UNIVERSIDAD DE GUANAJUATO

CAMPUS IRAPUATO-SALAMANCA
DIVISIÓN DE INGENIERÍAS
DEPARTAMENTO DE ESTUDIOS MULTIDISCIPLINARIOS

PREDICCION DE LA RUTA DE APRENDIZAJE DE PROGRAMACION PARA NIÑOS DE PRIMARIA USANDO INTERACCIONES CON LA INTERFAZ DE USUARIO

TESIS

PARA OBTENER EL GRADO DE DOCTOR EN CIENCIAS DE LA INGENIERÍA

PRESENTA

MAT. JOEL TAPIA FLORES

DIRECTOR DE TESIS

DRA.MARIA SUSANA AVILA GARCIA

CODIRECTOR DE TESIS

DR. MISAEL LOPEZ RAMIREZ

YURIRIA, GUANAJUATO 2026

Resumen

La enseñanza de la programación juega un papel importante en nuestros días, por eso muchas escuelas y organizaciones han buscado la mejor forma para su enseñanza. Una de estas formas son las rutas de aprendizaje, las cuales son secuencias de actividades o ejercicios generados especialmente para las habilidades que tiene el estudiante y de las cuales busca mejorar o adquirir nuevas. Esta forma de enseñanza busca abordar los principios básicos de la programación, y también, el desarrollo de habilidades de pensamiento computacional, el cual ayuda al aprendizaje de la programación y a mejorar la toma de decisiones en su día a día. Por este motivo la presente investigación está orientada a la predicción de las rutas de aprendizaje de programación para niños de primaria.

Por lo anterior se realizó como primera fase una investigación del estado del arte, esto, con el objetivo de ver las investigaciones que se han hecho respecto al tema. Como resultado de esta fase encontramos las principales plataformas que se utilizan para la enseñanza de la programación para niños, las cuales son: Blockly games, Scratch y Code.org. También se encontraron investigaciones que predicen el resultado del siguiente ejercicio a resolver en la plataforma code.org. Investigaciones que definen rutas de aprendizaje usando aprendizaje no supervisado con métodos como vecinos más cercanos en la plataforma Scratch y una investigación que responde si las niñas o niños aprenden más rápido los conceptos de la programación y a qué edad usando la plataforma Blockly games; estas investigaciones se explican de mejor manera en la sección del estado del arte.

Como siguiente fase se modificó uno de los juegos ya existentes la cual fue desarrollada con la biblioteca Blockly. Blockly define una biblioteca de instrucciones visuales, blockly games es una colección de juegos que se desarrollaron como ejemplo y del cuál se tomó el juego del laberinto el cual fue modificado, esto ayudó a que la herramienta estuviera enfocada en la gamificación, se crearon ejercicios de tipo laberinto donde los usuarios deberían de llegar de un punto inicial, a un punto final con el mínimo de bloques posibles y los menores intentos. Se implementó la captura de datos como: el número de intentos que realizó el usuario por ejercicio, el tiempo que tardó en ejecutar una solución entre otras, también permite la captura de interacciones de los usuarios, como, por ejemplo: cuantos movimientos hizo en un bloque, cuantos bloques creó, cuantos eliminó, entre otras. Se realizaron sesiones experimentales en la que se recolectaron datos sobre las sesiones

de trabajo de 26 niños de entre 8 y 10 años y se generó un conjunto de datos el cual fue analizado con distintos métodos como la regresión logística, pero no estaba encaminado al objetivo de la investigación, hasta que, finalmente se optó por utilizar la lógica difusa. Su elección se debió a que este paradigma permite moldear el razonamiento humano por medio de la incertidumbre y la vaguedad que modela, además de contar con grados de pertenencia los cuales permiten que un valor sea analizado y clasificado a un resultado difuso. Por lo anterior, el método de la lógica difusa fue el método elegido para su análisis. Como resultado de este análisis, se realizó la predicción del siguiente ejercicio que deberá realizar el usuario de acuerdo con los datos recabados del ejercicio anterior, esto permitió generar rutas de aprendizaje personalizadas para cada uno de los usuarios de acuerdo con las habilidades que va desarrollando en cada uno de los ejercicios.

Tabla de contenido

Tabla de contenido Ilustraciones

Tabla de contenido Tablas

A mi esposa por su amor incondicional e infinita fe y paciencia en mí, te amo.
A mi hija por mostrarme un nuevo amor.
A mi hijo que esta por nacer, gracias por llegar a ser mi nuevo compañero de vida.
A mis padres, gracias por su guía invaluable e infinita paciencia.
A mis hermanos por su ayuda y amor.
A mi abuelita, pilar fundamental en mi camino,
al fin cumplí con mi promesa, abrazos hasta el cielo.
A mi asesora y asesor por abrirme las puertas. Su apoyo a sido mi brújula en este camino.

Agradezco a la Secretaría de Ciencias, Humanidades, Tecnología e Innovación (SECIHTI) por el apoyo económico y la beca otorgada (934850)para la realización de los estudios de Doctorado en Ciencias de la ingenieria en la Universidad de Guanajuato campus irapuato-salamanca, división de ingenierías, departamento de estudios multidisciplinarios

Capítulo 1

1. Introducción

1.1 Antecedentes

Según [1], la educación informática allanará el camino para hacer del Pensamiento Computacional una alfabetización del Siglo XXI al alcance de todos, ya que se trata de un tipo de pensamiento analítico que según, [2], [3], combina elementos claves de distintas disciplinas: comparte con el pensamiento matemático el uso de conceptos como la abstracción y la descomposición para resolver problemas; integra el enfoque de la ingeniería en el diseño de sistemas; y adopta la metodología científica para comprender la inteligencia y el pensamiento humano.

Además, dado que la programación es una herramienta clave para desarrollar el Pensamiento Computacional [2] [4] países europeos (Francia y España) han incorporado la programación en sus planes de estudio [5], tal como lo advierte [1].

Sin embargo, surgieron interrogantes como: ¿Qué tipo de herramientas son efectivas para desarrollar el pensamiento computacional y cómo diseñar clases que hagan accesibles la programación a estudiantes con diversos perfiles?[1]. Estas cuestiones impulsaron investigaciones sobre estrategias pedagógicas, centradas en herramientas como Scratch [6] y CODE.org [7], para fomentar el pensamiento computacional en edades tempranas.

Pero esto no ha evitado que la personalización del aprendizaje siga siendo un reto. Para abordarlo, [8] propuso un sistema de recomendación de rutas de aprendizaje que sugiere uno o más problemas a resolver para lograr la puntuación objetivo. Para esto, consideraba tres factores claves: 1) el historial de acciones del usuario en un sistema OJ (Online Judge), 2) sus objetivos específicos, y 3) su progreso en la ruta de aprendizaje. Para ello, se categorizaron 2,000 problemas en las categorías "*qué*: clasificación de problemas" y "*cómo:* clasificación de algoritmos". La recomendación se realizó mediante la manipulación de datos, agrupación, capacitación de la red de memoria de largo y corto plazo (LSTM por sus siglas en ingles) y recomendaciones. Como resultado obtuvieron el siguiente

problema a resolver de las categorías "*qué*" y "*cómo*" de acuerdo con los últimos 5 ejercicios enviados por el usuario.

Aunque [8] se enfoca en la personalización de los ejercicios a resolver se enfrenta a otro reto: la dificultad de los lenguajes de texto para niños. Esto ha impulsado métodos como la programación visual [9], que aprovecha las habilidades cognitivas naturales para hacer el aprendizaje más intuitivo.

Un ejemplo de esto es la plataforma Kodetu desarrollada por [10], que analizó las iteraciones de más de 10,000 estudiantes (8-14 años) para generar indicadores sobre la adquisición del pensamiento computacional. Los resultados revelaron que factores como la edad y el género influyen significativamente en el aprendizaje, además de proporcionar: (1) recomendaciones para diseñar plataformas más efectivas, y (2) evidencia sobre el valor de los datos de iteraciones para evaluar procesos cognitivos.

Por su parte [11] aplicó una metodología mixta (K-12, cualitativo – cuantitativo) al analizar: 1) AST (por sus siglas en inglés *Abstract Syntax Trees*), 2) iteraciones y 3) capturas de pantalla en usuarios de 11 a 50 años. Sus resultados muestran que el tiempo de resolución de problemas predice el nivel de habilidad (63% de precisión), al igual que el tiempo entre ejecuciones (55% de precisión), y revelaron mediante métodos estadísticos una correlación positiva entre el uso de estructuras de control/bucles y la experiencia del programador.

Mientras que [12] desarrolló un método para evaluar el proceso de resolución de problemas de los estudiantes a escala en entornos de aprendizaje en línea. Para esto tomó en cuenta las soluciones y acciones de los usuarios, así como las características de los estudiantes, el tiempo y los AST; luego, extrajo las características principales para entrenar y evaluar su modelo de aprendizaje automático. Como resultado obtuvo que su modelo fue capaz de predecir el rendimiento de los estudiantes en la resolución de problemas con una precisión del 85%.

Por otro lado, [13] analizó la presencia de código duplicado (Clones) en proyectos de Scratch y su relación con el pensamiento computacional por parte de los estudiantes. Para lograrlo realizó una recolección de datos acerca las características de los códigos clonados, así como la dificultad del problema, la edad y experiencia del estudiante y su nivel de habilidad en el pensamiento

computacional los cuales fueron analizados mediante la estadística descriptiva y una regresión lineal. Sus resultados mostraron que existe una alta frecuencia de clonación de código, la cual aumenta con la complejidad de los ejercicios, pero la cual no tiene relación directa con el nivel de pensamiento computacional de los usuarios.

Aunque estas investigaciones han identificado factores claves que influyen en el desarrollo del pensamiento computacional (como la edad, genero, interacciones de los usuarios y uso de estructuras complejas), aún falta definir rutas de aprendizaje personalizadas. Por ello, nuestra investigación aborda esta brecha mediante la lógica difusa para la generación de rutas de aprendizaje personalizadas. Este enfoque predictivo analiza el desempeño en ejercicios realizados, y en función de estos resultados determina automáticamente el siguiente ejercicio a resolver, creando así trayectorias adaptativas para cada estudiante.

1.2 Justificación

El desafío para el sector de la educación es mejorar las habilidades de la fuerza laboral futura, pero lo más importante es empoderar a los jóvenes con las competencias para dominar y crear sus propias tecnologías digitales, y prosperar en la sociedad actual [5]. Se cree que enseñar y aprender a codificar, en entornos educativos formales y no formales, jugará un papel importante en este proceso, ya que se fomenta la resolución de problemas, la creatividad y el pensamiento lógico [5].

Es por ello, que en diferentes países de Europa se ha introducido la programación en sus modelos educativos. En la tabla 1 [5] mostramos las razones de su implementación.

Tabla 1. Razones por las cuales se implementó la enseñanza de la programación en programas educativos en Europa [5].

País	Fomentar el pensamiento lógico	Fomentando la resolución de problemas	Atraer estudiantes a la TIC´s	Fomentar las habilidades de codificación	Fomento de la empleabilidad de las TIC´s	Fomentar otras competencias claves
Austria	o	o	o	o	o	o
Bélgica (NL)			o		o	o
Bulgaria	o	o	o	o		
Republica Checa	o	o	o	o	o	o

Dinamarca	o	o				o
Estonia	o	o	o			o
Finlandia	o	o		o		
Francia			o		o	o
Irlanda	o	o	o	o		o
Israel	o	o	o	o	o	o
Hungría	o	o				
Lituania	o			o		
Malta			o	o		
Polonia	o	o	o	o	o	o
Portugal	o	o			o	o
España	o	o		o		o
Eslovaquia	o	o				
UK (Inglaterra)	o	o	o	o	o	

En la tabla 2 [5] se listan los niveles educativos que implementan la enseñanza de la programación en los países europeos (resaltando en color rojo los niveles obligatorios).

Tabla 2. Niveles educativos donde se implementan la enseñanza de la programación [5].

País	Primaria (edad: de 6 a 12 años)	Secundaria inferior (general, edad de 12 a 16)	Secundaria inferior (profesional, edad de 12 a 16)	Secundaria superior (General, edad de 16 a 18)	Secundaria superior (profesional, edad de 16 a 18/19)	Depende del plan de estudio regional o escolar (edad de 15 hasta 20 o más)
Austria		o		o	o	o
Bélgica (NL)	o	o	o			
Bulgaria				o	o	
Republica Checa					o	o
Dinamarca		o		o	o	
Estonia	o	o	o	o	o	o
Finlandia	o	o				

Francia	o	o		o		
Hungría				o	o	
Irlanda		o				o
Israel	o	o	o	o	o	
Lituania		o		o		
Malta				o		
Polonia		o		o	o	o
Portugal		o			o	
Eslovaquia	o	o	o	o	o	
España	o	o		o		o o
UK (Inglaterra)	o o	o o		o o		

Por su parte en México se logró la inscripción escolar del 94% de todos los niños del país en las actividades de la Semana de la Educación en Ciencias de la Computación, que fue realizada en línea entre los 6 y 14 años para el 2010 [14]. Sin embargo, esto contrasta con los datos registrados en el Instituto nacional de Evaluación Educativa (INEE) en cuanto al acceso en México por parte de los estudiantes a las TIC´s (Tecnologías de la información y la comunicación) y el estado de la educación informática. Ya que de acuerdo con su censo del 2019 en nivel preescolar y primaria solo el 28.4% y el 43.1% de las escuelas tienen computadoras para estudiantes, del mismo modo la presencia del internet en nivel preescolar y primaria es del 37% y el 43.1% respectivamente [5]. En la ilustración 1 se puede ver el porcentaje de computadoras e internet en todos los niveles educativos en México.

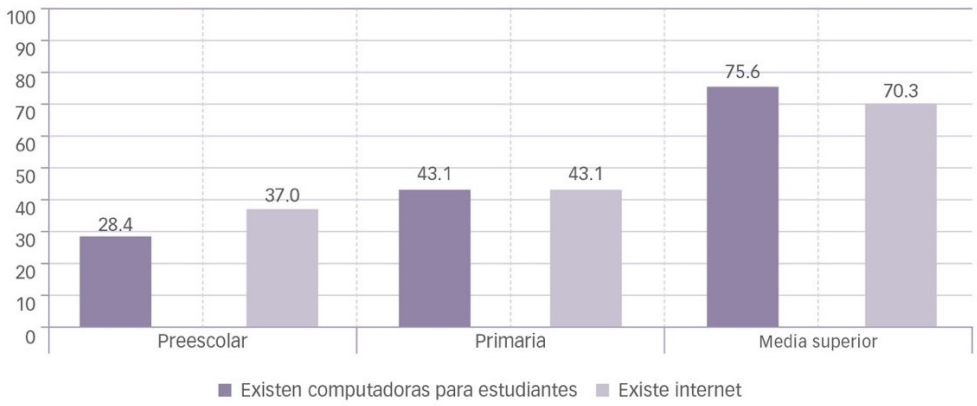


Ilustración 1. Gráfica de porcentaje de computadoras e internet en escuelas de México (Ilustración tomada del INEE 2019).

En los servicios administrados por el Consejo Nacional de Fomento Educativo (CONAFE) en nivel preescolar y primaria, menos del 10% de las escuelas cuentan con computadoras[15]. Por otro lado, en la educación privada cuentan con mayores recursos tecnológicos; en nivel preescolar se cuenta con el 64.2%, en primaria, 89.3% y en las escuelas Media Superior, el 86.0%, los bachilleratos autónomos cuentan con el 93.5% y el Colegio Nacional de Educación Profesional Técnica (CONALEP) con el 95.3% [5].

Esto se puede deber a que México solo avanzó 5 lugares en un periodo de 5 años en la clasificación de la investigación y desarrollo de las tecnologías para ubicarse en el lugar 87 [16]. En la tabla 3 [16] podemos ver los países mejor clasificados y las diferencias que tiene México con ellos.

Tabla 3. Clasificación IDT por país [16].

	Puesto	IDT	Clasificación	IDT
Economía	2012		2017	
Islandia	4	8.58	1	8.98
Suiza	13	7.94	3	8.74
Dinamarca	2	8.78	4	8.71
Reino Unido	7	8.28	5	8.65
…	…	…	…	…
México	**94**	**4.07**	**87**	**5.16**

Sin embargo, esto no queda aquí, ya que en México también se cuenta con un rezago educativo muy importante. Para [16] el rezago educativo en México consta de 4 agentes los cuales se muestran en la ilustración 2.

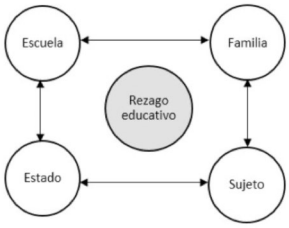


Ilustración 2. Agentes que ayudan al rezago educativo en México.

Cada uno de estos agentes ayuda a generar el rezago educativo en el país. Teniendo mayor impacto los agentes de estado y el sistema educativo. Para este último, lo más importante es "seleccionar a los más "adaptados", fragmentar a los menos aptos para cumplir con su función productora y reproductora de las desigualdades sociales y asegurar que sólo lleguen a la cima aquellos que sean capaces y estén convencidos de aquellas prácticas de sujeción y subordinación son las mejores para continuar con su funcionamiento" (Pág. 16) [16].

Por los factores anteriores, en México la enseñanza de la programación y el desarrollo del pensamiento computacional en niños es un lujo que muy pocos pueden darse, ya que las escuelas públicas no cuentan con una infraestructura que les permita su enseñanza y las pocas que las tienen no cuentan con los maestros calificados para ello o simplemente no tienen ningún interés en que los niños lo aprenden. Esto hace que el rezago educativo cada día sea mayor y nos quedemos muy por debajo de las grandes potencias mundiales.

1.3 Propuesta

Por lo anterior nuestra investigación propone generar rutas de aprendizaje personalizadas utilizando el método de lógica difusa, que utiliza como variables de entrada: el tiempo de resolución, el número de intentos y los errores cometidos en cada ejercicio. Como salida, el modelo determina el siguiente ejercicio a resolver, adaptándose al ritmo y a las habilidades de cada estudiante. Con esto, buscamos fomentar no solo el aprendizaje de la programación, sino también el desarrollo del pensamiento computacional.

1.4 Preguntas de investigación

Las preguntas que se tratarán de responder en esta investigación son:

- ¿Existen conjuntos de datos públicos que contengan datos de la interacción para ser usados en este proyecto?
- ¿Qué datos de la interacción de los niños con una plataforma de programación a bloques pueden usarse para la predicción de rutas de aprendizaje?
- ¿Qué herramientas de aprendizaje automático e inteligencia artificial se pueden usar para predecir rutas de aprendizaje en ejercicios de programación para niños?

1.5 Hipótesis

H1. Los datos de interacción de los niños con la plataforma de programación ayudan a la predicción de rutas de aprendizaje utilizando lógica difusa.

H0. Los datos de interacción de los niños con la plataforma de programación no ayudan a la predicción de rutas de aprendizaje utilizando lógica difusa.

1.6 Objetivo General

Predecir rutas de aprendizaje de programación utilizando la lógica difusa y los datos de las interacciones de los niños con la herramienta de programación, para que la experiencia de aprendizaje se ajuste a su experiencia de uso y desempeño.

1.7 Objetivos Específicos

- Analizar el proceso de aprendizaje de la programación durante la resolución de ejercicios basados en bloque, identificando los patrones de interacción propios del pensamiento computacional.
- Identificar los factores que influyen en el desarrollo del pensamiento computacional en entornos de programación por bloques.
- Diseñar una herramienta de programación por medio de la librería Blockly para la recolección de las interacciones de los usuarios.
- Recolectar las interacciones de los usuarios por medio de la resolución de problemas de programación para su análisis.
- Analizar las interacciones de los usuarios con la ayuda de diferentes técnicas de IA para la generación de las rutas de aprendizaje.

- Generar rutas de aprendizaje por medio de las diferentes técnicas de IA para el desarrollo del pensamiento computacionales en los usuarios.

1.8 Organización del documento

El resto del documento está conformado por el capítulo 2 del marco teórico, en el cual se definen los conceptos que se abordarán para esta investigación. En el capítulo 3 se presentan el estado del arte, aquí se verán los trabajos similares a este y sus diferencias. Para el capítulo 4, se describe la metodología utilizada para llevar a cabo este proyecto. En el capítulo 5 mostramos los resultados que obtuvimos en esta investigación, y en el capítulo 6 se presentan las conclusiones y trabajos a futuro.

Capítulo 2

2. Marco teórico

En este capítulo se definirán los conceptos claves que permitieron llevar a cabo esta investigación. A su vez se mostrarán trabajos relacionados con esta investigación, esto con el objetivo de mostrar diferencias y similitudes con nuestro trabajo de investigación.

2.1 Pensamiento Computacional

Para [3] el pensamiento computacional no solo es resolver un problema, sino también diseñar su solución mediante la interpretación del código como datos y los datos como códigos, es tomar un problema grande y complejo y desfragmentarlo en partes por medio de la abstracción y la descomposición para después modelar los aspectos más importantes de un problema y hacerlos más manejables.

Por su parte el grupo de trabajo [14] indica que el Pensamiento Computacional comparte elementos con diferentes tipos de pensamiento, como el algorítmico, el ingenieril y el pensamiento matemático. Barr [17] nos dice que el pensamiento computacional es aplicar las habilidades intelectuales de la computación y la informática a proyectos en cualquier otra disciplina. Y por último Hemmendinger [18]concluye que el objetivo final del Pensamiento Computacional no es enseñar a todos a pensar como un científico informático, sino enseñar los elementos comunes de la informática y aplicarlos a la resolución de los problemas de las distintas áreas y así generar nuevas preguntas de investigación. En este trabajo se define el Pensamiento Computacional como la habilidad de solucionar problemas en la vida diaria por medio del razonamiento lógico y los conceptos computacionales.

2.2 Dimensiones del pensamiento computacional

En la literatura encontramos que el Pensamiento Computacional está formado por diferentes elementos, los cuales también pueden ser llamados dimensiones [9],[17], [18],[19] habilidades [20] o conceptos y capacidades [17]. En la tabla 4 se muestran las diferentes dimensiones del Pensamiento Computacional encontradas en la literatura con sus respectivos autores.

Tabla 4. Dimensiones del Pensamiento Computacional según su autor.

Dimensiones	Robles Gregorio [13]	Moreno-León [21]	Atmatziduo et al. [22]	Lye & Koh [23]	Pérez-Marín [24]	García-Peñalvo [25]	Barr & Stephes on [17]
Pensamiento Lógico	X	X					
Representación de Datos	X	X					X
Interactividad del Usuario	X	X					
Control de Flujo	X	X					
Abstracción y descomposición de problemas	X	X	X				X
Paralelismo		X					x
Sincronización	X	X					
Generalización			X				
Algoritmo			X				
Modularidad			X				
Concepto Computacional				X	X	X	
Practicas Computacionales				X	X	X	
Perspectivas Computacionales				X	X	X	

Colección de Datos							X
Análisis de Datos							X
Automatización							X
Simulación							X

Las dimensiones presentadas en la Tabla 4 se definen de la siguiente manera:

- **Pensamiento lógico:** Según [13] Es la capacidad de representar las soluciones a problemas como secuencias ordenadas de instrucciones, además, de apoyarse del uso de operadores lógicos y condicionales para el análisis de datos. Por otro lado [21] lo define como a la capacidad de identificar errores en el código (depuración) y comprender la secuencia de acciones para predecir el comportamiento del programa.
- **Representación de datos:** Según [13] Es la habilidad de manejar la información utilizando variables, listas y estructuras de datos para modelar soluciones a los problemas representados. Por otro lado [21] lo define como la capacidad de gestionar información y organizarla dentro del programa. Incluyendo el uso adecuado de variables, listas o estructuras de datos para almacenar y manipular los valores necesarios para la lógica del programa. Por último [17] la define como la estructura de datos como matrices, listas, etc.
- **Interactividad del usuario:** Según [13] Es la capacidad de crear programas que responden a las acciones del usuario (eventos), como clics, entrada de teclado o comunicación entre personas. Por otro lado [21] la define como la habilidad de diseñar programas que respondan a las acciones del usuario (eventos de teclado, clic de ratón) o del entorno, logrando así que el programa sea interactivo y no solo una secuencia estática.
- **Control de flujo:** Según [13] Es usar estructuras que determinan el orden de ejecución de las instrucciones, incluyendo bucles (bucles de repetición), condicionales (if/else) y esperas. Por otro lado [21] lo define como la capacidad de gestionar el orden en que se ejecutan las instrucciones, incluyendo el uso de estructuras condicionales (si entonces) y bucles de repetición (loops) para alterar la secuencia lineal.
- **Abstracción y descomposición de problemas:** Según [13] Es la capacidad de simplificar un problema complejo en solo los detalles relevantes y dividirlos en partes más pequeñas y manejables. Por otro lado [21] la define como la división de un problema complejo en

subprogramas más pequeños, manejables y organizados, ocultando detalles innecesarios y enfocándose en lo fundamental para la resolución de ellos. Por su parte [22] la define como el proceso de dividir los problemas en partes más pequeñas, y así, crear algo simple a partir de algo complejo. Por último [17] la define como el encapsulamiento de un conjunto de comandos que se repiten con frecuencia y que realizan una función definida.

