and gender on implementing informal and non-formal science learning activities for children. Proceedings of the FabLearn Europe 2019 Conference on ZZZ - FabLearn Europe '19, 1–9. https://doi.org/10.1145/3335055.3335065

Turtlestitch—Coded Embroidery (s/f). Turtlestitch - Coded Embroidery. Recuperado el 31 de enero de 2020, de https://www.turtlestitch.org/page/about

Visual Studio 2019. (s/f). Visual Studio. https://visualstudio.microsoft.com/es/vs/

Yang, S. (2017). Knitting Visualizer: Connecting Craft and Code. Proceedings of the 2017 Conference on Interaction Design and Children - IDC '17, 705–708. https://doi.org/10.1145/3078072.3091985

Zayour, I., & Hajjdiab, H. (2013). How Much Integrated Development Environments (IDEs) Improve Productivity? Journal of Software, 8(10), 2425–2431. https://doi.org/10.4304/jsw.8.10.2425-2431

Zhu, K., Ma, X., Wong, G. K. W., & Huen, J. M. H. (2016). How Different Input and Output Modalities Support Coding as a Problem-Solving Process for Children. Proceedings of the The 15th International Conference on Interaction Design and Children - IDC '16, 238–245. https://doi.org/10.1145/2930674.2930697