- **Paralelismo:** Según [13] Es la habilidad de gestionar la ejecución simultanea de múltiples secuencias de instrucciones o scripts al mismo tiempo. Por otro lado [21] lo define como a la habilidad de programar acciones que ocurren simultáneamente o al mismo tiempo. Por último [17] lo define como la canalización de datos o tareas de tal manera que se procesan en paralelo.
- **Sincronización:** Según [13] Es la capacidad de coordinar la ejecución de diferentes scripts o actores para que se activen en el momento adecuando, evitando así conflictos en problemas paralelos. Por otro lado [21] la define como la coordinación entre diferentes acciones paralelas mediante el uso de eventos (mensajes, emisores/receptores) para asegurar que ocurran en el orden correcto.
- **Generalización:** Según [22] lo define como el proceso de transferir una solución de un problema a varios problemas.
- **Algoritmo:** Según [22] como las instrucciones paso a paso para llevar a cabo de manera correcta un proceso.
- **Modularidad:** Según [22] la define como el desarrollo de procesos autónomos que encapsulan comandos frecuentes para realizar una función que se puede usar en el mismo problema o diferentes.
- **Concepto Computacional:** según [23] lo define como los conceptos empleados por el programador a la hora de programar: secuencias, bucles, eventos, paralelismo, condicionales, operadores y datos. Por otro lado [24] lo define como los conceptos fundamentales de la informática que los estudiantes aplican a medida que programan. Incluyen nociones como eventos y manejo de datos. Por su parte [25] lo define como las bases teóricas y los elementos fundamentales que los programadores utilizan al crear código o herramientas de programación. Se refieren al ¿qué se está construyendo?
- **Prácticas Computacionales:** según [23] las define como la solución de los problemas que ocurren en el proceso de la programación. Por otro lado [24] las define como al desarrollo de procedimientos y estrategias de pensamiento al programar, tales como la depuración de

proyectos (debugging), la reutilización, remezclas de soluciones, y la abstracción y modularización. Por su parte [25] las define al ¿cómo se aplican los conceptos?, enfocándose en el proceso de desarrollo y el enfoque de aprendizaje más que al producto final.

- **Perspectivas Computacionales:** según [23] las define como la comprensión que tienen los estudiantes de sí mismos y sus relaciones con los demás y el mundo tecnológico. Por otro lado [24] las define como la evolución de la visión que los estudiantes construyen sobre sí mismos y sobre el mundo al tratar de resolver un problema, fomentando la autoconfianza, la colaboración y la percepción de sí mismos como creadores de tecnología. Por su parte [25] las define como a la evolución en la mentalidad del estudiante, cambiando su perspectiva sobre sí mismo y bore la tecnología. Es el ¿qué aprendí sobre mí y el mundo?
- **Colección de datos:** según [17] la define como la acción de recolección de datos para un área problemática.
- **Análisis de datos:** según [17] lo define como la creación de un programa para realizar cálculos estadísticos sobre un conjunto de datos.
- **Automatización:** según [17] la define como la sistematización directa del algoritmo.
- **Simulación:** según [17] la define como la generación de algoritmos de animación y el barrido de parámetros.

2.3. Ruta de Aprendizaje

Para [26][21] las rutas de aprendizaje significan una guía para la designación de proyectos (de nivel de menor a mayor) para el aprendizaje de un tema en específico. En cambio, para [27] una ruta de aprendizaje se trata de la creciente capacidad para juzgar o evaluar lo que podría ser o no factible dado la tecnología y el conjunto de habilidades que dispone el sujeto.

Mientras que para [28] las rutas de aprendizaje deben de ser flexible y debe de tener como objetivo la inclusión de diversos conocimientos, intereses y necesidades de los estudiantes. Su objetivo es alinear la diversidad en el grupo objetivo y las trayectorias de aprendizaje flexibles para que cada estudiante elija el camino correcto en función de sus conocimientos e intereses.

En este caso las rutas de aprendizaje se definirán prediciendo el siguiente ejercicio que deberá de realizar el usuario, el cual se asignará dependiendo de la forma en que solucionó el ejercicio anterior. Ejemplo si un usuario realiza el ejercicio uno y su solución fue de manera óptima pasará a un ejercicio más avanzado como el ejercicio 4, si lo solucionó de manera regular pasará al ejercicio 3, pero si lo solucionó de una manera incorrecta volverá a intentar ese mismo ejercicio (ejercicio 1). Con esto se garantiza que cada usuario vaya avanzado a su propio ritmo y a las habilidades que vaya desarrollando y obteniendo en cada uno de los ejercicios de programación, esto es a lo que se llama rutas de aprendizaje.

2.4. Árboles de Sintaxis Abstractos (AST)

Un árbol de sintaxis abstracta (*AST*, por sus siglas en inglés) se puede definir como una representación estructurada en árbol del código fuente [29]. Por ejemplo, la expresión 3 + (4 * 5) se estructura (ver ilustración 3) con + como nodo raíz, teniendo como hijos el número 3 y un subárbol * (con 4 y 5 hojas).

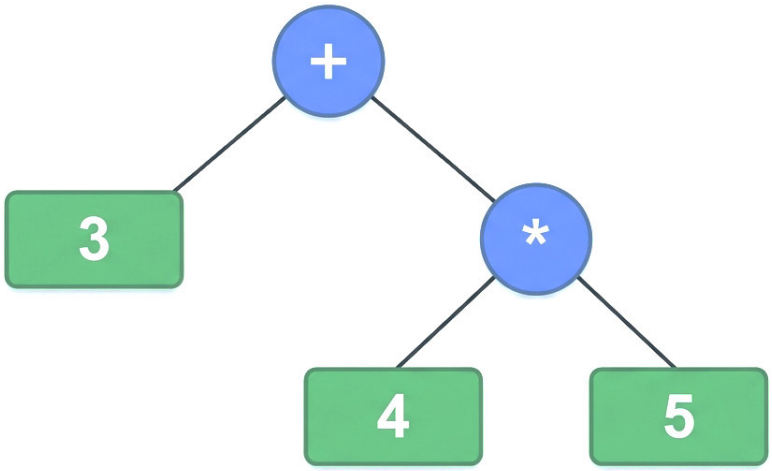


*Ilustración 3. AST de la expresión 3 + (4*5).*

Desglosamos la expresión x = 10 + y para mostrar otro ejemplo (ver ilustración 4), donde: la raíz es el nodo de asignación (=), el hijo del lado izquierdo es la variable x, el hijo del lado derecho es el nodo de operación binaria (+), que a su vez tiene como hijos el literal 10 y la variable y.

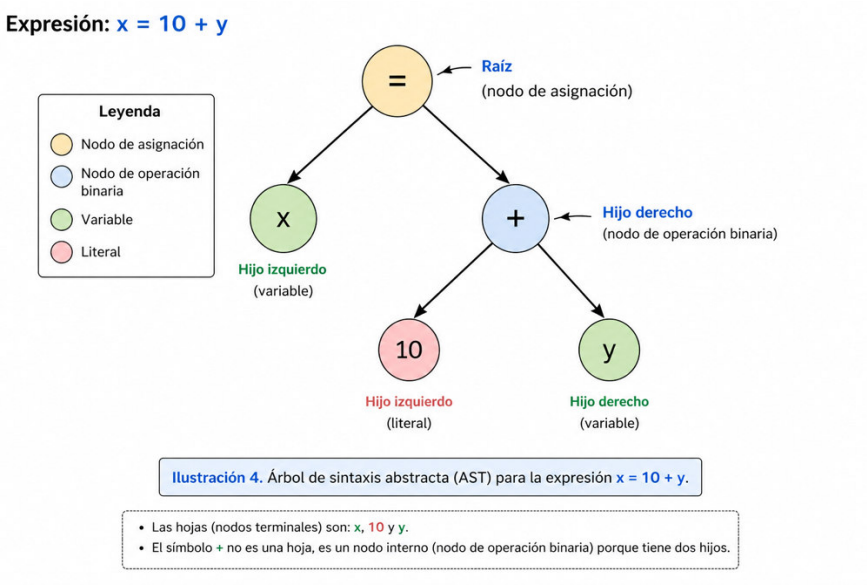


Ilustración 4. AST de la expresión x = 10 + y.

Los conceptos fundamentales de los AST son [30]:

- Abstracción: la cual no incluye todos los detalles del código fuente real, centrándose solo en los elementos estructurales y la lógica.
- Representación intermedia: sirve como representación intermedia antes de la generación de código o la ejecución, facilitando la optimización y la detección de errores.
- Nodos: Cada elemento en un AST se conoce como nodo. Los cuales representan constructores del lenguaje como operadores (+, -), declaraciones (if, while), literales (5, “hola”) o identificadores (variable).

Mientras que sus elementos incluyen [30]:

- Raíz: Es el nodo principal que suele representar el programa completo o la unidad de compilación (ejemplo: Program o Block).
- Nodos internos: representan operaciones o estructuras de control (ejemplo: BinaryExpression, IfStatement)
- Hojas: Son los operandos o valores literales (ejemplo: números, nombres de variables, cadenas) que no tienen hijos.
- Metadatos: Los ASTs incluye información como el número de línea y columna del código fuente original para mejorar los mensajes de error.

Los AST han sido utilizados para [31]:

- Compiladores e intérpretes: analizar el significado y generar el código ejecutable.
- Análisis estático (Linters): herramientas como ESLint analizan el AST para detectar vulnerabilidades, problemas de rendimiento o incumplimiento de reglas de estilo sin ejecutar el código.
- Refactorización y Transformación de Código: herramientas que modifican automáticamente el código fuente también manipulan el AST.
- Resaltado de Sintaxis (Syntax Highlighting): Los IDEs usan el AST para entender qué partes son variables, funciones o palabras claves y colorearlas.

Una definición de AST es una representación gráfica en la que el símbolo inicial de una gramática deriva a una cadena de lenguaje. A continuación, se muestra la estructura de un archivo en XML el cual será transformado en un AST:

```
<xml xmlns="http://www.w3.org/1999/xhtml">
        <block type="move" id="b}opX/7(b9=;Vz.];)A+" x="8" y="41">
                <next>
                <block type="move" id="kf,le($R):m4Ra_{NqO+">
                        <next>
                        <block type="move" id="tjoGRHxI,xGv@WFo~~R}">
                                </block>
                            </next><
                        </block>
                </next>
        </block>
</xml>
```

- el archivo XML anterior comienza con la etiqueta **<xml>**la cual indica el inicio de la estructura XML esta etiqueta es representada en un AST de tipo JSON con una llave **{** la cual indica que es el inicio de la estructura de un AST.
- La siguiente etiqueta que se utiliza para la generación del AST es la etiqueta **<block type>** en ella se encuentra el tipo de bloque que se creó, su ID, su posición y su campo (si es que el

bloque tiene campo). Esta etiqueta es representada por los campos **"type", "id", "x", "y", "fields"** en la estructura del AST.

- La etiqueta **<next>** que aparece en el XML indica la terminación del primer nodo y el inicio del siguiente en el archivo AST su sintaxis es: **"next": {**
- Estas etiquetas se repiten en los archivos dependiendo de los bloques creados, hasta que aparece la etiqueta **</xml>** el cual indica la terminación del archivo XML en el AST se representa con el símbolo **}** el cual indica la terminación de los nodos en el AST.

En la tabla 5 mostramos el código en formato XML y el resultado de su conversión a AST, cada uno de estos archivos tiene especificado un color diferente dependiendo de la etiqueta que se trate. Esto para facilitar la observación del cambio de XML a AST (ejemplo para las etiquetas de bloques su color es el azul).

Tabla 5. Conversión de código XML a AST.

XML	AST

<**xml**
xmlns="http://www.w3.org/1999/xhtml
"><block type="move"
id="b}opX/7(b9=;Vz.];)A+" x="8"
y="41"><next><block type="move"
id="kf,le($R):m4Ra_{NqO+"><next><blo
ck type="move"
id="tjoGRHxI,xGv@WFo~~R}"></block>
</next></block></next></block></xml>
	{
	"type": "move",
	"id": "b}opX/7(b9=;Vz.];)A+",
	"x": "8",
	"y": "41",
	"fields": {},
	"next": {
	"type": "move",
	"id": "kf,le($R):m4Ra_{NqO+",
	"x": null,
	"y": null,
	"fields": {},
	"next": {
	"type": "move",
	"id": "tjoGRHxI,xGv@WFo~~R}",
	"x": null,
	"y": null,

	"fields": {} } } }

En la ilustración 5 se muestra el AST del ejemplo anterior en su forma gráfica, el cual está constituido por 3 nodos que son los 3 bloques de movimiento que utilizó el usuario para llegar a una posible solución.

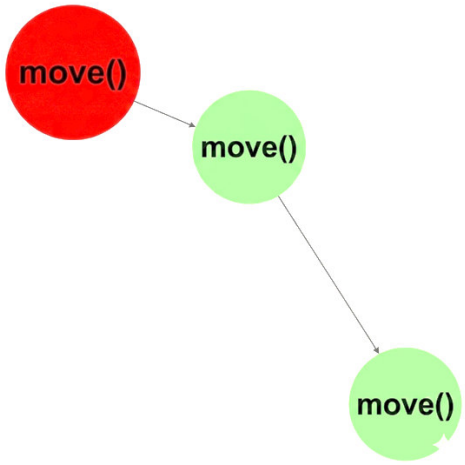


Ilustración 5. AST generado.

De manera formal, dada una gramática libre de contexto, la cual es una gramática formal en la que cada regla de producción es de la forma V -> w, donde V es un símbolo no terminal y w es una cadena de terminales y no terminales, un árbol de análisis sintáctico tiene las siguientes propiedades:

- La raíz se etiqueta con el símbolo inicial. (1)
- Cada hoja se etiqueta como *terminal* o con ϵ. (2)
- Cada nodo interior se etiqueta con un no *terminal*. (3)
- Si A es el *no terminal* que etiqueta a cierto nodo interior y X1,X2,…Xn son las etiquetas de los hijos de ese nodo de izquierda a derecha, entonces debe de haber una producción A --> X1X2…Xn. Aquí, cada una de las etiquetas X1,X2,…Xn representa a un símbolo que puede ser o no un *terminal*. Como un caso especial si A --> ϵ es una producción, entonces un nodo etiquetado como A puede tener un solo hijo, etiquetado como ϵ. En la ilustración 6 se

muestra un AST con las propiedades anteriores, donde su raíz es el signo “=” igual, sus terminales son: “c” y el signo “+”, mientras que las no terminales son: “a” y el número “5”.

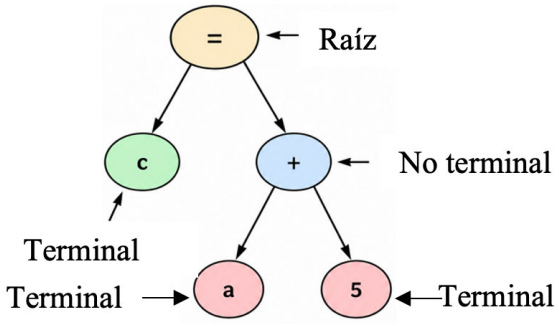
- 

Ilustración 6. Propiedades de un AST.

De acuerdo con [31] para la generación de los AST se necesitan cuatro pasos a seguir: A) **Análisis léxico y tokenización:** El código fuente se divide primero en unidades básicas llamadas tokens (palabras claves, identificadores, operadores). B) **Análisis sintáctico (Parsing):** un analizador sintáctico toma la lista de tokens y la compara con la gramática del lenguaje para construir el AST. C) **Estructura del árbol:** El analizador crea los nodos del árbol siguiendo las reglas gramaticales. Por ejemplo, una sentencia condicional if puede convertirse en un solo nodo con ramas para la condición y el cuerpo if. D) **Detección de errores:** Si el código no cumple con la gramática del lenguaje, el analizador detecta un error de sintaxis y detiene el proceso. Por ejemplo, en la ilustración 7 se muestra un AST con condicional if, donde la raíz es if, y tiene como hijos los signos “>” e “=” y call. El signo “>” tiene como hijos “a” y “b”, “=” tiene como hijos “a” y “c” y call tiene como hijos “fun” y “c”.

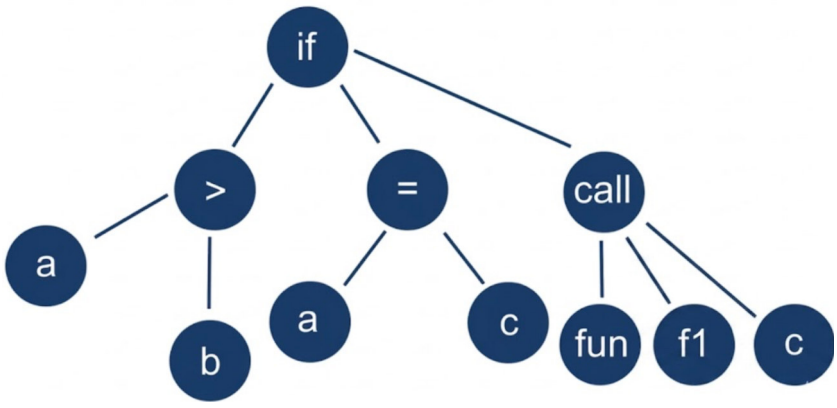


Ilustración 7. AST con condicional if.

2.5. Programación en Bloques

Existen diferentes formas de enseñar programación, y cada una de ellas puede estar destinada a cierto grupo de personas. Por ejemplo, para un niño aprender programación en un lenguaje basado en texto puede hacer aburrido, extenuante y difícil de aprender. Por ello se tienen diferentes métodos para su enseñanza, como la programación visual, la cual simplifica la programación tomando como base las habilidades del razonamiento humano [9], en esta programación se utiliza más de una dimensión para transmitir la semántica. Cada relación u objeto multidimensional (ejemplo, imagen y sonido) es una ficha, una o más fichas es una expresión visual. Cuando la sintaxis de un lenguaje de programación incluye expresiones visuales, entonces es un lenguaje de programación visual (*VPL*, por sus siglas en inglés) [32].

Un VPL permite al usuario crear programas mediante los elementos gráficos en lugar de especificar la sintaxis textual de la programación. Los VPL son utilizados popularmente para los sistemas educativos, multimedia, videojuegos, simulación de sistemas, almacenamiento de datos y análisis de negocios. A continuación, en la tabla 6 se presentan la comparación entre los VPL de código abierto y de código privado según [33].

Tabla 6. Comparación entre VPL de código abierto y privado según [29].

Tipo de VPL	Nombre de VPL	Ambiente de programación	Licencia	Repositorio de proyecto	Plataforma soportada
Código abierto	Node-Red	Web	Open Source-Apache 2.0	GitHub	Raspberry Pi, BeagleBone Black, Docker, Arduino, Android, IBM Bluemix, Amazon Web Service, Microsoft Azure under.
	NETLab Toolkit	Web	GPL	Privado	Arduino, Linux y plataformas privadas con sistemas embebidos como la Raspberry Pi, Intel Galileo, and Arduino
	Ardublock	Web	GPL	Privado	Arduino
	Scratch for Android (S4A)	Web	GPL2	Privado	Arduino

	Modkit	Desktop	GLP2	Privado	Arduino, littleBits, Photon, MSP340, Tiva C
	miniBloq	Deskyop	RMPL	Privado	Multiplo, Arduino, RedBot, and RedBoard.
	NooDL	Web	NEUL	Privado	Arduino, Android
Código Privado	DGLux5	Desktop	DGLux Engineering License	Privado	Raspberry Pi, BeagleBone, DGBox
	AT&T Flow Designer	Desktop	GPL3	Github	AT&T IoT SIM
	Reactive Blocks	Desktop	EPL	Privado	Modbus, Raspberry P and USB Camera
	GraspIo	Desktop	BSD	Privado	Arduino, Raspberry Pi, Gio Arm, GIO TetraPod, and GraspIO boards, Android
	Wyliodrin	Web	GPL3	Privado	Arduino, BeagleBone, Black, Raspberry Pi, Intel Galileo, Intel Edison, UDOO, ZedBoard and Red Pitay
	Zenodys	Desktop	-------	Privado	Raspberry Pi, Zenobox

2.6. Herramientas

En la tabla 6 se compararon algunos VPL disponibles en el mercado, pero no son los únicos, tal es el caso de los 3 VPL más recomendados del mercado: Scratch, CODE.org y Blockly games.

2.6.1 Scratch

Scratch es una plataforma de programación para niños que usa un lenguaje de programación visual con una interfaz sencilla que permite a los jóvenes crear historias, juegos y animaciones. Scratch está diseñado, desarrollado y administrado por la fundación Scratch, una organización sin fines de lucro[34]. En la ilustración 8, se muestra la interfaz de programación de Scratch.

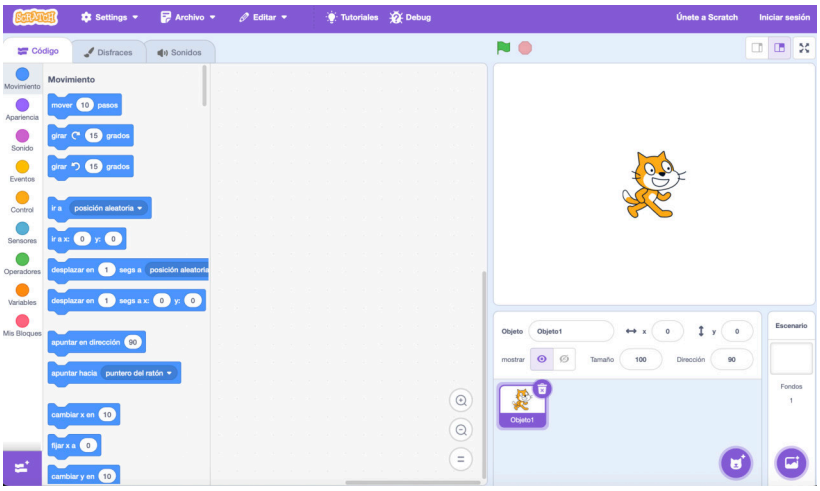


Ilustración 8. Interfaz de programación en Scratch.

2.6.2 CODE.org

CODE.org es una organización sin fines de lucro de innovación educativa dedicada a la visión de que cada estudiante en cada escuela tiene la oportunidad de aprender ciencias de la computación como parte de su educación básica K-12. CODE.org ofrece el acceso a las escuelas, con un enfoque en incrementar la participación de mujeres jóvenes y estudiantes de otros grupos subrepresentados. Es el proveedor líder de currículos de informática K-12 en los distritos escolares más grandes de Estados Unidos, CODE.org también organiza la campaña anual Hora de Código, que ha involucrado a más del 15% de todos los estudiantes en el mundo. CODE.org cuenta con el apoyo de donantes como: Microsoft, Amazon, Google y muchos otros [35]En la ilustración 9 se muestra la interfaz de programación de CODE.org.

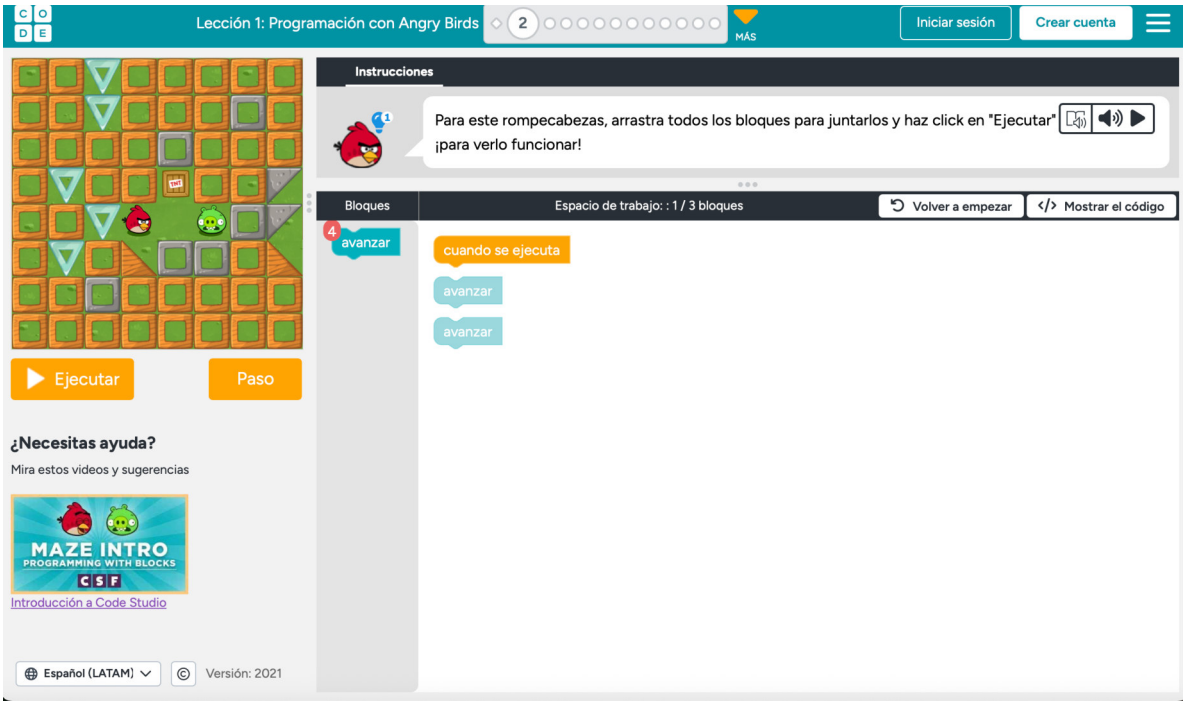


Ilustración 9. Interfaz de programación en CODE.org.

2.6.3 Blockly

Blockly es una biblioteca que agrega un editor de código visual a aplicaciones web y para dispositivos móviles. El editor de Blockly usa bloques gráficos entrelazados para representar conceptos de código como variables, expresiones lógicas, bucles y mucho más. Permite a los usuarios aplicar principios de programación sin tener que preocuparse por la sintaxis[36]. En la ilustración 10 se muestra el resultado de agregar la biblioteca Blockly a una plataforma web.

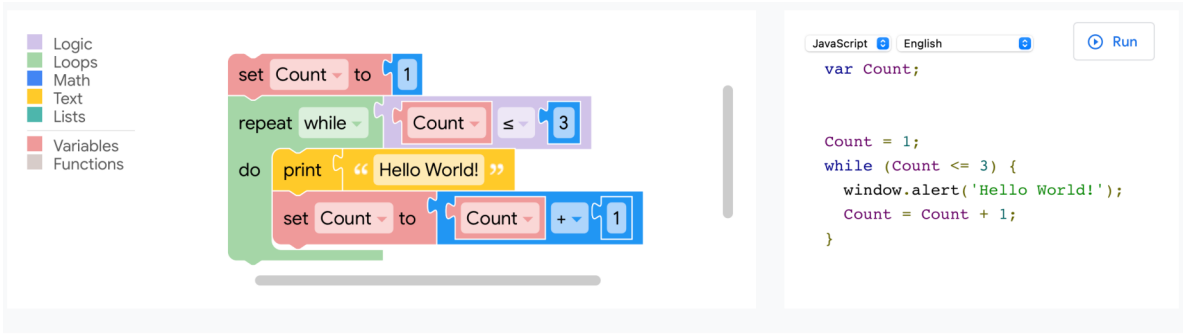


Ilustración 10. Ejemplo de la inserción de bloques con ayuda de la biblioteca Blockly (tomada de https://developers.google.com/blockly?hl=es-419).

2.6.3.1 Blockly Games

Blockly contiene una serie de juegos educativos que enseñan programación llamados Blockly Games. Está diseñada para que niños sin experiencia en programación aprendan sus conceptos y al final de sus retos estén listos para pasar a un lenguaje de programación basado en texto. Las habilidades informáticas ayudan a los estudiantes a colaborar, crear y hacer que casi todos los temas parezcan más relevantes. Diseñados para funcionar a su propio ritmo, Blockly Games se puede descargar para usarlo sin conexión, lo que garantiza la accesibilidad para todos los estudiantes y la tecnología. Todo el código es de código abierto, lo que significa que es gratuito y personalizable para satisfacer las necesidades de sus usuarios (Blockly Game, 2023). En la ilustración 11 se muestra uno de los ejercicios de laberinto que ofrece la plataforma Blockly Games.

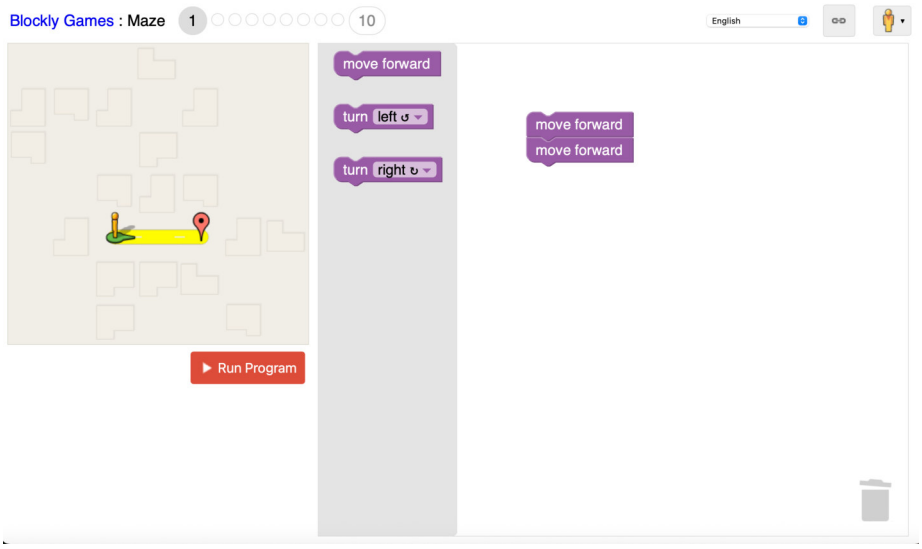


Ilustración 11. Interfaz de programación en Blockly Games.

2.7 Inteligencia Artificial

La inteligencia artificial (IA) es una rama de la informática centrada en la creación de sistemas o máquinas capaces de realizar tareas que normalmente requieren inteligencia humana. Estas tareas incluyen aprendizaje, razonamiento, resolución de problemas, percepción, comprensión del lenguaje y toma de decisiones[37].

Por otro lado, para[38] la IA, es la capacidad de un sistema para interpretar datos externos, para aprender de ellos y utilizar estos aprendizajes para lograr objetivos y tareas específicas a través de una adaptación flexible.

Mientras que [39] define la IA como la capacidad de aprender de la experiencia, retener los conocimientos y la capacidad de dar una respuesta rápida y exitosa a una nueva situación.

A continuación, se muestran las técnicas de regresión lineal y arboles de decisión que en un principio se tomaron como opciones para este proyecto, pero al final sus resultados no ayudaban a cumplir con el objetivo de este proyecto.

2.8 Regresión Lineal

Según [40] define a la regresión, como la técnica estadística que consiste en calcular dicha similitud en forma de función matemática. Además de listar las técnicas más elementales que son: la regresión lineal y la regresión lineal múltiple.

Por su parte [41] menciona que la regresión lineal consiste en encontrar (aproximar) los valores de una variable a partir de los de otra, usando una relación funcional de tipo lineal, es decir describe la relación entre dos variables X e Y. Cuando la asociación entre ambas variables es fuerte, la regresión nos ofrece un modelo estadístico que puede alcanzar finalidades predictivas.

2.9 Árboles de decisión

Un árbol de decisión es un modelo de predicción cuyo objetivo principal es el aprendizaje inductivo a partir de observaciones y construcciones lógicas. Son muy similares a los sistemas de predicción basados en reglas, que sirven para representar y categorizar una serie de condiciones que suceden de forma sucesiva para la solución de un problema [42]. En la ilustración 12, se muestra la estructura de un árbol decisión.

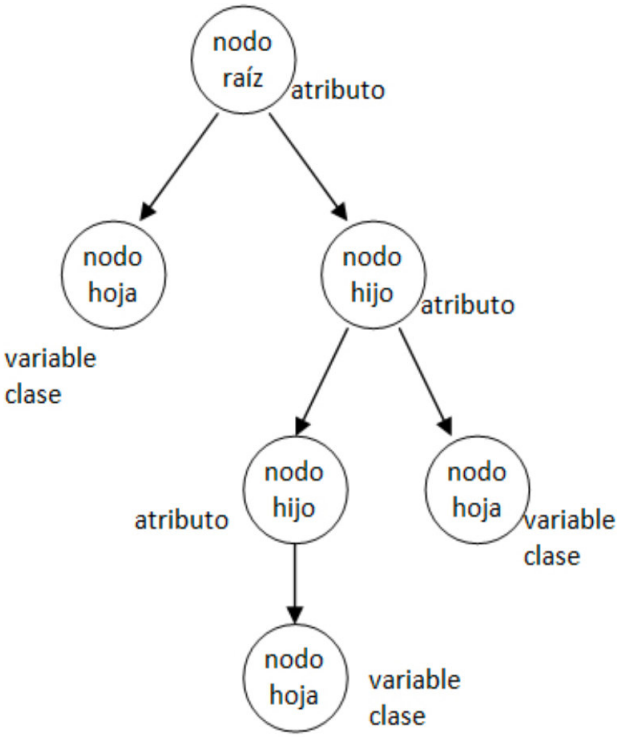


Ilustración 12. Estructura de un árbol de decisión.

Para [43] los árboles de decisión se presentan como una herramienta efectiva para la predicción de probabilidades de incumplimiento, no solo a nivel de capacidad de discriminación (potencia), estabilidad a través del tiempo, sino como una herramienta de fácil entendimiento que permite potencializar sus usos y servir además como herramienta de predicción.

2.10 Lógica Difusa

La lógica difusa es una rama de la inteligencia artificial que aborda el razonamiento aproximado no exacto. Esto se refiere a una generalización de las diversas lógicas multivaluadas. Además, puede controlar o describir sistemas utilizando reglas de sentido común que se refieren a cantidades indefinidas[44].

Un sistema de lógica difusa (SDL) es único porque puede manejar simultáneamente datos numéricos y conocimiento lingüístico. Es una asignación de características, que asigna números a números. La teoría de conjuntos difusos y la lógica difusa establecen las particularidades de la asignación no lineal [45].

La lógica difusa es un enfoque matemático que permite manejar la imprecisión y la incertidumbre, yendo más allá de los valores binarios de “verdadero” o “falso” de la lógica clásica para aceptar grados de verdad entre 0 y 1, imitando el razonamiento humano mediante conjuntos difusos y variables lingüísticas (como “alto”, “bajo”, “cerca”) en reglas “SI-ENTONCES” para sistemas de control y toma de decisiones, usada en electrodomésticos, robótica y diagnóstico médico [44].

La lógica difusa contiene una serie conceptos claves, los cuales se mencionan a continuación:

- **Grados de Pertenencia:** A diferencia de la lógica clásica (0 o 1), en la lógica difusa un elemento puede pertenecer parcialmente a un conjunto, con un valor entre 0 y 1.
- **Conjuntos Difusos (Fuzzy Sets):** Representan categorías imprecisas (ejemplo: temperatura cálida) y pueden solaparse, permitiendo que una misma entrada pertenezca a varios conjuntos con diferentes grados.
- **Variables Lingüísticas:** Palabras como muy alto, moderado, cerca se pueden utilizar para describir valores difusos.
- **Reglas Difusas SI-ENTONCES:** Conectan entradas difusas con salidas difusas, como: SI la distancia es cerca Y la velocidad es alta, ENTONCES reducir aceleración [46].

El funcionamiento de la lógica difusa se puede resumir en los siguientes 3 pasos:

- **Fuzzyficación**: Se traduce la entrada numérica (ejemplo, 25°C) a grados de pertenencia a conjuntos difusos (ejemplo, 0.2 templado, 0.8 cálido).
- **Inferencia Difusa**: Se aplican las reglas “SI-ENTONCES” utilizando estos grados de pertenencia.
- **Defuzzyficación**: Se convierte el resultado difuso (ejemplo, una aceleración “ligeramente negativa”) a un valor numérico concreto para la acción [45].

Algunas aplicaciones donde se utiliza la lógica difusa son:

- **Control Automático**: Lavadoras, aires acondicionados, frenos ABS, control de tracción en vehículos.

- **Robótica**: Robots que navegan en entornos inciertos.
- **Medicina**: Diagnóstico asistido por computadora, procesamiento de imágenes médicas.
- **Finanzas y Sistemas Expertos**: Para manejar información imprecisa y emular el razonamiento de un experto humano.

Capítulo 3

3. Estado del arte

En este capítulo se muestran los resultados de las investigaciones encontradas en la literatura acerca de la generación de rutas de aprendizaje por medio de las herramientas de inteligencia artificial y el aprendizaje automático. La revisión arrojó un número selecto de investigaciones enfocadas en este nicho. A continuación, los trabajos se presentan organizados progresivamente: primero, aquellos enfocados en la mejora de algoritmos de predicción y trazado de conocimiento (Knowledge Tracing) en programación general, seguido por los modelos de detección temprana, y finalmente, aquellos estudios aplicados específicamente a entornos de programación visual por bloques orientados a principiantes y niños.

Comenzando con la mejora de los modelos predictivos, el estudio presentado por el autor en [47] propone scrML-DKT (es una extención de Code-DKT que utiliza representaciones de código basada en srcML, lo que permite la extracción de características tanto de código analizable como no analizable) una mejora del modelo Code-DKT (es un método de seguimiento del conocimiento que utiliza redes neuronales recurrentes para modelar el progreso de aprendizaje del estudiante) que utiliza el formato srcML (permite la extracción de características tanto de código analizable como no analizable) en lugar de los AST llamada scrML-DKT, este modelo permite capturar detalles estructurales y sintácticos del código directamente del texto, incluso si el código presenta errores o está incompleto. Esto permite analizar absolutamente todos los intentos de los estudiantes. Para probar su modelo, utilizaron más de 610 estudiantes universitarios. Los estudiantes resolvieron seis tareas de programación que se enfocaban progresivamente en el uso de sentencias condicionales (if). Como técnicas de análisis de datos utilizaron respresentaciones de código basda en srcML combinada con secuencia de intentos de los estudiantes (pares problema – intentos). Está técnica busca solucionar la limitación estructural de los AST cuando el código del estudiante tiene errores de sintaxis y no puede ser parseado por los métodos tradicionales. Además de utilizar una extensión de Deep Knowledge Tracing (DKT) utilizando Redes Neuronales Recurrentes (RNN). Como resultado obtuvieron que el modelo demostró una probabilidad de predicción superior respecto a los modelos estándar basados únicamente en AST, prediciendo con una precisión del 83% la probabilidad de que

un estudiante responda correctamente en su siguiente intento, incluso cuando el código previo estaba incompleto o no era ejecutable.

Por su parte [48] desarrolló un modelo predictivo explicable basado en inteligencia artificial explicable o XAI, que pronostica las calificaciones del examen final de los estudiantes utilizando únicamente la información de los envíos de sus tareas de programación en etapas tempranas del curso. Para esto utilizaron 545 estudiantes universitarios inscritos en un curso introductorio de programación. De los datos recolectados extrajeron características directamente del comportamiento de programación de los estudiantes, tales como: tiempo invertido, número de problemas intentados, envíos correctos e incorrectos, errores de compilación y la cantidad de cambios en el código (distancia de edición). Para la predicción, construyeron un modelo de ensamble apilado (stacked ensemble) respaldado por redes neuronales convolucionales (CNN) y redes LSTM (Long Short-Term Memory) que combina las fortalezas de los algoritmos KNN, SVM y XGBoost, utilizando regresión lineal para consolidar la decisión final. Como resultado, obtuvieron que el modelo logró predecir el 85% de las calificaciones finales de los estudiantes utilizando únicamente la información de los envíos de programación de las primeras tres semanas del semestre, permitiendo una detección temprana de estudiantes en riesgo.

En la misma linea el trabajo de [49] habla acerca de los modelos tradicionales de Knowledge Tracing los cuales intentan predecir si un alumno acertará o fallará un ejercicio y tratan a estos problemas como etiquetas simples (por ejemplo: ejercicio de bucles), las cuales son insuficientes por: a) la complejidad del código: un estudiante resuelve el ejercicio de manera incorrecta e ineficiente, y el modelo actual no entiende que parte específica del código está mal, b) Relación oculta: existe una interacción entre las habilidades de programación y la calidad del código enviado, las cuales no se están aprovechando para hacer predicciones precisas. Por lo anterior, el autor propone un nuevo modelo que integra dos fuentes de información: 1) habilidades (Skills) mapea qué conceptos de programación (ejemplo: recrusión, arreglos, condicionales) se requieren para cada ejercicio. 2) código del estudiante: utiliza técnicas de procesamiento de lenguaje natural (basado en atención, el cual es un subcampo de la lingüística computacional, centrado en modelos de inteligencia artificial (IA), que interpretan y generan lenguaje humano) para analizar el código real que el alumno envió. El estudio utilizó conjuntos de datos reales de interacciones extraídos de más de 527 estudiantes de

cursos introductorios de programación (a menudo provenientes de plataformas de evaluación automática o Online Judges). Los datos consisten en 38,000 de secuencias de intentos donde los estudiantes envían sus códigos fuente (con errores de compilación, errores lógicos o aciertos) para resolver diferentes problemas que requieren habilidades específicas (como condicionales, bucles, etc.). Las técnicas principales para procesar y analizar los datos de los estudiantes se enfocaron en el procesamiento del código fuente real: generando AST, además de utilizar incrustaciones de código (Code Embeddings): para esto utilizaron técnicas tomadas del Procesamiento de Lenguaje Natural (NLP) para convertir fragmentos de código en vectores matemáticos densos, lo que permite a la computadora medir matemáticamente la calidad y estructura del envío del alumno y Análisis Secuencial: procesaron estos datos como series de tiempo (trayectorias de aprendizaje), emparejando cada intento de programación con los conceptos o habilidades que exigía el ejercicio. Los autores desarrollaron un modelo avanzado de Deep Learning (Aprendizaje Profundo) que llamaron PKT-ATTN. Sus componentes incluyen: Redes Neuronales para Código: Utilizaron modelos como Code2Vec, ASTNN y Redes Neuronales de Grafos (GGNN) para que la red entienda el código escrito por el estudiante. Mecanismos de Atención (Attention y Rastreo de Conocimiento Profundo (DKT): Todo esto se integra en un modelo de predicción secuencial (Deep Knowledge Tracing) para estimar el estado de conocimiento oculto del estudiante. Como resultado el modelo demostró que al analizar cómo se escribe el código (semántica y sintaxis), se puede interpretar con exactitud qué conceptos especificos de programación están fallando. Además, que superar las técnicas tradicionales como el DKT clásico en las métricas de precisión (Acurracy) y AUC (Área bajo la curva), demostrando ser 25% mejor para la predecir si el alumno será capaz de resolver el siguiente ejercicio basándose en la forma en la que codificó en los ejercicios pasados.

Adicionalmente [50] identifica dos desafíos principales en la enseñanza de la programación por medio de plataformas: 1) falta de etiquetas de conocimiento: muchas preguntas en sistemas OJ no tienen etiquetas que indiquen que concepto se está evaluando, 2) información estática insuficiente: los análisis basados en AST solo consideran la estructura estática del código, ignorando el comportamiento dinámico. Por lo cual, proponen el modelo DGGANN (Dynamics-Aware Gated Attention Neural Network), que se caracteriza por:

A) crea una variante del AST que integra coeficientes de frecuencia de llamada. La cual ejecuta el código con casos de prueba para ver que líneas se ejecutan más veces, lo que permite que el modelo de más peso a las partes más utilizadas del código y menos a las que no.

B) Análisis dinámico: al combinar la estructura del código con datos de ejecución real, el modelo logra entender mejor la funcionalidad del programa.

Esta propuesta se aplicó a dos tareas experimentales utilizando datos de CodeForce y libre, como resultados se obtuvieron que el modelo supero a modelos existentes en identificar correctamente a qué pregunta correspondía un código enviado por un estudiante, además de demostrar ser más preciso para predecir si un estudiante podrá resolver una futura pregunta basándose en su historial de respuestas y la calidad de sus códigos previos con ayuda de Knowledge Tracing.

Transicionado hacia los modelos aplicados a la programación visual y el trazado de procesos dentro de un mismo ejercicio, el artículo [51] y [52] presenta el modelo de rastreo de conocimiento de procesos (PKT). Su objetivo es predecir el estado de conocimiento de un alumno utilizando exclusivamente los datos de sus intentos intermedios dentro de un solo ejercicio (HOC 18 y HOC 4). Se analizaron las trayectorias de más de 164,000 estudiantes de la plataforma Hour of Code. Utilizando la distancia de edición (Tree Edit Distance) entre el árbol de sintaxis del estudiante y la solución óptima, apoyados en regresión logística y modelos ocultos de Markov, los resultados revelaron que PKT superó en un 15% a cuatro modelos del estado del arte (como BKT (Bayesian Knowledge Tracing) y DKT (Deep knowledge Tracing) alcanzando un AUC de 0.8X. Esto demostró que es posible rastrear el dominio de conocimiento evaluando solo el proceso dentro de una sola práctica, sin necesidad de un historial largo.

En cuanto a las plataformas específicas para el desarrollo del pensamiento computacional, el estudio de [21] propone un enfoque basado en datos para crear itinerarios de aprendizaje. Analizaron 250 proyectos de Scratch (50 por cada una de las 5 categorías evaluadas) desarrollados por más de 100 participantes Utilizando técnicas de agrupación secuencial, obtuvieron como resultado una ruta de aprendizaje conformada por tres clústeres principales: 1) animación, arte y música; 2) historias y 3) juegos, con un nivel de fiabilidad del [Insertar métrica de resultado].

Por otro lado, la investigación [8] presenta un sistema de recomendación de rutas de aprendizaje basadas en redes neuronales. En este estudio participaron 1,000 usuarios, de los cuales se extrajo una muestra de 7 para pruebas detalladas. El sistema procesó los resultados de los usuarios y

obtuvieron como resultado una precisión del 80% en la recomendación de problemas de programación adecuados para el nivel de habilidad actual del estudiante.

De manera similar, el estudio de [10] presenta un estudio sobre el desarrollo del pensamiento computacional en programación en estudiantes principiantes. Utilizó datos de 1,004 estudiantes entre 8 y 14 años (3ro. de primaria hasta 2do. De secundaria). Mediante análisis estadisticos obtuvieron como resultado que la edad y el sexo han demostrado influir en el aprendizaje del CT.

Finalmente, el trabajo de [11] presenta un enfoque híbrido para evaluar el proceso de resolución de problemas a gran escala utilizando la herramienta Blockly Games (sección laberinto). Los usuarios resolvieron distintos ejercicios, almacenando sus iteraciones y respuestas, las cuales fueron convertidas en AST. Con una población de 50 usuarios implementaron algoritmos y modelos de aprendizaje y estadística descriptiva que arrojaron como resultado que el tiempo que se tarda en resolver un nivel (63% de precisión) y el tiempo entre ejecuciones (55% de precisión) pueden predecir el nivel de habilidad del programador. Es importante destacar que, a diferencia del trabajo de [11] —el cual se limita a clasificar o predecir el nivel de habilidad del estudiante—, nuestra propuesta da un paso más allá al generar una recomendación directa y personalizada del siguiente ejercicio específico que el alumno debe resolver.

En conclusión, si bien las investigaciones analizadas demuestran avances significativos en el trazado de conocimiento y la predicción de rendimiento, la gran mayoría se centra en predicciones generales (aprobar/reprobar) o en estudiantes de nivel universitario, siendo escasas aquellas que propongan rutas dinámicas con ejercicios específicos para que los niños desarrollen su pensamiento computacional. Es por ello que la presente investigación se enfoca en el análisis detallado de las soluciones intermedias en entornos de programación infantil; con el objetivo de que dicho análisis dé como resultado la recomendación exacta del siguiente ejercicio a resolver, garantizando así la generación de rutas de aprendizaje verdaderamente personalizadas para cada usuario.

Capítulo 4

4. Metodología de Desarrollo de Proyecto

En el presente capitulo se muestra la metodología (ver ilustración 13) utilizada en nuestra investigación, la cual consta de 13 fases: Entendimiento del problema, Investigación del estado del arte, diseño de la actividad, identificación de herramientas a utilizar, adaptación del juego a la librería blockly, adaptación de la captura de interacciones, generación de los AST, pruebas, generación de las soluciones óptimas, diseño experimental, recolección de datos, definición de características y análisis experimental.

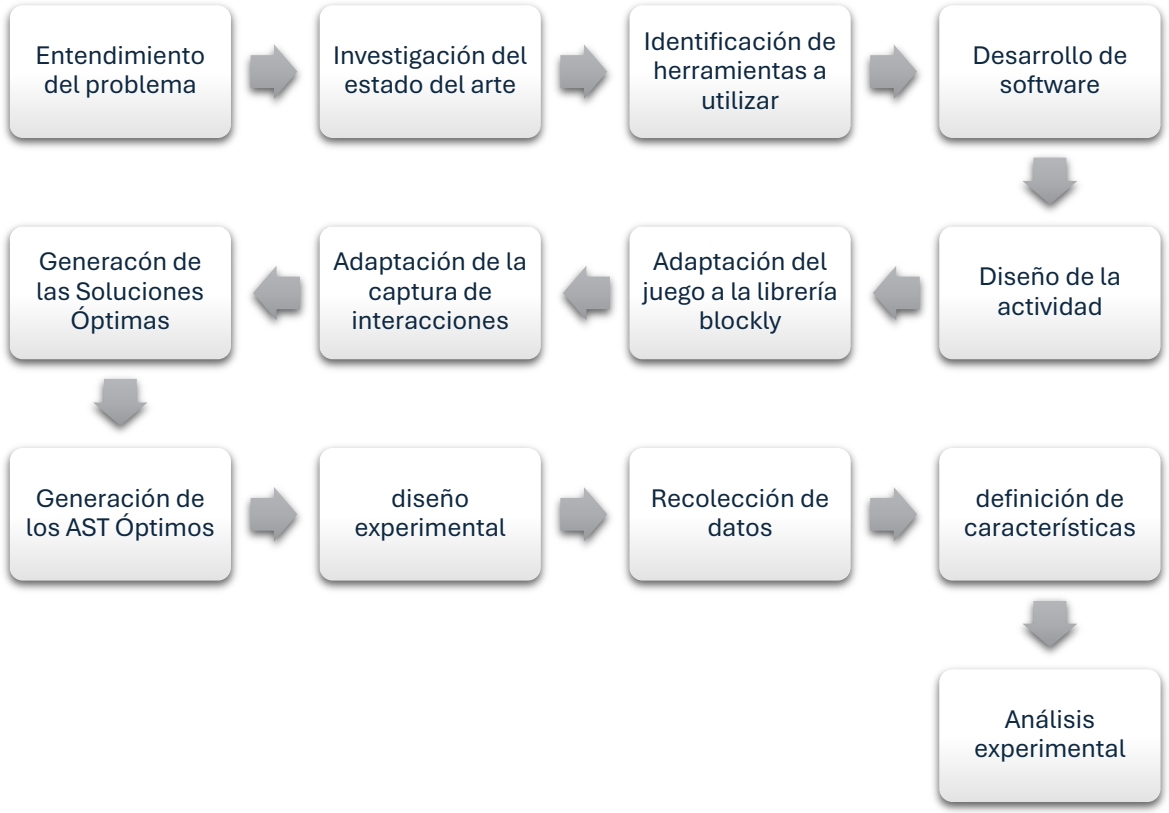


Ilustración 13. Metodología utilizada.

4.1 Entendimiento del problema

El objetivo general de nuestro proyecto es generar rutas de aprendizaje de programación personalizadas para niños. Para lograrlo, se requiere contar con datos de las interacciones de los usuarios, así como con representaciones del código que permitan analizar si su respuesta es correcta o no.

Al iniciar el proyecto se identificó un problema fundamental: la falta de una fuente adecuada de datos sobre el desempeño de los alumnos en actividades de programación con bloques. Las plataformas educativas que existen en el mercado ofrecen ejercicios que se ajustan a los niños; sin embargo, no permiten el acceso a su base de datos ni capturar las interacciones de los usuarios. Se intentó obtener información mediante scripts y grabaciones, pero no fue posible recolectar datos para realizar un análisis.

También se exploró la posibilidad de utilizar ejercicios realizados por nosotros, tales como cuestionarios y la predicción del funcionamiento de bloques. Sin embargo, estas alternativas no generaban información adecuada ni permitían generar el razonamiento del estudiante, por lo que no facilitaban la creación de rutas de aprendizaje personalizadas.

Ante estas limitaciones, se identificó una necesidad clave: desarrollar una herramienta propia que permitiera registrar, de forma completa las acciones del alumno mientras resuelven ejercicios de programación.

A partir de este entendimiento del problema surge la fase dos de nuestra metodología: investigar el estado del arte, la cual explicamos a continuación

4.2 Investigación del estado del arte

Para fundamentar nuestra investigación, llevamos a cabo la investigación de nuestro estado del arte, el cual consto de lo siguiente:

- Investigar trabajos similares al que se propone en este trabajo. Como resultado se encontró que el trabajo de [43] es similar al de esta investigación, en el cual utiliza AST para analizar

la forma en la cual se resuelven los ejercicios de programación, como resultado predicen si el ejercicio siguiente el usuario lo resolverá de manera correcta o incorrecta.

- Investigar qué tipo de ejercicios de programación realizar a los niños. Se encontró que los ejercicios idóneos para niños no son a base de texto, si no a base de bloques de código, en los cuales los niños deberán de llegar de un punto “A” a un punto “B” utilizando la menor cantidad de bloques posibles para lograrlo.
- Investigar el tipo de herramientas que se usan para el aprendizaje de la programación. Se encontraron que las herramientas más utilizadas para la enseñanza de la programación son: Scratch, CODE.org y Blockly Games.
- Investigar acerca de las bases datos disponibles para la realización de experimentos y análisis. Como resultado no se obtuvo ninguna base de datos disponible para realizar el análisis de lo mismo. Solo en la investigación de [3] menciona y da una liga de acceso de los conjuntos de datos que utilizaron, pero al momento de acceder a él la página no estaba disponible; se intentó comunicarse con los autores vía correo electrónico para preguntar sobre la disposición de la base de datos, pero no hubo respuesta alguna.
- Investigar los tipos de análisis generados en el aprendizaje de la programación. Se encontró que los análisis generados fueron: 1) análisis de clúster en el cual categorizaban los códigos de los usuarios en las diferentes secciones con las que cuenta Scratch. 2) Redes neuronales para la predicción del resultado del siguiente ejercicio a responder (correcto, incorrecto). 3) Análisis estadístico que dice el nivel del usuario de acuerdo con el tiempo y la edad que tiene al momento de realizar los ejercicios de programación.

Con esta investigación se logró identificar las herramientas y tipos de ejercicios necesarios que nos ayudarían a generar las rutas de aprendizaje. En la siguiente sección mencionamos dichas herramientas.

4.3 Identificación de herramientas a utilizar

En esta tercera fase, se identificaron las herramientas que podrían servirnos para la recolección de datos, las cuales se muestran en la tabla 7.

Tabla 7. Plataformas Web para la enseñanza de la programación visual.

Herramienta	Definición	Método	Dimensiones del pensamiento computacional
Scratch	Es un lenguaje de programación gratuito en línea donde se utilizan bloques de código.	Crear tus propias historias interactivas, juegos y animaciones.	Dependiendo de los proyectos a realizar puede estar presente de 1 a las 7 (pensamiento lógico, representación de datos, interactividad con el usuario, control de flujo, abstracción y descomposición de problemas, paralelismo y sincronización), dimensiones del pensamiento computacional.
Blockly games	Juegos para la enseñanza de la programación.	Lecciones de programación basadas en bloques para principiantes.	Dependiendo de los proyectos a realizar puede estar presente de 1 a las 7 (pensamiento lógico, representación de datos, interactividad con el usuario, control de flujo, abstracción y descomposición de problemas, paralelismo y sincronización) dimensiones del pensamiento computacional.
CODE.org	Es una organización sin fines de lucro dedicada a garantizar que todos los estudiantes de cada escuela tengan la oportunidad de aprender informática.	Utiliza un sistema de arrastrar y soltar bloques visuales para construir programas. Las lecciones se presentan como desafíos y juegos donde los	Dependiendo de los proyectos a realizar puede estar presente de 1 a las 7 (pensamiento lógico, representación de datos, interactividad con el usuario, control de flujo, abstracción y descomposición de problemas, paralelismo y sincronización)

		estudiantes deben resolver problemas para avanzar de nivel.	dimensiones del pensamiento computacional.

Para las tres herramientas se intentaron recolectar las interacciones y los datos de los usuarios por medio de grabaciones de pantalla y la creación de Scripts, pero no se pudieron llevar por motivos de seguridad de las herramientas y por los datos que se podrían recabar no nos servían para el propósito de nuestra investigación.

Por su parte [52] proporciona una base de datos generada con la herramienta CODE.org la cual se intentó descargar, pero al momento de la descarga ya no estaba disponible. Por lo que se optó por contactar a [52], pero no hubo respuesta alguna.

Dadas las circunstancias anteriores, se optó por diseñar una herramienta propia, la cual no solo ayudaría con la recolección de datos y las interacciones de los usuarios, sino también guardaría el código generado por los mismos.

4.4 Desarrollo de software

El software desarrollado tiene como objetivo principal recolectar las interacciones de los usuarios mientras resuelven ejercicios de programación en la herramienta desarrollada, con el fin de analizar su proceso de aprendizaje. Se enfoca en la captura de eventos (clics, bloques creados, tiempos de ejecución, bloques eliminados, almacenamiento de código etc.) dentro de entornos de programación por bloques basados en la librería Blockly.

Para el desarrollo de la herramienta se utilizaron los lenguajes de programación PHP y JavaScript, además de HTML y CSS. Para la generación de los bloques entrelazados se utilizó la librería Blockly, el lenguaje de marcado XML y el formato de datos JSON.

En la ilustración 14 se muestra la interfaz principal de la herramienta, la cual se compone de los elementos siguientes:

- **Cinta de ejercicios (1):** lugar donde se listan los ejercicios a resolver, además de encontrar los botones de inicio y categorías de bloques.
- **Lienzo del problema (2):** Lugar donde se muestra el ejercicio que debe de resolver el usuario. Además, incluye la animación de la solución del problema de acuerdo con el código ejecutado por el usuario.
- **Caja de Herramientas (3):** Lugar donde se encuentran los bloques para la solución de los ejercicios de programación.
- **Lienzo de trabajo (4):** Lugar donde el usuario coloca los bloques que considera necesarios para la solución de los ejercicios.
- **Caja de botones (5):** Lugar donde se muestra el nombre del usuario que ingreso, así como los botones de:
    - **Ejecutar:** La función de este botón es ejecutar el código creado por el usuario, así como de guardarlo en archivos XML en una base de datos.
    - **Adelantar:** La función de este botón es recorrer el código generado bloque a bloque cada que se da un clic en él.
    - **Retroceder:** La función de este botón es retrasar el código generado bloque a bloque cada que se da un clic en él.
    - **Salir:** La función de este botón es salir de la sección de ejercicios y regresar a la página de inicio de sesión.

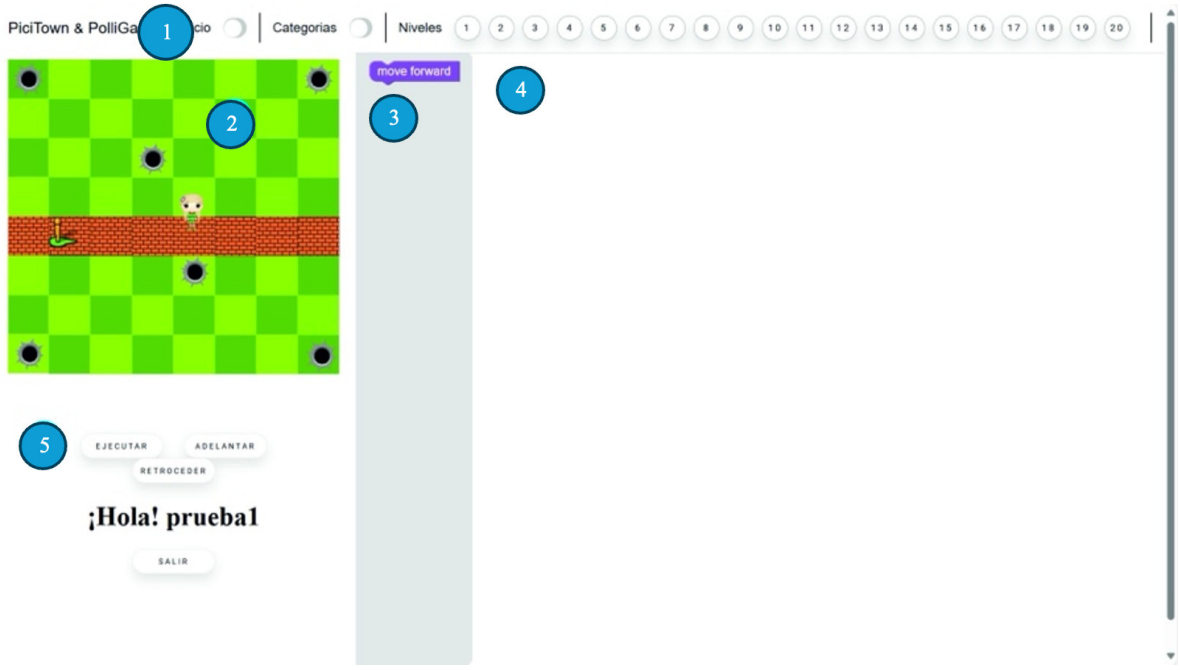


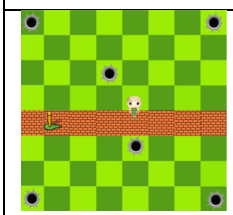
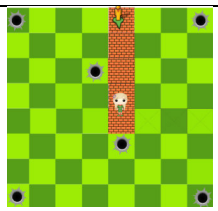
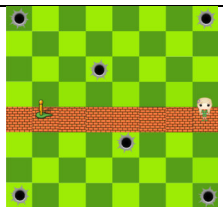
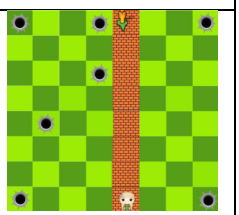
Ilustración 14. Interfaz principal de la herramienta de programación.

4.5 Diseño de la actividad

Para diseñar las actividades de la herramienta, se optó por realizar ejercicios de tipo laberinto semejantes a los de CODE.org y Blockly games.Para ello se tomaron en cuenta las 7 dimensiones del pensamiento computacional (pensamiento lógico, representación de datos, interactividad del usuario, control de flujo, abstracción y descomposición de problemas y por último sincronización), así como el nivel de dificultad de cada uno de ellos, iniciando con nivel básico e ir subiendo de nivel conforme se van desarrollando los ejercicios.

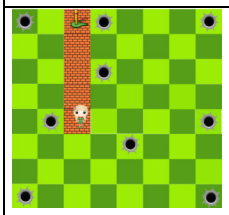
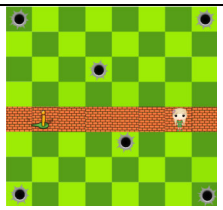
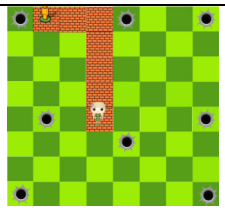
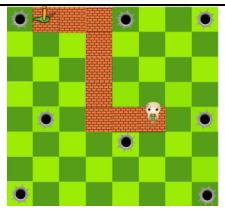
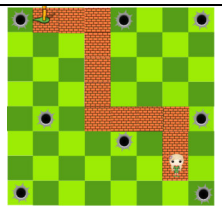
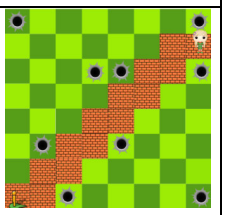
Los ejercicios del 1 al 4 (ver tabla 8) tienen como objetivo abordar las dimensiones del pensamiento computacional: pensamiento lógico. Ejercicios en los que se usen sólo bloques de movimiento.

Tabla 8. Primeros 4 ejercicios de la herramienta de programación a resolver.

Ejercicio 1	Ejercicio 2	Ejercicio 3	Ejercicio 4
			

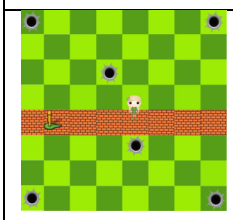
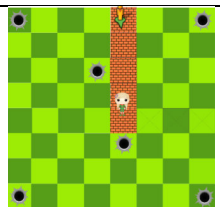
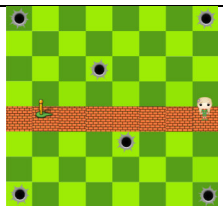
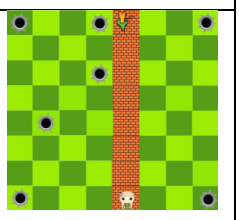
Los ejercicios del 5 al 10 (ver tabla 9) tienen como objetivo abordar las dimensiones del pensamiento computacional: pensamiento lógico, abstracción y descomposición de problemas.

Tabla 9. Ejercicios del 5 al 10 de la herramienta de programación con bloques de giro.

Ejercicio 5	Ejercicio 6	Ejercicio 7	Ejercicio 8	Ejercicio 9	Ejercicio 10
					

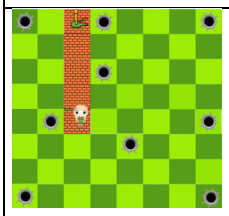
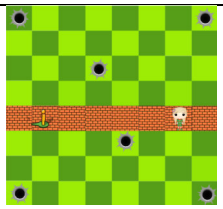
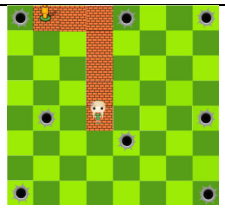
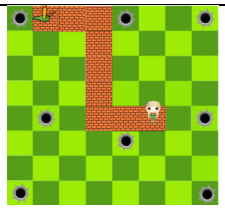
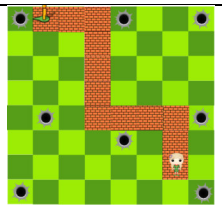
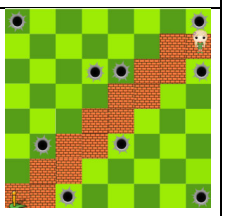
Para los ejercicios del 11 al 14 (ver tabla 10) se tiene como objetivo abordar las dimensiones del pensamiento computacional: pensamiento lógico, control de flujo y abstracción y descomposición de problemas.

Tabla 10. Ejercicios del 11 al 14 de la herramienta de programación con bloques de repetición.

Ejercicio 11	Ejercicio 12	Ejercicio 13	Ejercicio 14
			

Para los ejercicios del 15 al 20 (ver tabla 11) se tiene como objetivo abordar las dimensiones del pensamiento computacional: pensamiento lógico, control de flujo, abstracción y descomposición de problemas y sincronización.

Tabla 11. Ejercicios del 15 al 20 de la herramienta de programación con bloques de giro y repetición.

Ejercicio 15	Ejercicio 16	Ejercicio 17	Ejercicio 18	Ejercicio 19	Ejercicio 20
					

Una vez diseñado los ejercicios para la herramienta de programación, se continuo con la etapa de programación para la captura de código. Dicha fase se muestra a continuación.

4.6 Adaptación del juego a la librería Blockly

En la sección 4.4 se describió el desarrollo de la herramienta, la cual hasta este punto cumple con la función ejecutar los códigos generados por el usuario y decir si los solucionó de manera correcta o no, pero aún no cumple con el objetivo de guardar los códigos generados por los usuarios, así como sus interacciones. Para lograr lo anterior se utilizó la librería Blockly, la cual no solo permite la generación de bloques de programación si no también obtener las interacciones y sus características cada vez que se hace una acción con ellos.

En la tabla 12, se muestra la línea de código (1) que permite mostrar el espacio de trabajo en la herramienta de programación.

Tabla 12. Código para la generación del espacio de trabajo.

<script src="libs/blockly/blockly_compressed.js></script>	(1)

Por su parte la línea de código 1 de la tabla 13 permite crear la caja de herramientas, que es el lugar donde se ubicarán los bloques de código para resolver los ejercicios de programación.

Tabla 13. Código para la generación del espacio de trabajo.

<script src="libs/blockly/blocks_compressed.js"></script>	*(1)*

Mientras que la tabla 14 muestra el código para la creación del bloque de movimiento en formato JSON. La línea (2) muestra el tipo de bloque que se desea crear, en este ejemplo un bloque de

movimiento, la línea (4) permite agregar el nombre que aparecerá impreso en bloque creado, las líneas (5) y (6) muestran si tienen un bloque conectado anterior o posteriormente y la línea (7) da el color del bloque creado.

Tabla 14. Código para la generación del espacio de trabajo.

```
'move': {                                                    (1)
        method: 'MOVE',                                      (2)
        json: {                                              (3)
          'message0': 'move forward',                        (4)
          'previousStatement': null,                         (5)
          'nextStatement': null,                             (6)
          'colour': 285                                      (7)
        }
      }
   }
})
```

La ilustración 15 muestra el resultado generado del código anterior para la creación del bloque de movimiento.

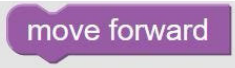


Ilustración 15. Bloque de movimiento.

En la tabla 15 se muestra las líneas de código para la generación del espacio de trabajo. La línea (2) crea un div en la cual será colocado el espacio de trabajo, la línea (3) inserta el área de trabajo en la cual se colocarán los bloques necesarios para cada ejercicio. En la línea (4) se inserta un bloque de movimiento el cual aparecerá en el área de trabajo de la herramienta.

Tabla 15. Código para la generación del espacio de trabajo.

```
<div id="blockly-editor">                                    (1)
        <div id="blockly-div"></div>                         (2)
        <xml id="toolbox" style="display: none">             (3)
          <block type="move"></block>                        (4)
          </block>                                           (5)
```

```
</xml>                                                    (6)
```

Para la colocación de los ejercicios de programación en la herramienta se necesitan de las siguientes líneas de código:

La línea de código 1 de la tabla 16 genera el espacio donde se colocarán los diseños de los ejercicios de programación. Para esto se debe de asignar un id a cada ejercicio, en la línea (1) se tiene un ID 1 el cual colocará en el espacio de ejercicios el primer ejercicio de programación, también se puede asignar el ancho y largo (widht, height) así como la posición en donde se quiere establecer el ejercicio.

Tabla 16. Creación del espacio para la colocación de los ejercicios de programación.

```
<svg xmlns="http://www.w3.org/2000/svg" style="display: none" version="1.1" id="1"
width="400px" height="400px" viewBox="0 0 400 400">                    (1)
```

La línea 1 de la tabla 17 permite insertar la imagen del ejercicio de programación en formato png. Se pueden colocar las medidas que deseen siempre y cuando, no sobre pase las medidas del espacio de trabajo creado en la tabla 16 línea (1).

Tabla 17. Colocación de la imagen de laberinto.

```
<image xlink:href="images/nivel_1.png" class="maze" x="0" y="-1" width="400"
height="399"></image>                                                  (1)
```

La línea 1 de la tabla 18 permite colocar la imagen objetivo, que en este caso es una elfa a la cual deberá llegar el pegman para completar el nivel. Para esto se define el id de cada una de ellas, en el ejemplo de la tabla 18 el id del primer objetivo es “finish1”, además se debe de colocar la ruta donde se encuentra la imagen (en su computadora), así como su tamaño y posición.

Tabla 18. Colocación del personaje objetivo.

```
<image id="finish1" xlink:href="images/Pici_Elfa.png" height="45" width="35" x="205" y="170"></image>  (1)
```

El bloque de código (1) mostrado en la tabla 19 genera un máscara, la cual tiene como función no distorsionar la imagen del pegman y solo mostrar la porción de la imagen que cae precisamente dentro del rectángulo de la máscara.

Tabla 19. Generación de mascara para la no distorsión del pegman.

```
<clipPath id="pegmanClipPath1">
  <rect id="clipRect1" width="49" height="52" x="101" y="191"></rect>
</clipPath>  (1)
```

Mientras que el bloque de código 1 de la tabla 20 permite colocar la imagen del pegman. Para esto se le asigna un ID, en el ejemplo de la tabla 20 el ID asignado es “pegman1”, el cual permite su identificación, su posición y sus medidas, las cuales son adaptadas a las medidas de la máscara de tabla 19 línea (1).

Tabla 20. Colocación de la imagen del pegman.

```
<image id="pegman1" xlink:href="images/pegman.png" height="80" width="1029" clip-path="url(#pegmanClipPath1)" x="65" y="180" transform="rotate(0, 0, 0)"></image>  (1)
```

En la ilustración 16, se muestra el resultado final del bloque de código de la tabla 20. Donde se puede observar el diseño del primer ejercicio de programación. El cual está formado por:

- el **pegman,** el cual se ubica ubica en el segundo cuadro (contando de izquierda a derecha) de ladrillos,
- y la **elfa**, que es el objetivo al que debe de llegar el pegman ubicada en el 5to. cuadro (contando de izquierda a derecha).

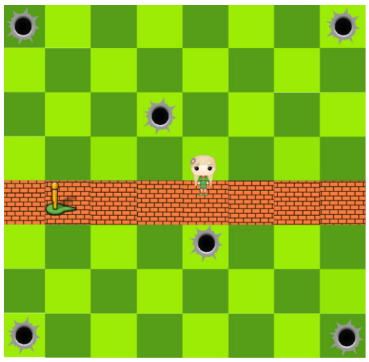


Ilustración 16. Ejercicio 1 de la herramienta de programación.

Una vez colocados todos los ejercicios se debe de programar el funcionamiento de los mismos. En el bloque de código (1) de la tabla 21 se manda llamar con su ID al ejercicio que se le aplicará su funcionamiento; en este ejemplo se manda llamar al ejercicio 1.

Tabla 21. Colocación de la imagen del pegman.

```
const levels = [
  // nivel 1
  {
    id: 1,                                                    (1)
```

El bloque de código (1) de la tabla 22 permite colocar en la caja de herramientas solo los bloques que aparecerán en el ejercicio. Para los ejercicios del 1 al 4 solo se muestran bloques de movimiento, para los ejercicios del 5 al 10 los bloques de movimiento y giros y para los ejercicios del 11 al 20, los bloques de movimiento, giros y repetición.

Tabla 22. Colocación de los bloques de código en la caja de herramienta.

```
    // bloques mostrados en la caja de herramienta
    blocks: [
      'move'
    ],                                                        (1)
```

En el bloque (1) de la tabla 23 se colocan las coordenadas (x, y) del punto de partida del pegman (izquierda, derecha, arriba o abajo). Su inicio será en el punto de origen, por lo cual, sus coordenadas en el eje X y Y son 0, 0.

Tabla 23. Colocación de los bloques de código en la caja de herramienta.

```
    // datos del juego
  game: {
    // datos pegman
    pegman: {
      direction: 1, // ver 1: Izquierda, ver 2: Derecha
      x: 0,
      y: 0
    },                                                    (1)
```

Por su parte el bloque (1) de la tabla 24 se coloca la orientación hacia donde está mirando el pegman (derecha, izquierda, arriba o abajo). Las coordenadas x: -160, y:180 muestra la orientación en el plano x,y hacia donde esta dirigo el pegman. Para este ejercicio su orientación será mirando hacia la derecha (ver ilustración 14).

Tabla 24. Colocación de los bloques de código en la caja de herramienta.

```
start: [{
    //Direccion donde apunta Pegman
    pegman: {
      x: -160,
      y: 180
    },                                                    (1)
```

Mientras que para el bloque (1) de la tabla 25 se coloca la dirección donde se colocará el pegman en el ejercicio de programación. Para el ejercicio 1 el pegman tiene las coordenadas X: 38, Y: 190 en el plano cartesiano, lo que lo posición en el segundo cuadro de ladrillos (ver ilustración 14).

Tabla 25. Colocación física del pegman.

```
    //Lugar donde colocar a pegman en la recta
    rect: {
      x: 38,
      y: 190
    }
  },                                                    (1)
```

Para el bloque 1 de la tabla 26 se colocan las coordenadas objetivas a la cual deberá de llegar el pegman. Para el ejercicio 1 se utilizaron las coordenadas X:3, Y: 0, lo que se traduce a que el pegman deberá de caminar 3 bloques a la derecha y ningún bloque arriba o abajo para llegar a su objetivo.

Tabla 26. Colocación de las coordenadas objetivas.

```
    //Lugar donde colocar a pegman en la recta
      // marker data
      marker: {
        x: 3,
        y: 0
      }, (15)
(1)
```

El bloque de código (1) de la tabla 27 tiene como objetivo activar las celdas por las cuales el pegman se puede desplazar. En el ejercicio 1 se activan las coordenadas 0,0 hasta la coordenada 6,0 no se activan más celdas ya que no hay bloques de giro que permitan al pegman desplazarse a más lugares del ejercicio.

Tabla 27. Colocación de las coordenadas objetivas.

```
      // game path
      path: [
        // [x, y]
        [0, 0],
        [1, 0],
        [2, 0],
```

```
        [3, 0],
        [4,0]
        …
      ]
    }
  },                                                                    (1)
```

Con los bloques y líneas de código anteriores se tiene listo el funcionamiento del ejercicio 1.Para el resto de los ejercicios los pasos son similares, lo único que cambia son las coordenadas del punto de inicio, el punto objetivo y las coordenadas activas por las cuales se pueda mover el pegman.

4.7 Adaptación de la captura de interacciones

El siguiente paso de programación para la herramienta es adaptar la captura de las interacciones realizadas por el usuario y su código generado. Para esto se utilizaron las siguientes funciones.

4.7.1 Creación de la función de movimiento

El bloque de código 1 de la tabla 28 crea una función de movimiento, la cual permite detectar los movimientos que realiza el usuario en los bloques de programación.

Tabla 28. Función de movimiento de bloques de código.

```
function movimiento(event) {//Inicia función Movimiento
      var jsonString = "";
      …
      if (event.type == Blockly.Events.BLOCK_MOVE) {//Ciclo If para detectar los movimientos de
los bloques
        var idMovimiento = event.blockId; //variable que Guarda el ID del botón en movimiento
        …
}                                                                       (1)
```

Los datos que se guardan cuando se activa esta función son:

- **Id_Movimiento:** esta variable guarda el id del bloque que se movió.
- **Padre_Anterior:** Si el bloque que fue movido estaba conectado con otro, esta variable permite guardar el id del bloque al que estaba conectado.
- **Padre_Nuevo:** Si el bloque que fue movido lo conectan con otro bloque, esta variable permite guardar el id del bloque al que fue conectado.
- **Tipo:** Guarda el tipo de bloque que fue movido (ejemplo: bloque de movimiento, bloque de giro, bloque de repetición, etc.).

4.7.2 Creación de la función de cambio.

En el bloque (1) de la tabla 29 se crea la función cambiar, la cual permite identificar los cambios que se realizaron dentro de un bloque de programación. Por ejemplo, cambio de la dirección de un bloque de giro de derecha a izquierda o viceversa.

Tabla 29. Función de cambio de variables en bloques de código.

```
function cambiar(event) { // Inicio funcion cambiar
        var IdCambio = "";
        …
        if (event.type == Blockly.Events.BLOCK_CHANGE) { //Ciclo if para detectar los cambios de
los botones
            //Creacion de las variables para guardar los datos del evento
            var IdCambio = event.blockId; //Variable que Guarda el Id del botón de cambio
          …
}                                                                                    (1)
```

Los datos que guarda esta función son:

- **Id_Cambio:** Esta variable guarda el id del bloque que sufrió el cambio.
- **Elemento:** Variable que guarda el elemento del cambio.
- **Nombre:** variable que guarda el nombre del cambio.
- **Valor_Anterior:** Variable que guarda el valor que tenía el bloque antes de sufrir un cambio.
- **Valor_Nuevo:** Variable que guarda el valor nuevo del bloque.

4.7.3 Creación de la función de eliminar

El bloque de código (1) de la tabla 30 crea la función que permite saber que bloques han sido eliminados por los usuarios.

Tabla 30. Función de eliminar bloques de código.

```
function eliminar(event) { //Inicio funcion eliminar

var XmlEliminado = "";

if (event.type == Blockly.Events.BLOCK_DELETE) {//Ciclo if para detectar los botones eliminados

var XmlEliminado = event.oldXml; //Variable que guarda la informacion de los bloques Eliminados
…
} (1)
```

Los datos que guarda esta función son:

- **XML_Eliminado:** esta variable guarda un archivo XML con todos los datos del bloque eliminados. Los cuales son:
    - **ID del bloque eliminado**.
    - **Nombre del bloque eliminado**.
    - **Coordenadas de su posición final**
    - **Tipo de bloque**.

4.7.4 Creación de la función clics

Por su parte el bloque de código (1) de la tabla 31 crea la función que detecta los clics que hace el usuario en la caja de herramienta y en el espacio de trabajo.

Tabla 31. Función clics en bloques de código y área de trabajo.

```
    var idClick
    function clicks(event) { //Inicia funcion Clicks
      if (event.isUiEvent == Blockly.Events.CLICK) { // Ciclo if para detectar los clicks de botones
        //Se crean las variables para Guardar los datos de los clicks y seleccion
        var IdClick = event.blockId//Se crea la varia para almacenar el Id de los botones
        console.log('id_click: ',IdClick)
    …
}                                                                                    (1)
```

Los datos que guarda esta función son:

- **ID** de los bloques donde el usuario hace clic.

4.7.5 Creación de la función crear

El bloque de código (1) de la tabla 32 genera la función crear, la cual se activa cuando el usuario arrastra un bloque de la caja de herramientas al área de trabajo.

Tabla 32. Función crear nuevos bloques de código.

```
function creador(event) {//Inicia la funcion creacion

        if (event.type == Blockly.Events.BLOCK_CREATE) {// inicia ciclo if para detectar la creacion de los botones

            var info = event.xml; //Se crea la variable para Guardar la información del bloque

            const creacion = [info];
…
}                                                                                                    (1)
```

Los datos que guarda esta función son:

- **Evento_XML:** esta variable guarda un archivo XML con todos los datos del bloque creado, por ejemplo: ID_bloque, posición_X, posición_Y.
- **Tiempo_Creacion:** esta variable guarda el tiempo que tarda un usuario en crear un nuevo bloque de código.

4.7.6 Creación de la función guardar código

Por último, el bloque de código (1) de la tabla 33, crea una función que permite guardar el código generado por el usuario cada vez que lo ejecuta. Esta función permite ver el número de intentos que necesito el usuario para resolverlo, así como el código de todas sus soluciones antes de llegar solución correcta.

Tabla 33. Función guardar código generado por usuario.

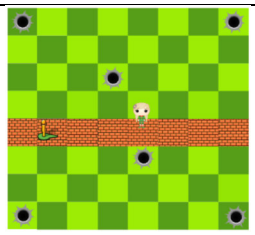
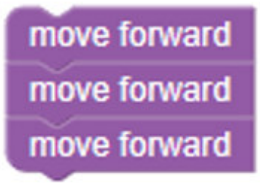
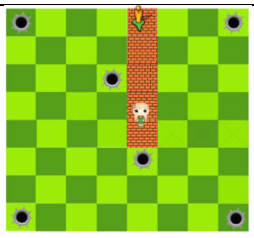
```
function guardarBloques() {
   var xml = Blockly.Xml.workspaceToDom(Blockly.getMainWorkspace());
   var xmlText = Blockly.Xml.domToText(xml);
   var file = new File([xmlText], "misBloques.xml", {type: "text/xml"});
   saveAs(file);
 }                                                                                                   (1)
```

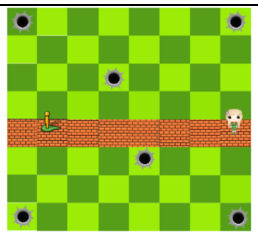
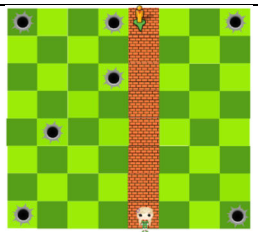
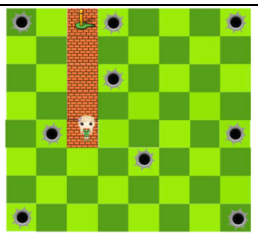
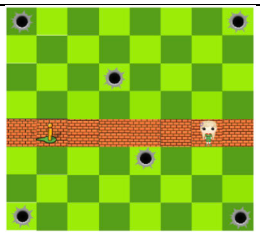
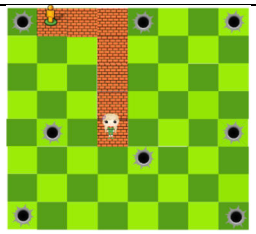
Con este último bloque de código se asegura la recolección de todas las interacciones necesarias y las capturas de los códigos generados por los usuarios, las cuales son base para este trabajo de investigación.

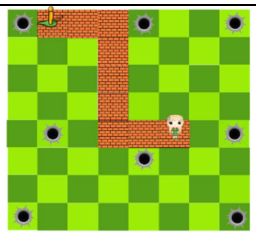
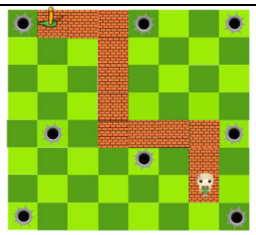
4.8 Generación de las Soluciones Óptimas

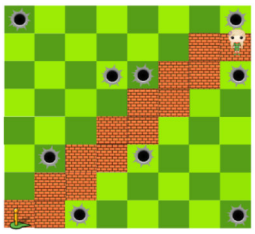
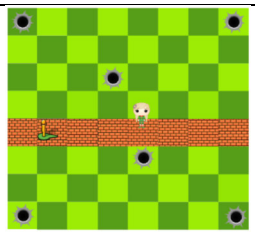
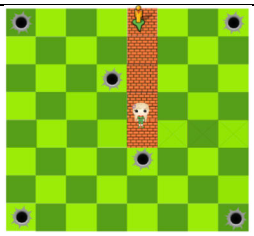
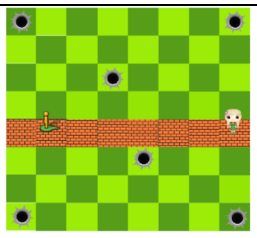
Ahora que la herramienta de trabajo ya está completa, se generarán las soluciones óptimas. Una solución óptima es aquella que resuelve un ejercicio de programación de manera correcta y con el menor número de bloques posibles. Para la generación de las soluciones óptimas de los ejercicios de programación se contestaron de manera correcta y con el menor número de bloques posibles cada uno de los ejercicios de programación. En la tabla 34 se muestran los 20 ejercicios y sus soluciones óptimas. Por ejemplo: para el ejercicio 9 se inicia dos bloques de giro a la derecha para que el pegman quede frente al camino y dos de movimiento para llegar a la esquina, despues un bloque de giro de a la derecha que le permite seguir con el camino de ladrillos y 4 bloques de movimiento antes de dar el a la izquierda, se continua con 3 bloques de movimiento seguido de un giros a la derecha y por último dos bloques de movimiento para llegar al objetivo.

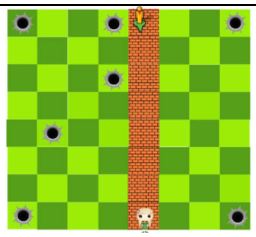
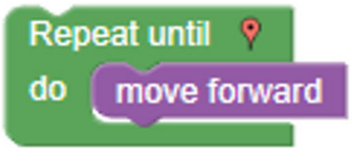
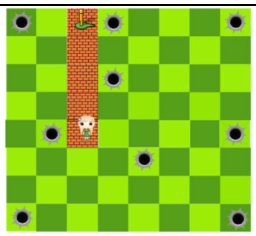
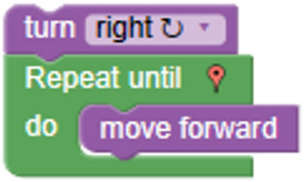
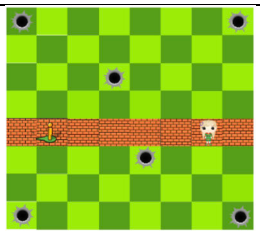
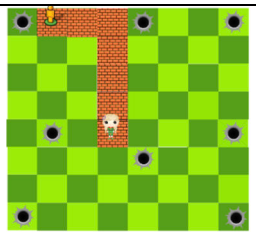
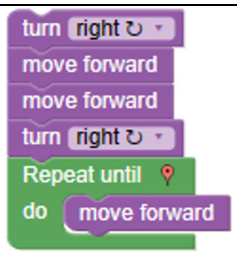
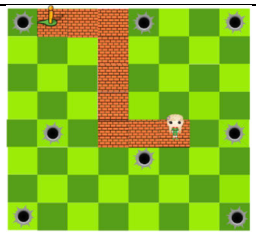
Tabla 34. Soluciones Óptimas de los 20 ejercicios de programación.

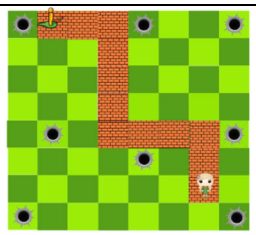
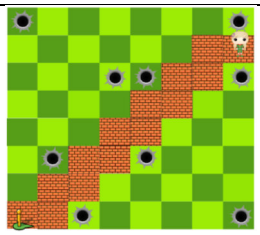
Número de ejercicio	Ejercicio	Solución Óptima
1		
2		

3		move forward move forward move forward move forward move forward move forward
4		move forward move forward move forward move forward move forward move forward move forward
5		turn right ↻ move forward move forward move forward move forward
6		turn right ↻ turn right ↻ move forward move forward move forward move forward move forward
7		turn right ↻ move forward move forward turn right ↻ move forward move forward move forward move forward

8		turn right ↻ turn right ↻ move forward move forward turn right ↻ move forward move forward move forward move forward turn left ↺ move forward move forward
9		turn right ↻ turn right ↻ move forward move forward turn right ↻ move forward move forward move forward move forward turn left ↺ move forward move forward move forward turn right ↻ move forward move forward

10		move forward turn left ↺ move forward turn right ↻ move forward turn left ↺ move forward turn right ↻ move forward turn left ↺ move forward turn right ↻ move forward turn left ↺ move forward turn right ↻ move forward turn left ↺ move forward turn right ↻ move forward turn left ↺ move forward turn right ↻ move forward
11		Repeat until do move forward
12		Repeat until do move forward
13		Repeat until do move forward

14		 Repeat until do move forward
15		 turn right Repeat until do move forward
16		 turn right turn right Repeat until do move forward
17		 turn right move forward move forward turn right Repeat until do move forward
18		 turn right turn right move forward move forward turn right move forward move forward move forward move forward turn left Repeat until do move forward

19		
20		

4.9 Generación de los AST

Con las soluciones óptimas ya generadas, se continuo con su conversión a AST óptimos. Para esto se siguió una pequeña metodología de dos fases, la cual también permite convertir las soluciones intermedias de los usuarios en AST:

- **Fase 1:** Cuando el usuario ejecutaba una solución de un problema la herramienta de programación convertía los bloques de código en formato XML para poder guardarlos en el set de datos.
- **Fase 2:** Una vez guardadas todas respuestas en el conjunto datos se procedió a convertirlas en AST. Para esto se desarrolló un programa en Python el cual convierte las respuestas de formato XML a AST de tipo JSON. En la ilustración 17 se muestra un AST de forma gráfica, mientras que en la tabla 35 se muestra un ejemplo de la transformación de código XML a AST.

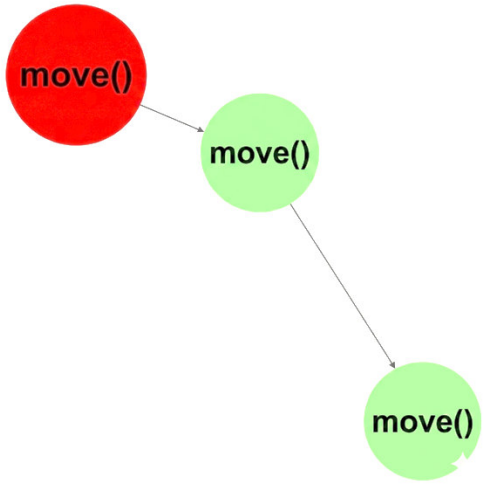


Ilustración 17. AST Generado.

Tabla 35. Conversión de XML a AST de tipo JSON.

XML	AST

<xml
xmlns="http://www.w3.org/1999/xhtml">
block type="move"
id="b}opX/7(b9=;Vz.];)A+" x="8"
y="41"><next><block type="move"
id="kf,le($R):m4Ra_{NqO+"><next><block
type="move"
id="tjoGRHxI,xGv@WFo~~R}"></block></n
ext></block></next></block></xml>
	{
	"type": "move",
	"id": "b}opX/7(b9=;Vz.];)A+",
	"x": "8",
	"y": "41",
	"fields": {},
	"next": {
	"type": "move",
	"id": "kf,le($R):m4Ra_{NqO+",
	"x": null,
	"y": null,
	"fields": {},
	"next": {
	"type": "move",
	"id": "tjoGRHxI,xGv@WFo~~R}",
	"x": null,
	"y": null,
	"fields": {}
	}

	} }

Por lo anterior cada AST generado por la herramienta fue almacenado en el conjunto de datos, esto con el objetivo de generar un análisis comparativo entre las soluciones óptimas y las soluciones de los usuarios. Para lo cual, se generó un programa en Python, el cual permitió comparar la altura de los árboles, el campo del bloque y el tipo de bloque. A continuación, se muestra un ejemplo tomando como base el ejercicio 5, en él, se comparará (ver ilustración 18) una solución intermedia (imagen del lado derecho) con una solución óptima (imagen del lado izquierdo). El nodo de color rojo indica el bloque con el cual se empezó a desarrollar el código, mientras que los nodos de color verde representan los nodos de movimiento.

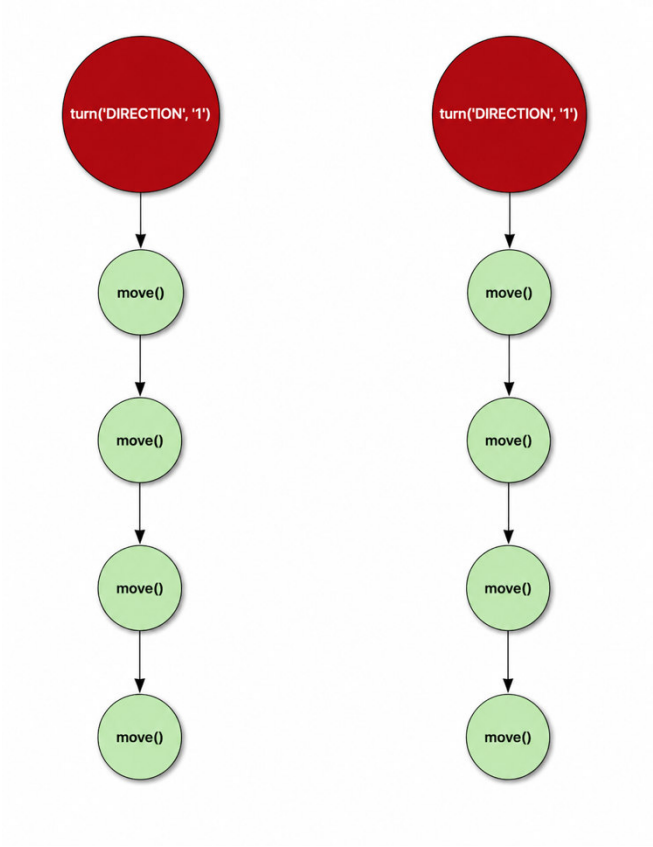


Ilustración 18. Solución óptima y Solución intermedia.

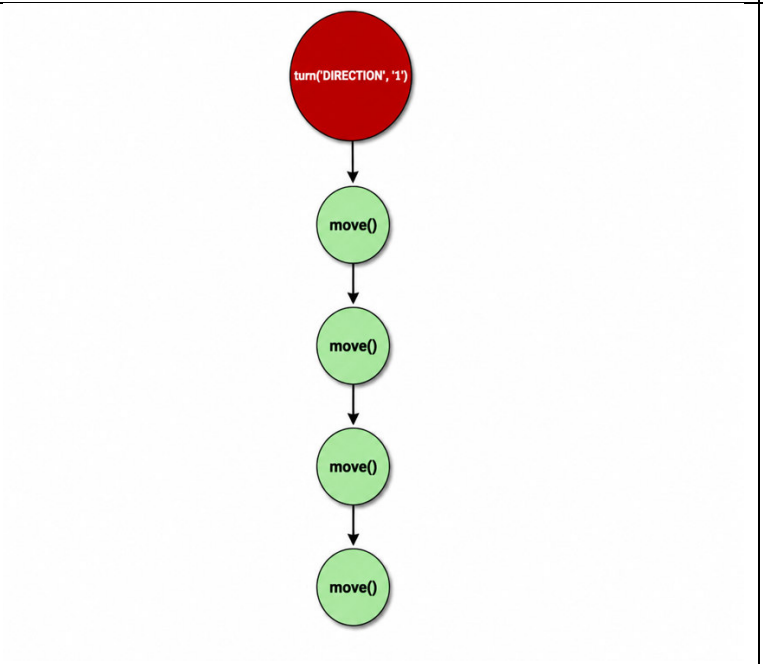
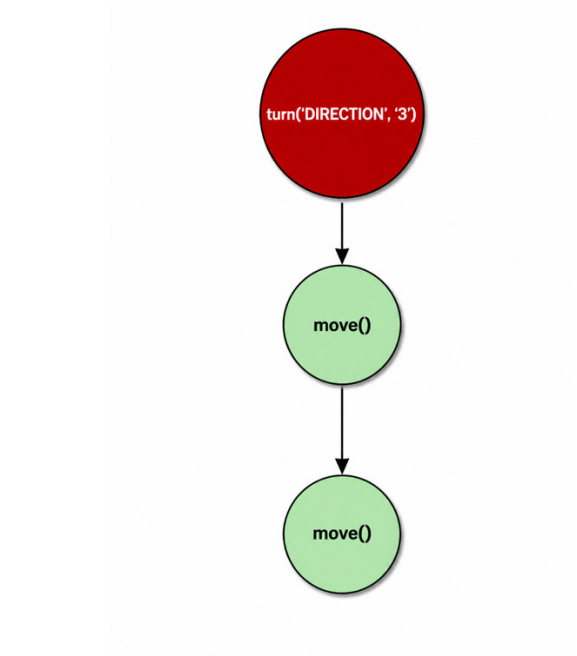
Por lo anterior se puede definir que existen tres tipos de soluciones:

- **Solución Óptima:** Llega a la solución correcta con el menor número de bloques posibles.

- **Solución Candidata o intermedia:** Cada solución generada por el usuario al momento de resolver los ejercicios de programación. Estas soluciones pueden llegar a hacer soluciones óptimas si se cumple con la condición de llegar al objetivo final con el menor número de bloques posibles.
- **Solución Final:** Es el último intento que realizó el usuario en cada ejercicio de programación. Una solución final también puede ser una solución óptima.

En la tabla 36 se muestra los AST gráficos de la solución óptima del ejercicio 5 y la solución generada de un usuario del mismo ejercicio, mientras que en la tabla 36 se muestran paso a paso la comparación que se hizo entre estos dos AST y sus resultados:

Tabla 36 Comparación entre una solución óptima y una solución de usuario.

Solución óptima	Solución dada por un usuario
	

- **Altura 0:** ambos son bloques de tipo turn (giro) por lo cual no hay diferencias en su tipo, sin embargo, en el campo de Fields la solución óptima tiene un valor de 1 (giro a la derecha) pero la solución generada por el usuario es 3 (giro a la izquierda), lo que indica que existe un error en el campo fields por parte del usuario,
- **Altura 1, 2:** los campo para ambos es nulo así que no hay diferencia, el tipo de bloques también es move (movimiento) para ambos así que no hay diferencia en este rubro.
- **Altura 3, 4:** para las alturas 3 y 4 la solución óptima cuenta con un bloque de movimiento para cada una de ellas sin embargo para la solución generada por el usuario hace falta bloques en esta altura por lo que la diferencia en type es True.

Tabla 37. Análisis de los AST.

Altura actual	Fields	Type	Diferencia en fields	Diferencia en type
0	{'DIRECTION': '1'}	turn	3	FALSO
1	{}	move	FALSO	FALSO

2	{}	move	FALSO	FALSO
3	{}	move	FALSO	TRUE
4	{}	move	FALSO	TRUE

Después de realizar este análisis se puede concluir que la solución intermedia del usuario es una solución errónea ya que desde el inicio tiene una dirección diferente a la solución óptima además de faltarle dos bloques de movimiento para completar el ejercicio.

4.10 Diseño experimental

4.10.1 Participantes

Para llevar a cabo nuestro experimento se tomaron como usuarios a 17 niños y 14 niñas de entre 8 y 10 años, los cuales no tenían experiencia previa con herramientas de programación ni con el uso de computadoras o laptops, cursaban el 4° grado de primaria en una escuela pública de Moroleón Gto. México, y eran originarios de este municipio, así como del municipio de Uriangato.

4.10.2 Sesión Experimental

Las sesiones para el experimento se hicieron de manera individual. Para esto, se instaló una zona de trabajo la cual consistía de una mesa, dos sillas y laptop, se realizarlo de esta forma debido a que la escuela no contaba con un centro de cómputo o computadoras para que los participantes realizarán al mismo tiempo la actividad. Cuando llegaba un participante se le explicaba en qué consistía la actividad y la manera en la cual funcionaba la plataforma, después de esto el participante comenzaba a realizar los ejercicios de programación, si tenían alguna duda o comentario nosotros estábamos ahí para ayudarlos en el desarrollo de sus actividades.

El tiempo promedio para la realizacion de la sesión completa es de 25 minutos, donde el ejercicio 10 es el ejericicio que toma más tiempo en realizarse con un promedio de 7.5 minutos al realizarlo, seguido del ejercicio 9 con 5 minutos. Los promedios de estos dos ejercicios se debe a la forma en la cual se resuelven, ya que necesitan multiples bloques de grio y movimiento, sus soluciones se

mencionan la sección 4.8 Generación de las soluciones óptimas y los resultados de las sesiones se encuentran en el capitulo 5 resultados.

4.10.3 Permisos

Debido a que los participantes de nuestro experimento eran niños, se generó un documento de aceptación (este documento se muestra en la sección de anexos) donde se explicaba todo lo relacionado con las sesiones de programación y la captura de los datos generados, (los cuales no eran datos sensibles). Además de notificarles y darse por enterados tanto a los participantes como a sus padres o tutores acerca de las sesiones y de que podían retirarse o retirar a sus niños de esta actividad.

4.11 Recolección de Datos

Como mencionamos anteriormente nuestra una herramienta que permite guardar las interacciones de los usuarios y sus respuestas. Los datos de las interacciones recolectadas se muestran en las siguientes tablas:

La tabla 38 guarda los datos: ID e info cuando se crea un nuevo bloque en el conjunto de datos generados por la herramienta.

Tabla 38.Acciones registradas al crear un bloque.

Acción del bloque crear	Significado
ID	Identificador del bloque creado
info	Contiene información acerca del bloque generado

Mientras que la tabla 39 guarda los datos: ID y Borrados al momento en el que se elimina un bloque.

Tabla 39. Acciones registradas al eliminar un bloque.

Acción del bloque eliminar	Significado
Id	Identificador del bloque eliminado
Borrados	Contiene información del bloque eliminado

Por su parte, la tabla 40 guarda los datos: id, idCambio, Elemento y Nombre cuando se realiza un cambio en el (derecha a izquierda o viceversa) bloque de giro.

Tabla 40. Acciones registradas en cambios en bloques.

Acción de cambio para los bloques tipo ciclo	Significado
Id	Identificador de cambio en un bloque
idCambio	ID del bloque que sufrió el cambio
Elemento	Campo que fue cambiado dentro del bloque
Nombre	Nombre del cambio
ValorAnterior	Valor que tenía anteriormente el bloque
ValorNuevo	Valor nuevo después del cambio

Para entender de mejor manera los datos recolectados por la acción de cambio, se colocó la tabla 41 donde se muestran los datos guardados en el momento que se activa esta acción.

Tabla 41. Datos guardados derivado de la acción cambio.

id	idCambio	Elemento	nombre	ValorAnterior	ValorNuevo
10	"L{J-S5(2So4el;F)o+0b"	"field"	"DIRECTION"	"3"	"1"

- El campo ***id*** contiene un valor de 10, lo que significa que este es el identificador del usuario que generó el cambio.
- El campo ***idCambio*** contiene el identificador del bloque que sufrió el cambio.
- El campo ***elemento*** muestra el elemento que sufrió el cambio, en este caso fue el campo “field” (campo en español).
- El campo ***nombre*** muestra que la dirección de giro fue cambiada.
- El campo ***ValorAnterior*** tiene un valor de 3 el cual representa un giro a la izquierda.
- El campo **ValorNuevo** tiene un valor 1 el cual representa el cambio sufrido, en este caso se cambió de giro a la izquierda por giro a la derecha. Los únicos valores que puede tomar tanto el campo ValorAnterior, como el campo ValorNuevo solo es 1 o 3 (derecha o izquierda).

La tabla 42 guarda los datos: ID y xmlText cuando se ejecuta una solución de los usuarios.

Tabla 42.Propiedades registradas al momento de la ejecución del código.

Propiedad XML	Significado
ID	Identificador del código XML generado
xmlText	Códigos generados por los usuarios en formato XML

La tabla 43 guarda los datos: ID, idMovimiento, Padreanterior, PadreNuevo y tipo al momento que el usuario mueve un bloque.

Tabla 43. Acciones registradas en la acción de movimiento de bloques.

Acción de Movimiento	Significado
Id	Identificador del movimiento generado
idMovimiento	ID del bloque que sufrió un movimiento
Padreanterior	Muestra el ID del bloque en el que estaba conectado. Si no lo hay no lo muestra
PadreNuevo	Muestra el ID del bloque al que fue conectado el bloque en movimiento. Si no lo hay no lo muestra
Tipo	Muestra el nombre del bloque sufrió el movimiento

Para explicar de mejor manera la recolección de los datos de la acción de movimiento se generó la tabla 44.

Tabla 44. Ejemplo de los datos guardados en la acción de movimiento de bloque.

id	idMovimiento	padreanterior	padrenuevo	tipo
20	"YTOqJn:u.FYmA]Dnp3_Q"	")G*wqy5mrsBvzdKa1+md"	")G*wqy5mrsBvzdKa1+md"	"move"

- El campo ***id*** muestra el identificador del movimiento realizado.
- El campo ***idMovimiento*** muestra el identificador del bloque que se movió.
- El campo ***padreanterior*** muestra el identificador del bloque al que estaba conectado el bloque que se movió.

- El campo ***padrenuevo*** muestra el identificador del bloque al que fue conectado el bloque que se movió.
- El campo ***tipo*** muestra el tipo de bloque que se movió, en este caso era un bloque de movimiento.

La tabla 45 guarda los datos ID, datoRe y datosJs al momento que inicia la ejecución de la solución del usuario.

Tabla 45. Acciones registradas en la propiedad ruta al momento de ejecutar el código.

Propiedad Ruta	Significado
ID	Identificador de la ruta generada
datoRe	Muestra la simulación del código ejecutado en formato de texto
datosJs	Muestra las coordenadas de la ejecución del código y la dirección del mismo

Para entender mejor los datos almacenados en la tabla 46 se colocó un ejemplo en la tabla 46 que desglosa paso a paso la ruta seguida por la solución del usuario.

- El campo **ID** almacena el identificador de la ruta que se está activando.
- **datosRe** almacena el método que se está activando en la solución del usuario, por ejemplo:
    - En el ID 42 el método “STARTED” hacer referencia que el usuario acaba de ejecutar una solución.
    - En cambio, para el ID 43 el método “MOVE” se refiere a que el primer bloque colocado es un bloque move, muestra el id del bloque que se está activando, y finalmente dice si su ejecución fue correcta o no.
- El campo ***datosJs*** almacena las coordenadas en las cuales se está moviendo el pegman, por ejemplo:
    - En el ID 42 las coordenadas tanto para el eje X y Y son 0, ya que está es la posición inicial del pegman y su dirección es 1, esto quiere decir que su dirección es hacia la derecha.
    - En cambio, para el ID 43 las coordenadas en Y sigue siendo 0 pero para el eje X cambio a 1, esto quiere decir que el pegman sufrió el movimiento de un cuadro en el eje de las X, mientras su dirección sigue siendo hacia la derecha del camino.

Tabla 46. Explicación de las propiedades de la ruta generada.

ID	datosRe	datosJs
42	{"method":"STARTED","args":[],"id":"STARTED"}	{"y":0,"x":0,"direction":1}
43	{"method":"MOVE","args":[],"id":"*ru?9OThQ6Se#_8rD*ZJ","data":[true,1]}	{"y":0,"x":1,"direction":1}
44	{"method":"MOVE","args":[],"id":"Bba1Fk{|qscI1RAE1q{m","data":[true,1]}	{"y":0,"x":2,"direction":1}
45	{"method":"MOVE","args":[],"id":"-w+W;BYuuTLH[j(tg1._","data":[true,1]}	{"y":0,"x":3,"direction":1}
46	{"method":"FINISHED","args":[],"id":"FINISHED"}	{"y":0,"x":3,"direction":1}

La tabla 47 almacena los datos: ID y Tiempo al momento que el usuario oprime el botón ejecutar.

Tabla 47. Acciones registradas del tiempo al momento presionar el botón ejecutar.

Propiedad Tiempo	Significado
ID	Identificador del tiempo
Tiempo	Tiempo en segundos que tardó el usuario en ejecutar una respuesta.

Por último, la tabla 48 almacena los datos: Id, User_name, Password y Edad en el momento en el que se registra un nuevo usuario en la herramienta de programación.

Tabla 48. Almacenamiento de los datos del usuario registrados en la herramienta de programación.

Propiedad Usuario	Significado
Id	Identificador de usuario
User_name	Nombre creado por el usuario
Password	Contraseña creada por el usuario
Edad	Edad del usuario

4.12 Implementación de la Lógica Difusa

Ya que se tienen tanto las interacciones y los códigos generados por el usuario, el siguiente paso es definir las características que ayudarán a realizar el análisis para la generación de rutas de

aprendizaje. En la tabla 49 se presenta el conjunto de datos recolectados por la herramienta de programación. Mientras que en la tabla 50 se presentan las características seleccionadas para la predicción de rutas de aprendizaje usando lógica difusa. A continuación, se justifica las características seleccionadas para el sistema difuso.

- Medir el **tiempo** es clave para evaluar la eficiencia y rendimiento del usuario al resolver ejercicios de programación. Ante la falta de una escala oficial de tiempos ideales, un sistema de lógica difusa permite definir rangos y categorizar el desempeño de forma efectiva.
- La variable **intentos totales** es un indicador fundamental para evaluar el nivel de dominio y la estrategia de resolución del usuario. Un número reducido de intentos sugiere una comprensión sólida del problema para su resolución; por el contrario, una alta cantidad de intentos puede reflejar una metodología de ensayo y error, o bien, dificultades de entendimiento del problema. Ya que no existe un límite global que separe una cantidad de intentos aceptable de una excesiva (ya que esto varía según la dificultad de cada ejercicio), la implementación de un sistema de lógica difusa permite modelar esta incertidumbre. Así, es posible clasificar el esfuerzo del usuario en conjuntos difusos (ejemplo., bajo, moderado, alto) y evaluar su desempeño de forma más contextualizada.
- La **altura de la solución** constituye una métrica esencial para analizar la estructural y la optimización del código desarrollado por el usuario. Una solución óptima no solo debe cumplir con el objetivo, sino hacerlo con una estructura eficiente, evitando redundancias o anidamientos innecesarios que incrementen su altura. Debido a que la altura ideal es depende de las restricciones específicas de cada problema de programación, no es factible establecer un valor numérico como estándar. El uso de lógica difusa proporciona la flexibilidad para evaluar esta característica, permitiendo categorizar la arquitectura de la solución según sus grados de pertenencia y determinando qué tan cerca se encuentra el usuario de una implementación optimizada.
- Por su parte, la **altura del error** es una característica clave para definir la gravedad y la profundidad de las fallas lógicas cometidas por el usuario. No todos los errores tienen el mismo impacto: un error superficial puede deberse a un simple fallo o descuido, mientras que un error con una gran altura puede indicar deficiencias en la comprensión de la lógica de programación. Valorar la magnitud de un error es un proceso cualitativo y subjetivo. Por ello, el sistema de lógica difusa es la herramienta idónea, ya que permite transitar de una

evaluación correcto/incorrecto a una evaluación gradual, cuantificando la severidad del error para proporcionar una retroalimentación más precisa y adaptada al proceso de aprendizaje.

Tabla 49. Conjuntos de datos recolectados por la herramienta de programación.

Datos	Significado
ID_User	Número de identificador de usuario.
Intento	Número de intento que se está analizando.
Problema_ID	Identificador del problema que se está resolviendo.
Tiempo	Tiempo de ejecución del intento a analizar.
Dificultad S.O (Solución Óptima)	Dificultad de la solución óptima.
Dificultad S.A. (Solución Usuario)	Dificultad de la solución generada por el usuario.
Altura S.O. (Solución Óptima)	Altura de la solución óptima.
Altura S.A. (Solución Usuario)	Altura de la solución generada por el usuario.
Éxito	Resolvió correcta o incorrectamente el problema analizado.
Tipo Error	Error que se encontró en el código (en caso de haberlo).
Altura Error	Altura donde se encontró el error en el código (en caso de haberlo).
Total Intentos	Número de intentos que necesito el usuario para resolver el ejercicio analizado.

Tabla 50. Características seleccionadas para el sistema difuso.

Datos	Significado
Tiempo	Tiempo de ejecución del intento a analizar.
Total Intentos	Altura de la solución generada por el usuario.
Altura S.A.	Dificultad de la solución generada por el usuario.

Altura Error	Altura donde se encontró el error en el código (en caso de haberlo).

4.13 Análisis Experimental

La lógica difusa nos ayuda a modelar la incertidumbre y la imprecisión en grados de pertenencia cuando tenemos datos ambiguos o que no se pueden analizar de una forma matemáticamente exacta. Por ello que se eligió este método para la generación de las rutas de aprendizaje, ya que no se puede medir exactamente una solución óptima de los ejercicios de programación. Esto debido a:

- que no existe un estándar oficial de alguna entidad regulatoria de programación que muestre cual es el tiempo ideal para resolver los ejercicios de programación. Ya que el tiempo para resolver un ejercicio de un adulto no es el mismo que un niño, o el tiempo que puede tomarse en resolver un ejercicio de programación con conocimientos avanzados a alguien que apenas va empezando, o el tiempo en resolver un ejercicio de programación "A" no va ser el mismo que resolver un ejercicio "B".
- Además de no tener una forma exacta de medir los tipos de errores que se cometen al resolver los ejercicios y la dificultad de los mismos. Estas consideraciones son debatibles ya que las hace el investigador, justificando los valores de cada error y su clasificación. Similar a como se hizo en la herramienta de Dr. Scratch. Por lo cual, al no tener una manera exacta de medir los tiempos para resolver los ejercicios de programación y como calcular los errores en el código se utilizará la lógica difusa en la presente investigación.

4.14 Modelado usando Lógica Difusa

Para implementar la lógica difusa, se deben de implementar los siguientes cuatro pasos:

1. **Definir las variables difusas**: estas variables son las que se usarán como entrada al sistema difuso. En este trabajo se definieron las siguientes variables de: tiempo, solución, Intentos totales, altura del error. Se eligieron estas variables ya que cada una de ellas por sus propiedades se les puede agregar un grado de incertidumbre, el cual es fundamental para para realizar el modelado usando lógica difusa.
2. **Definir los conjuntos difusos:** Estos son los rangos de valores para cada variable difusa. En este trabajo se normalizaron los valores de las variables en un rango de 0 a 1, donde 1

representa el valor óptimo y los valores más alejados a 1 son los menos ideales. Se eligió estandarizar los valores entre 0 y 1 por que los valores de las características eran muy diferentes, y se optó por asegurar el modelado correcto normalizando los valores de las variables. Los rangos para las variables utilizables son las siguientes:

a. **Variable tiempo:**
    i. **valor alto:** de 0 a 0.5
    ii. **valor medio:** de 0.4 a 0.8
    iii. **valor bajo:** de 0.7 a 1.5
b. **Altura de la solución:**
    i. **Mala:** de 0 a 0.5
    ii. **Regular:** de 0.4 a 0.8
    iii. **Buena:** de 0.7 a 1.5
c. **Intentos totales:**
    i. **Altos:** 0 a 0.5
    ii. **Medios:** de 0.4 a 0.8
    iii. **Bajo:** de 0.7 a 1.5
d. **Altura del error:**
    i. **Alta:** de 0 a 0.5
    ii. **Media:** de 0.4 a 0.8
    iii. **Baja:** de 0.7 a 1.5

En la ilustración 19, se muestran graficados los conjuntos difusos junto con sus rangos y valores.

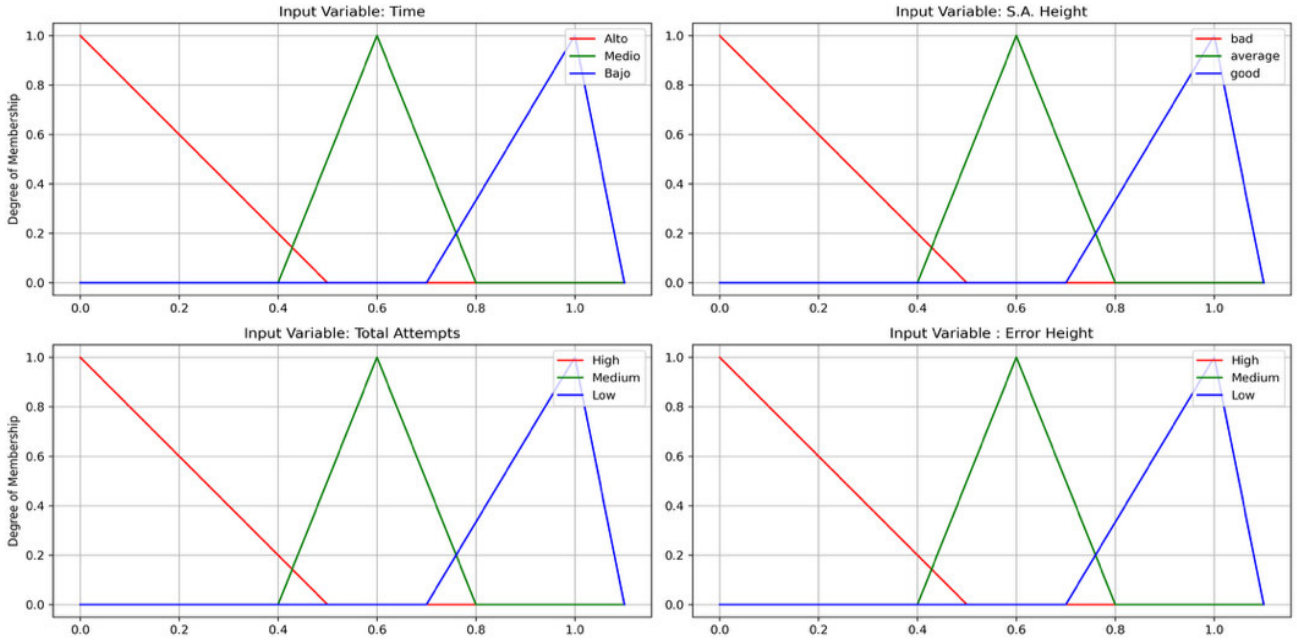


Ilustración 19. Conjunto de valores difusos.

- Cada variable de entrada tiene tres estados de posible salida:
    - **Alto, medio y bajo para las variables:** Tiempo, Total de intentos y altura del error.
    - **Malo, Regular, Bueno para la variable:** Altura de la Solución (SA).

Ejemplo: clasificación de la variable tiempo: Rango de tiempo alto: 0 a 0.5, Rango de tiempo medio: 0.4 a 0.8 y rango de tiempo bajo: 0.7 a 1.5.

3. **Crear las reglas:** Como tercer paso, se generan las reglas difusas. Estas se crean combinando las salidas de las variables de entrada y asignando un resultado. Por ejemplo, si queremos hacer una regla que dé como resultado pasar al siguiente ejercicio las salidas deben de tener: un tiempo bajo, una altura buena, intentos bajos y un error de altura bajo. Esta regla se vería de la siguiente forma: ctrl.Rule(time['low'] & height['good'] & attempts['low'] & error_height['low'], action['advance']) donde: ctrl.Rule es la función en Python para la creación de la regla, y las variables dentro del paréntesis (“()”) corresponden a los valores difusos y sus grados de pertenencia. Para asegurar que todos los valores produzcan un resultado, se definieron todas las combinaciones de salidas posibles. A continuación, se muestran las reglas que se definieron:

4. Si el *tiempo es medio*, la *altura regular*, el *número de intentos es medio* y la *altura del error es medio*, entonces realiza un ***ejercicio de refuerzo***.
5. *Si el tiempo es bajo, la altura es buena, el número de intentos es bajo y la altura del error es bajo, entonces pasa al* ***siguiente ejercicio.***
6. *Si el tiempo es bajo, la altura es buena, el número de intentos es bajo y la altura del error es medio, entonces pasa al* ***siguiente ejercicio****.*
7. *Si el tiempo es bajo, la altura es buena, el número de intentos es bajo y la altura del error es alto, entonces pasa al* ***siguiente ejercicio****.*
8. *Si el tiempo es bajo, la altura es buena, el número de intentos es medio y la altura del error es bajo, entonces pasa al* ***siguiente ejercicio****.*
9. *Si el tiempo es bajo, la altura es buena, el número de intentos es medio y la altura del error es medio, entonces pasa al* ***siguiente ejercicio****.*
10. *Si el tiempo es bajo, la altura es buena, el número de intentos es medio y la altura del error es alto, entonces realiza un* ***ejercicio de refuerzo****.*
11. *Si el tiempo es bajo, la altura es buena, el número de intentos es alto y la altura del error es bajo, entonces pasa al* ***siguiente ejercicio****.*
12. *Si el tiempo es bajo, la altura es buena, el número de intentos es alto y la altura del error es medio, entonces realiza un* ***ejercicio de refuerzo****.*
13. *Si el tiempo es bajo, la altura es buena, el número de intentos es alto y la altura del error es alto, entonces realiza un* ***ejercicio de refuerzo****.*
14. *Si el tiempo es bajo, la altura es regular, el número de intentos es bajo y la altura del error es bajo, entonces pasa al* ***siguiente ejercicio****.*
15. *Si el tiempo es bajo, la altura es regular, el número de intentos es bajo y la altura del error es medio, entonces pasa al* ***siguiente ejercicio****.*
16. *Si el tiempo es bajo, la altura es regular, el número de intentos es bajo y la altura del error es alto, entonces realiza un* ***ejercicio de refuerzo****.*
17. *Si el tiempo es bajo, la altura es regular, el número de intentos es medio y la altura del error es bajo, entonces pasa al* ***siguiente ejercicio****.*
18. *Si el tiempo es bajo, la altura es regular, el número de intentos es medio y la altura del error es medio, entonces realiza un* ***ejercicio de refuerzo****.*
19. *Si el tiempo es bajo, la altura es regular, el número de intentos es medio y la altura del error es alto, entonces realiza un* ***ejercicio de refuerzo****.*

20. *Si el tiempo es bajo, la altura es regular, el número de intentos es alto y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
21. *Si el tiempo es bajo, la altura es regular, el número de intentos es alto y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
22. *Si el tiempo es bajo, la altura es regular, el número de intentos es alto y la altura del error es alto, entonces realiza un **ejercicio de refuerzo**.*
23. *Si el tiempo es bajo, la altura es mala, el número de intentos es bajo y la altura del error es bajo, entonces pasa al **siguiente ejercicio**.*
24. *Si el tiempo es bajo, la altura es mala, el número de intentos es bajo y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
25. *Si el tiempo es bajo, la altura es mala, el número de intentos es bajo y la altura del error es alto, entonces realiza un **ejercicio de refuerzo**.*
26. *Si el tiempo es bajo, la altura es mala, el número de intentos es medio y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
27. *Si el tiempo es bajo, la altura es mala, el número de intentos es medio y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
28. *Si el tiempo es bajo, la altura es mala, el número de intentos es medio y la altura del error es alto, entonces realiza un **ejercicio de refuerzo**.*
29. *Si el tiempo es bajo, la altura es mala, el número de intentos es alto y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
30. *Si el tiempo es bajo, la altura es mala, el número de intentos es alto y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
31. *Si el tiempo es bajo, la altura es mala, el número de intentos es alto y la altura del error es alto, entonces **repite el ejercicio**.*
32. *Si el tiempo es medio, la altura es buena, el número de intentos es bajo y la altura del error es bajo, entonces pasa al **siguiente ejercicio**.*
33. *Si el tiempo es medio, la altura es buena, el número de intentos es bajo y la altura del error es medio, entonces pasa al **siguiente ejercicio**.*
34. *Si el tiempo es medio, la altura es buena, el número de intentos es bajo y la altura del error es alto, entonces realiza un **ejercicio de refuerzo**.*
35. *Si el tiempo es medio, la altura es buena, el número de intentos es medio y la altura del error es bajo, entonces pasa al **siguiente ejercicio**.*

36. *Si el tiempo es medio, la altura es buena, el número de intentos es medio y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
37. *Si el tiempo es medio, la altura es buena, el número de intentos es medio y la altura del error es alto, entonces realiza un **ejercicio de refuerzo**.*
38. *Si el tiempo es medio, la altura es buena, el número de intentos es alto y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
39. *Si el tiempo es medio, la altura es buena, el número de intentos es alto y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
40. *Si el tiempo es medio, la altura es buena, el número de intentos es alto y la altura del error es alto, entonces realiza un **ejercicio de refuerzo**.*
41. *Si el tiempo es medio, la altura es regular, el número de intentos es bajo y la altura del error es bajo, entonces pasa **al siguiente ejercicio**.*
42. *Si el tiempo es medio, la altura es regular, el número de intentos es bajo y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
43. *Si el tiempo es medio, la altura es regular, el número de intentos es bajo y la altura del error es alto, entonces realiza un **ejercicio de refuerzo**.*
44. *Si el tiempo es medio, la altura es regular, el número de intentos es medio y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
45. *Si el tiempo es medio, la altura es regular, el número de intentos es medio y la altura del error es medio, entonces realiza un **ejercicio de refuerzo.***
46. *Si el tiempo es medio, la altura es regular, el número de intentos es medio y la altura del error es alto, entonces realiza un **ejercicio de refuerzo**.*
47. *Si el tiempo es medio, la altura es regular, el número de intentos es alto y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
48. *Si el tiempo es medio, la altura es regular, el número de intentos es alto y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
49. *Si el tiempo es medio, la altura es regular, el número de intentos es alto y la altura del error es alto, entonces **repite el ejercicio**.*
50. *Si el tiempo es medio, la altura es mala, el número de intentos es bajo y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
51. *Si el tiempo es medio, la altura es mala, el número de intentos es bajo y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*

52. *Si el tiempo es medio, la altura es mala, el número de intentos es bajo y la altura del error es alto, entonces realiza un **ejercicio de refuerzo**.*
53. *Si el tiempo es medio, la altura es mala, el número de intentos es medio y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
54. *Si el tiempo es medio, la altura es mala, el número de intentos es medio y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
55. *Si el tiempo es medio, la altura es mala, el número de intentos es medio y la altura del error es alto, entonces **repite el ejercicio**.*
56. *Si el tiempo es medio, la altura es mala, el número de intentos es alto y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
57. *Si el tiempo es medio, la altura es mala, el número de intentos es alto y la altura del error es medio, entonces **repite el ejercicio**.*
58. *Si el tiempo es medio, la altura es mala, el número de intentos es alto y la altura del error es alto, entonces **repite el ejercicio**.*
59. *Si el tiempo es alto, la altura es buena, el número de intentos es bajo y la altura del error es bajo, entonces pasa al **siguiente ejercicio**.*
60. *Si el tiempo es alto, la altura es buena, el número de intentos es bajo y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
61. *Si el tiempo es alto, la altura es buena, el número de intentos es bajo y la altura del error es alto, entonces realiza un **ejercicio de refuerzo**.*
62. *Si el tiempo es alto, la altura es buena, el número de intentos es medio y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
63. *Si el tiempo es alto, la altura es buena, el número de intentos es medio y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
64. *Si el tiempo es alto, la altura es buena, el número de intentos es medio y la altura del error es alto, entonces realiza un **ejercicio de refuerzo**.*
65. *Si el tiempo es alto, la altura es buena, el número de intentos es alto y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
66. *Si el tiempo es alto, la altura es buena, el número de intentos es alto y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
67. *Si el tiempo es alto, la altura es buena, el número de intentos es alto y la altura del error es alto, entonces **repite el ejercicio**.*

68. *Si el tiempo es alto, la altura es regular, el número de intentos es bajo y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
69. *Si el tiempo es alto, la altura es regular, el número de intentos es bajo y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
70. *Si el tiempo es alto, la altura es regular, el número de intentos es bajo y la altura del error es alto, entonces realiza un **ejercicio de refuerzo**.*
71. *Si el tiempo es alto, la altura es regular, el número de intentos es medio y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
72. *Si el tiempo es alto, la altura es regular, el número de intentos es medio y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
73. *Si el tiempo es alto, la altura es regular, el número de intentos es medio y la altura del error es alto, entonces **repite el ejercicio**.*
74. *Si el tiempo es alto, la altura es regular, el número de intentos es alto y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
75. *Si el tiempo es alto, la altura es regular, el número de intentos es alto y la altura del error es medio, entonces **repite el ejercicio**.*
76. *Si el tiempo es alto, la altura es regular, el número de intentos es alto y la altura del error es alto, entonces **repite el ejercicio**.*
77. *Si el tiempo es alto, la altura es mala, el número de intentos es bajo y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
78. *Si el tiempo es alto, la altura es mala, el número de intentos es bajo y la altura del error es medio, entonces realiza un **ejercicio de refuerzo**.*
79. *Si el tiempo es alto, la altura es mala, el número de intentos es bajo y la altura del error es alto, entonces **repite el ejercicio**.*
80. *Si el tiempo es alto, la altura es mala, el número de intentos es medio y la altura del error es bajo, entonces realiza un **ejercicio de refuerzo**.*
81. *Si el tiempo es alto, la altura es mala, el número de intentos es medio y la altura del error es medio, entonces **repite el ejercicio**.*
82. *Si el tiempo es alto, la altura es mala, el número de intentos es medio y la altura del error es alto, entonces **repite el ejercicio**.*
83. *Si el tiempo es alto, la altura es mala, el número de intentos es alto y la altura del error es bajo, entonces **repite el ejercicio**.*

84. *Si el tiempo es alto, la altura es mala, el número de intentos es alto y la altura del error es medio, entonces **repite el ejercicio**.*
85. *Si el tiempo es alto, la altura es mala, el número de intentos es alto y la altura del error es alto, entonces **repite el ejercicio**.*

4. Cálculo del grado de pertenencia: Como paso final, se calculan las pertenencias de las variables. Esto da como resultado el grado de pertenencia de cada una de ellas. En la ilustración 20, se muestran las funciones de pertenencia. Ejemplo:

a. Un valor menor que 1 tiene como resultado la repetición del ejercicio.
b. Un valor mayor de 0.5 a 2 conducirá a un ejercicio de refuerzo.
c. Un valor mayor de 1.5 hasta 3.0: da como resultado pasar al siguiente ejercicio. En la siguiente sección se muestran todos los resultados del sistema difuso.

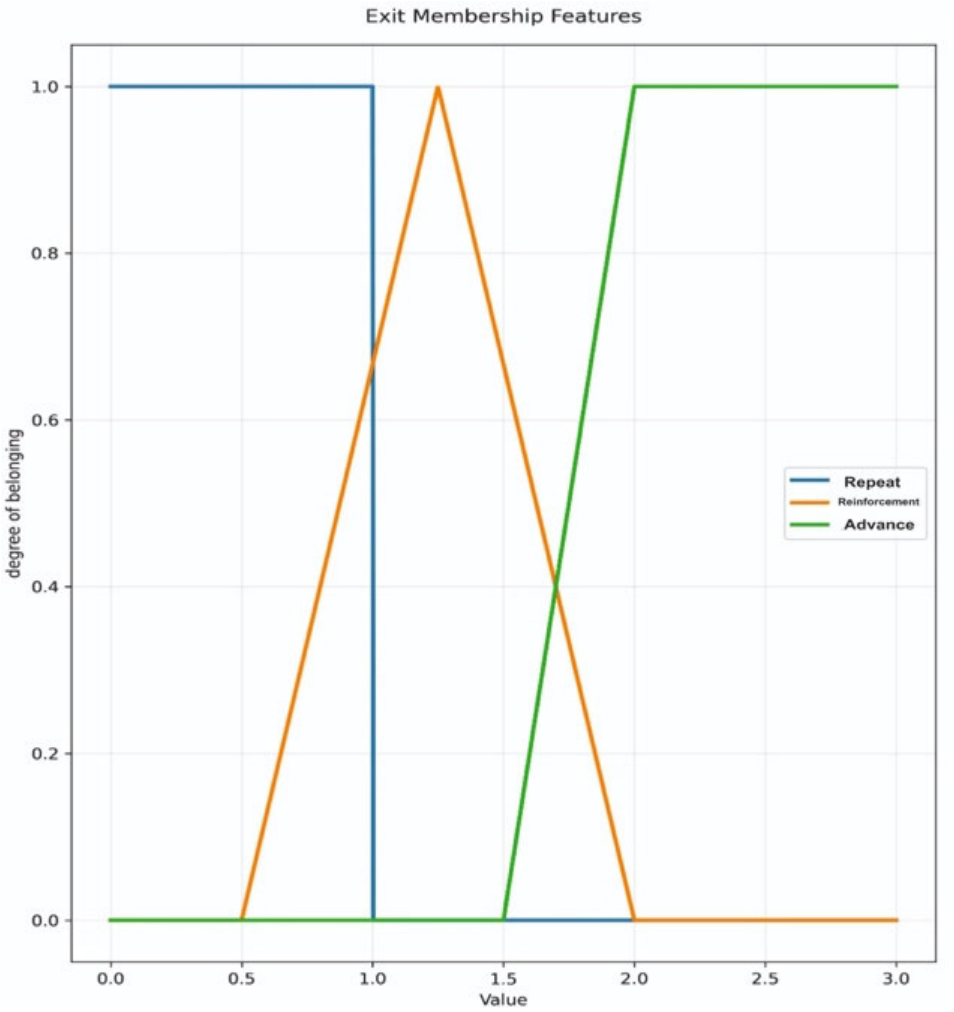


Ilustración 20 Salida de las funciones de membresía.

Capítulo 5

5. Resultados

En esta sección se muestran los resultados obtenidos de aplicar 20 ejercicios de programación a 27 usuarios de entre 8 y 10 años, de los cuales 16 eran niñas y 11 niños.

En la ilustración 21, se presenta la gráfica con el número total de ejercicios que realizaron los usuarios, la cual esta ordenada de mayor a menor, en la cual se puede notar que el usuario 20 fue el que realizó el mayor número de ejercicios con 19, mientras que los usuarios 15, 14, 9, 7 y 5 solo realizarón 11.

Ilustración 21. Total, de Ejercicios realizados por usuarios.

En la ilustación 22, se muestra la gráfica del número total de intentos por ejercicio. Se puede observar que el ejercicio con mayor número de intentos fue el ejercicio 10 con mas de 270 intentos, lo que lo califica como el ejercicios de mayor dificultad de resolver, seguido del ejercicio 7 con 96

intentos y el ejericicio 9 con 94 intentos. Por otra parte el ejercicio con menor número de intentos fue el ejericicio 11 con 3 intentos, seguido del ejericicio 12 y 13 con 12 intentos cada uno.

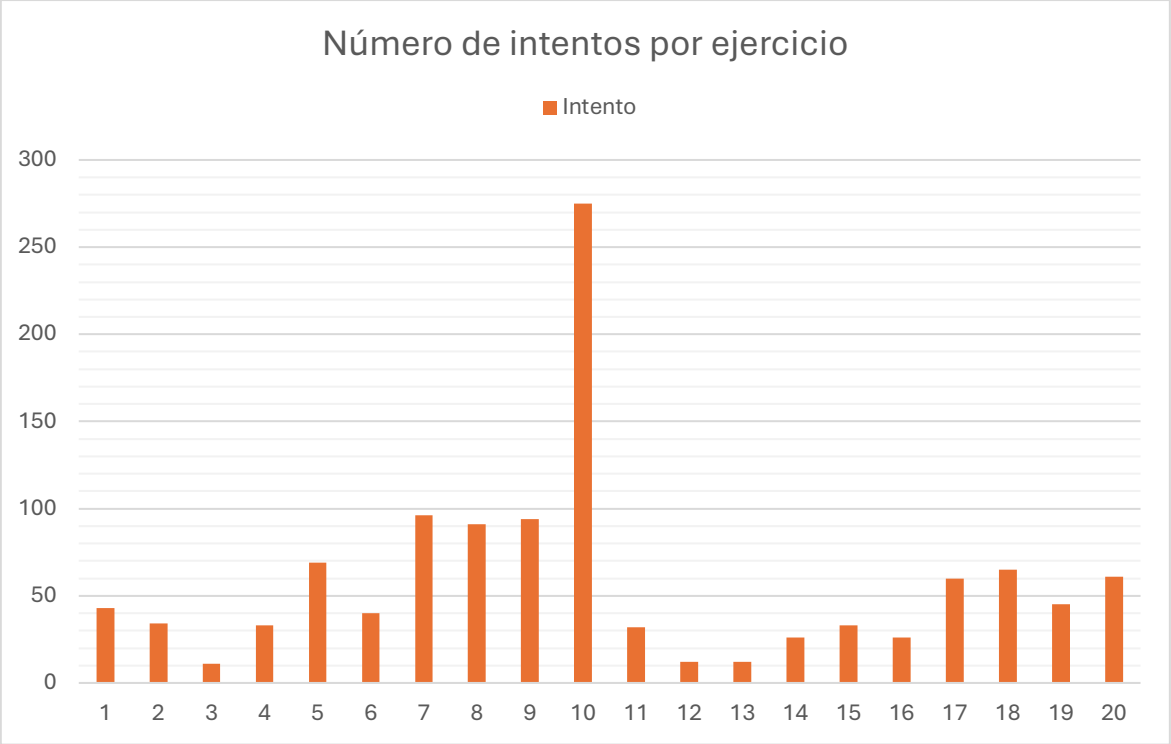


Ilustración 22. Nivel de dificultad por ejercicios.

La ilustración 23, muestra la gráfica de frecuencia de tipos de errores, la cual esta ordenada de mayor a menor. En la gráfica se puede notar que el tipo de error más frecuente es bloques faltantes con mas de 380 errores, seguido de ***move*** en lugar de turn con 130 errores y ***giro izquierda*** en lugar de derecha con 120 errores. Por su parte, el tipo de error con menor frecuencia es ***repeat_until*** en lugar de ***turn*** con una frecuencia de 6, seguido de ***repeat_until*** en lugar de ***move*** con 10 y avanzó en lugar ***de girar derecha*** con una frecuencia de 11.

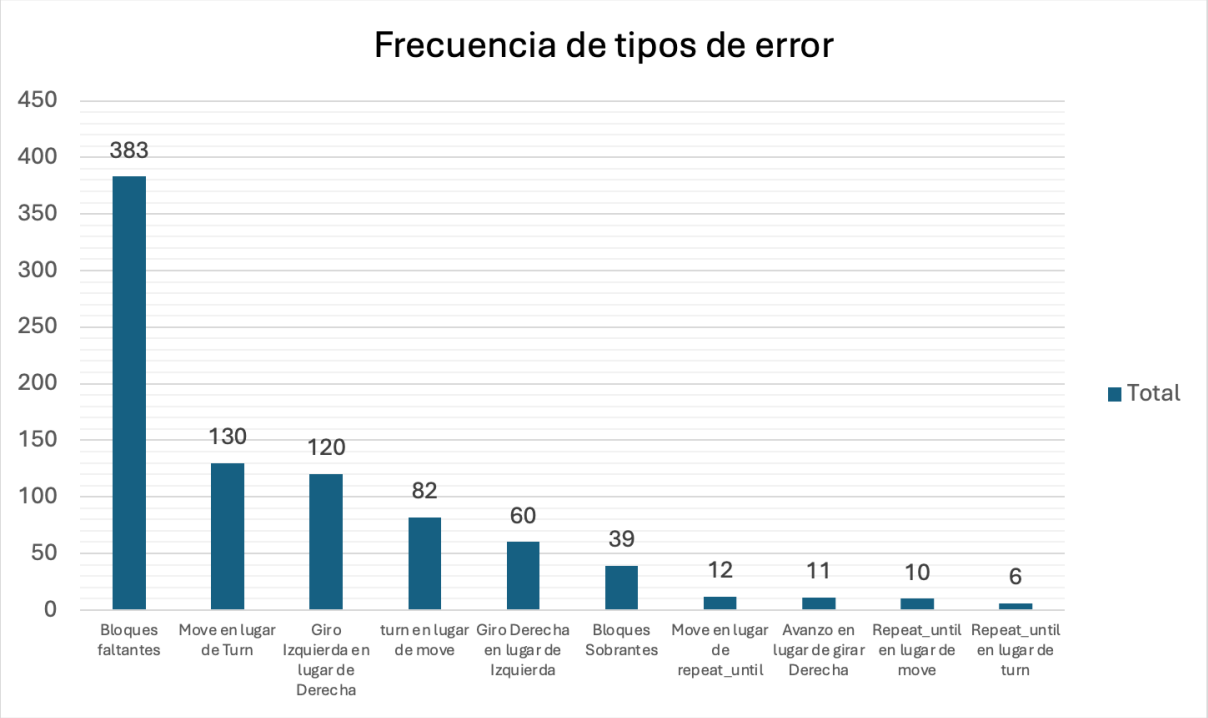


Ilustración 23. Frecuencia de tipos de error.

La ilustración 24, Nos muestra los errores que cometieron los usuarios al momento de realizar el ejercicio 1. El error más recurrente fue ***bloques sobrantes*** con 11 intentos, mientras que para ***bloques faltantes*** el error se presento 10 veces. La similitud de ambos errores puede deberse a la adaptación del usuario con los ejercicios, ya que al ser el primero regularmente suele haber errores del manejo de la lógica y el uso de la plataforma.

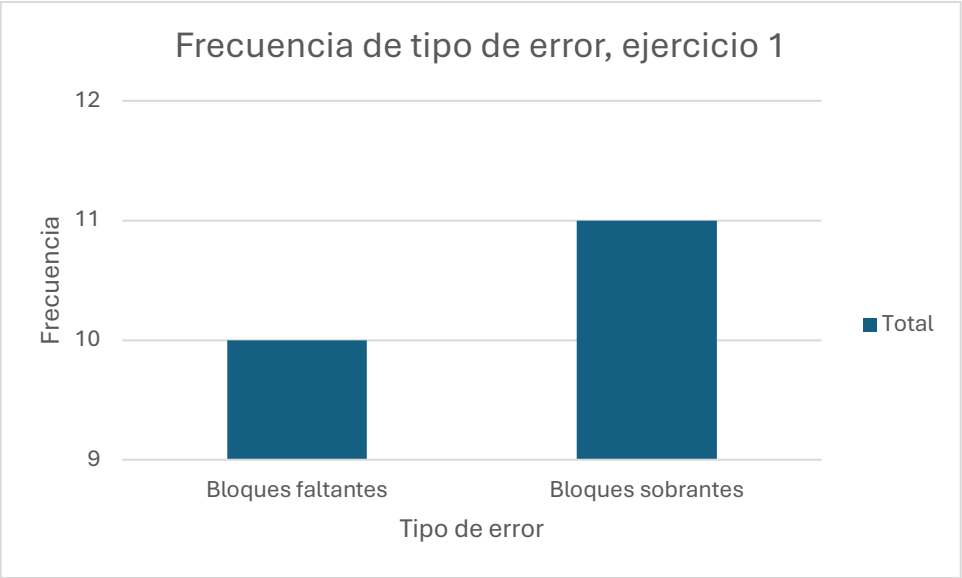


Ilustración 24. Frecuencia de tipo de error, ejercicio 1.

En la ilustración 25, se observa que el error bloques faltantes subió a 13 registros, 3 más que el ejercicio anterior, mientras que los bloques sobrantes bajaron a 4.

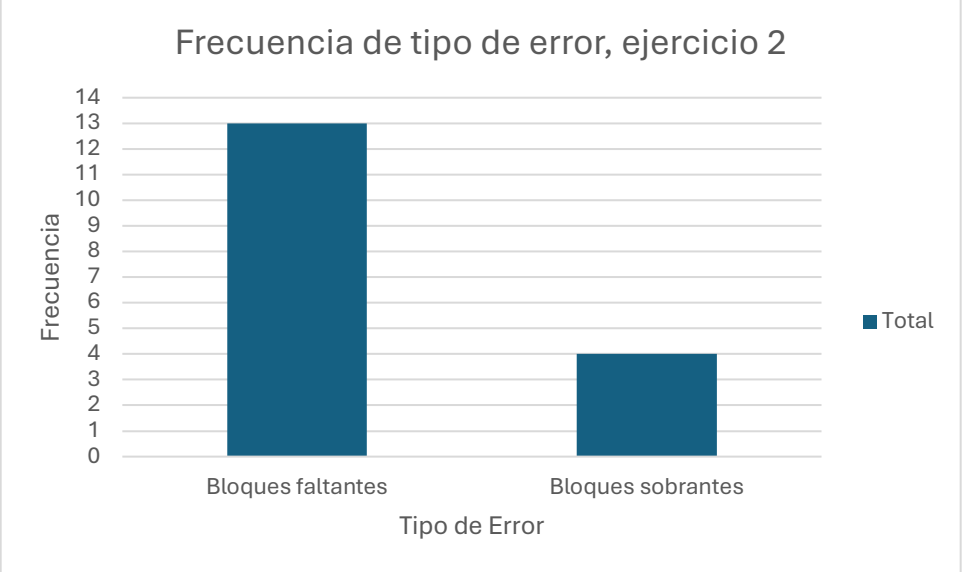


Ilustración 25. Frecuencia de tipo de error, ejercicio 2.

Para el ejercicio 3 (ver ilustración 26) el error de bloque faltantes bajo a 6 y bloques sobrantes bajo a 1, esto puede indicar que a medida que el usuario avanza va comprendiendo más la solución de los problemas y el manejo de la plataforma.

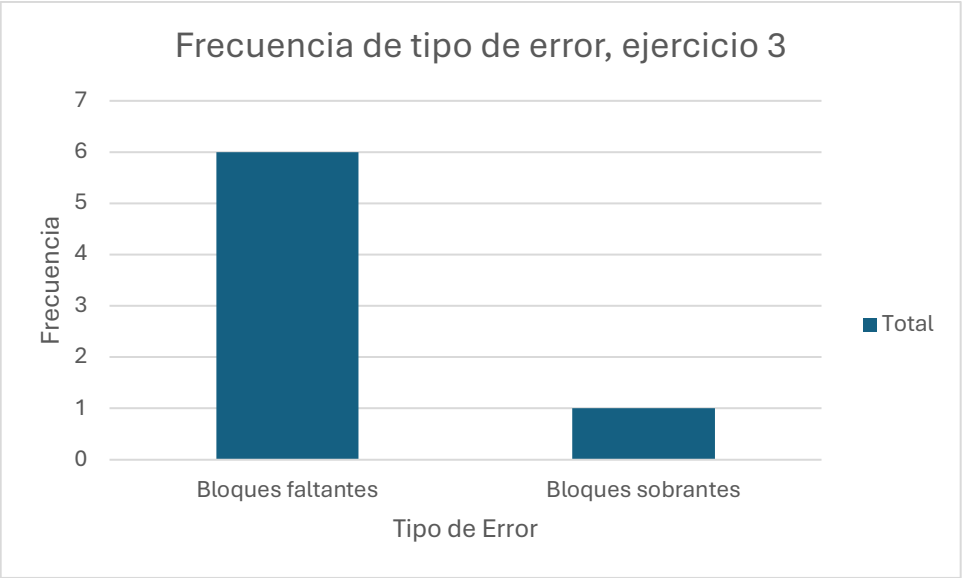


Ilustración 26. Frecuencia de tipo de error, ejercicio 3.

Para el ejercicio 4 (ver ilustración 27) la frecuencia de errores tuvo nuevamente un aumento, para el error ***bloques faltantes*** se tuvieron 22 errores, y para bloques sobrantes 6.

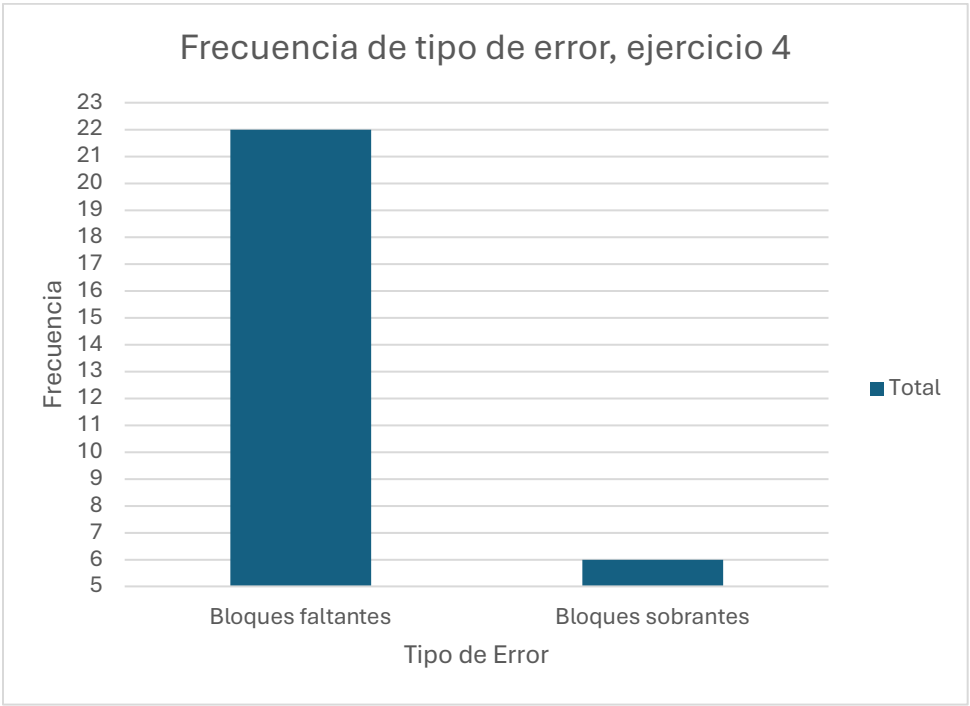


Ilustración 27. Frecuencia de tipo de error, ejercicio 4.

Para el ejercicio 5 (ver ilustración 28), se registraron 4 nuevos errores, ya que desde este ejercicio comienzan los ejercicios con giro. Por lo anterior, el error con más frecuencia fue ***giro izquierda,*** en lugar de ***giro a la derecha*** con 20 registros, seguido de bloques faltantes con 12 y ***turn*** en lugar de ***move*** con 6.

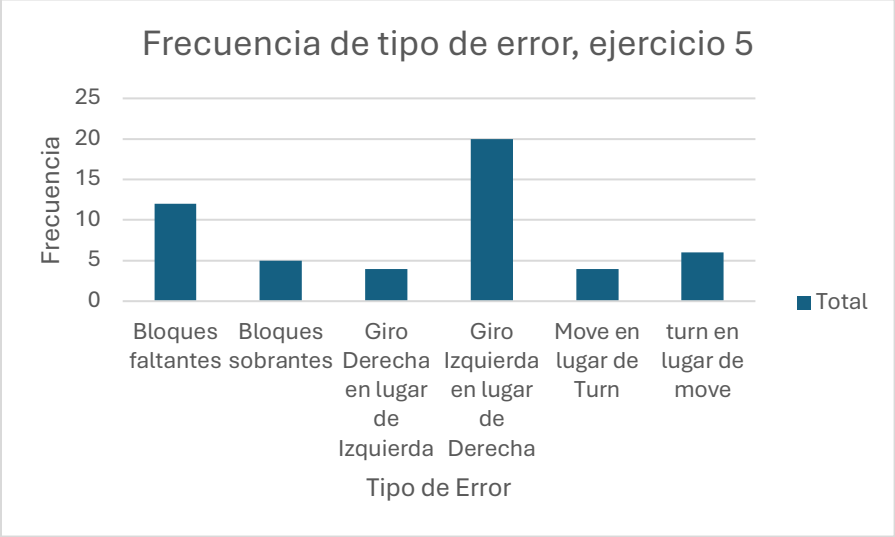


Ilustración 28. Frecuencia de tipo de error, ejercicio 5.

Para el ejercicio 6 (ver ilustración 29) el error más frecuente volvío hacer ***giro a la izquierda*** en lugar de ***giro a la derecha*** con 9 registros, seguido de ***move*** en lugar de ***turn*** y ***bloques faltantes*** con 7 registros cada uno.

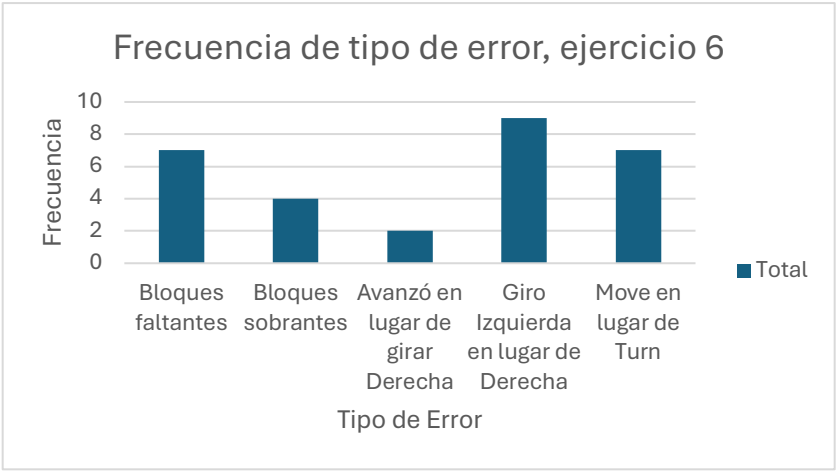


Ilustración 29. Frecuencia de tipo de error, ejercicio 6.

El ejercicio 7 (ver ilustración 30), registro como error más frecuente a ***move*** en lugar de ***turn*** con 29 registros, seguido ***de giro izquierda*** en lugar de ***giro a la derecha*** con 23 registros y ***bloques faltantes*** con 17.

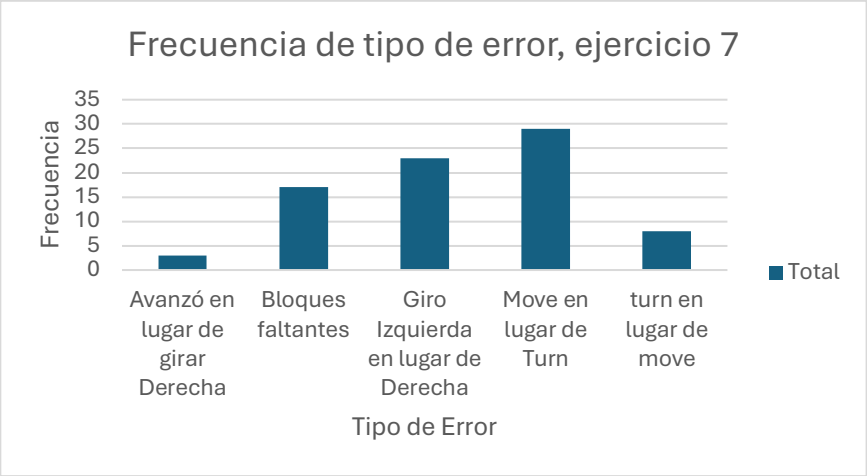


Ilustración 30. Frecuencia de tipo de error, ejercicio 7.

Para el ejercicio 8 (ver ilustración 31) la tendencia se mantiene, siendo ***move*** en lugar de ***turn*** el error más frecuente con 33 registros, seguido de ***bloques faltantes*** con 30 registro.

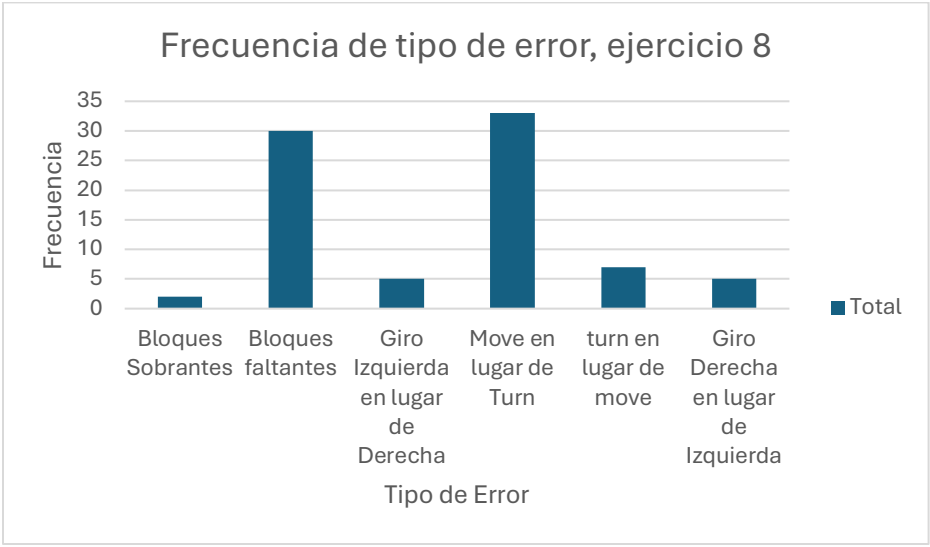


Ilustración 31. Frecuencia de tipo de error, ejercicio 8.

Para el ejercicio 9 (ver ilustración 32) el error más frecuente fue ***giro izquierda*** en lugar de ***giro a la derecha*** con 25 registros, seguido de ***bloques faltantes*** con 21 y ***move*** en lugar de ***turn*** con 16.

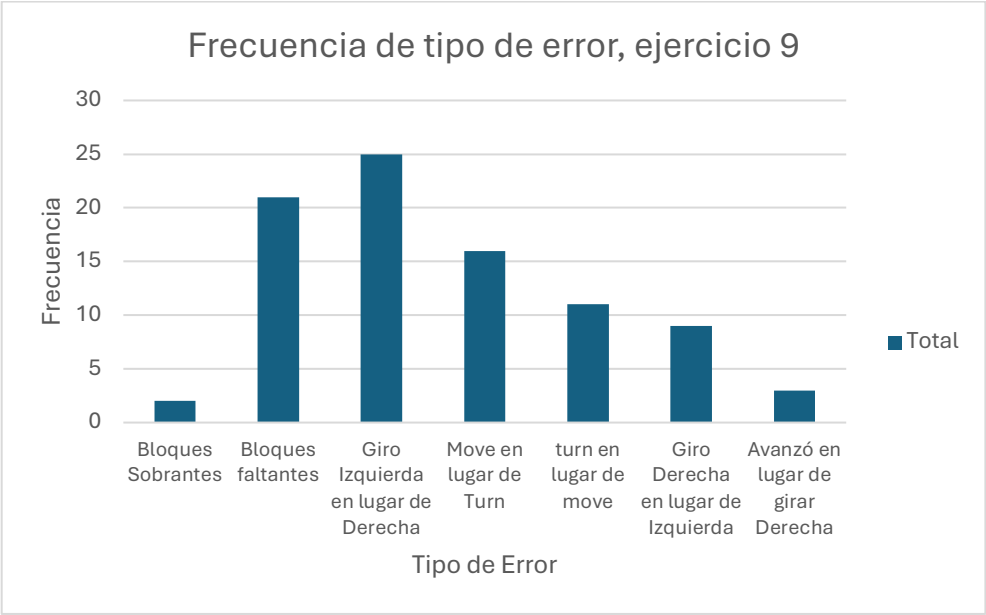


Ilustración 32. Frecuencia de tipo de error, ejercicio 9.

El ejercicio 10 (ver ilustración 33) al ser un ejercicio largo de resolver el error más frecuente fue ***Bloques faltantes*** con más de 170 registros, seguido de ***giro a la derecha*** en lugar de ***giro a la izquierda*** con más de 20 registros y seguido de ***move*** en lugar de ***turn*** y ***turn*** en lugar de ***move*** con 19 registros cada uno.

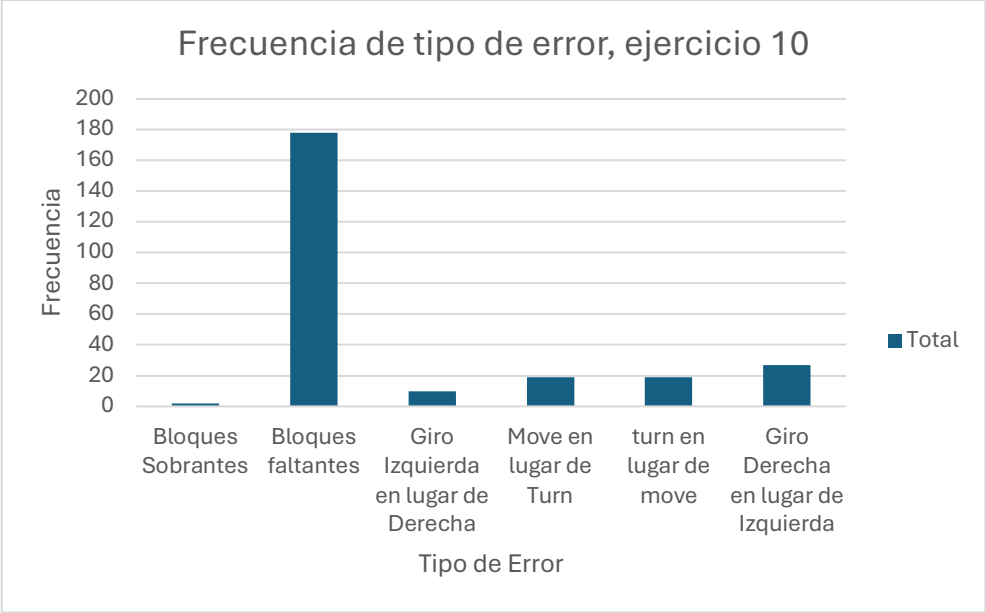


Ilustración 33. Frecuencia de tipo de error, ejercicio 10.

A partir del ejercicio 11 y siguientes comienza los ejercicios con ***repeat until***. Para los ejercicios 11, 12 y 13 no se encontraron errores en soluciones, en cambio para el ejercicio 14 (ver ilustración 34), se encontraron dos errores: ***bloques faltantes*** y ***move*** en lugar de ***repeat until*** con 1 registro cada uno.

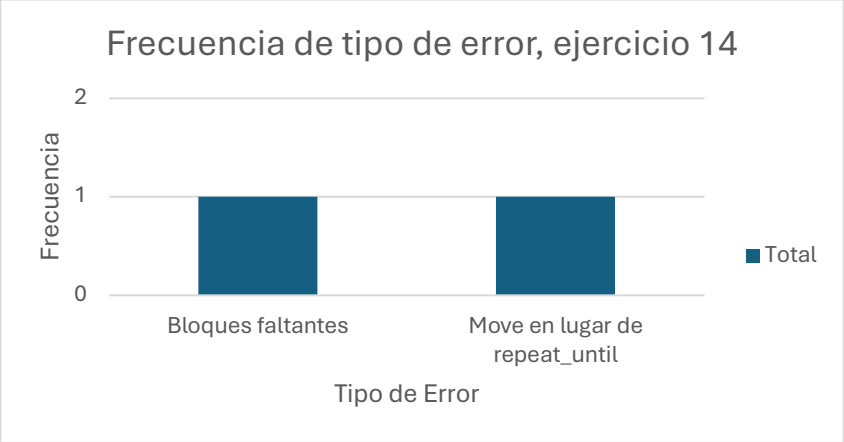


Ilustración 34. Frecuencia de tipo de error, ejercicio 14.

Por su parte el ejercicio 15 (ver ilustración 35), registro como error más frecuente ***bloques faltantes*** con 5 registros, seguido de ***giro a la izquierda*** en lugar de ***giro a la derecha*** con 3.

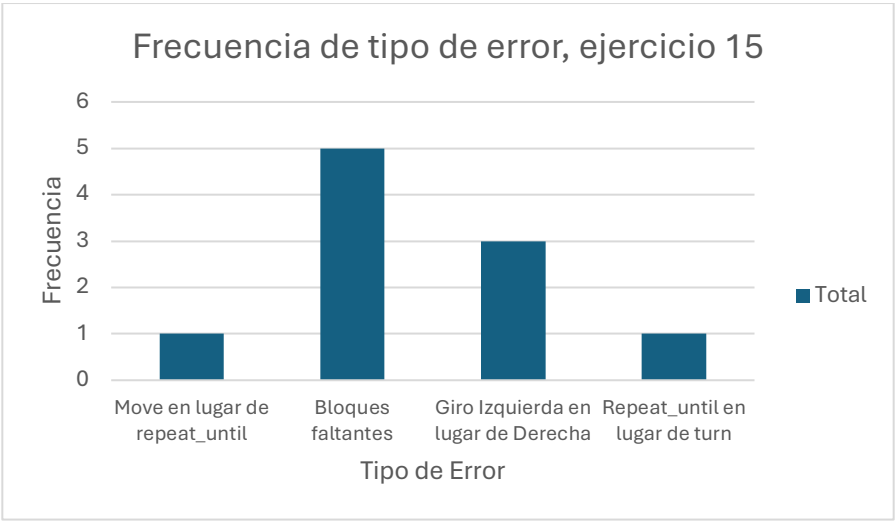


Ilustración 35. Frecuencia de tipo de error, ejercicio 15.

El ejercicio 16 (ver ilustración 36), registro como error más frecuente ***giro a la izquierda*** en lugar de ***giro a la derecha*** con 5 seguido de ***bloques faltantes*** con 4 y ***repeat until*** en lugar de ***turn*** con 3.

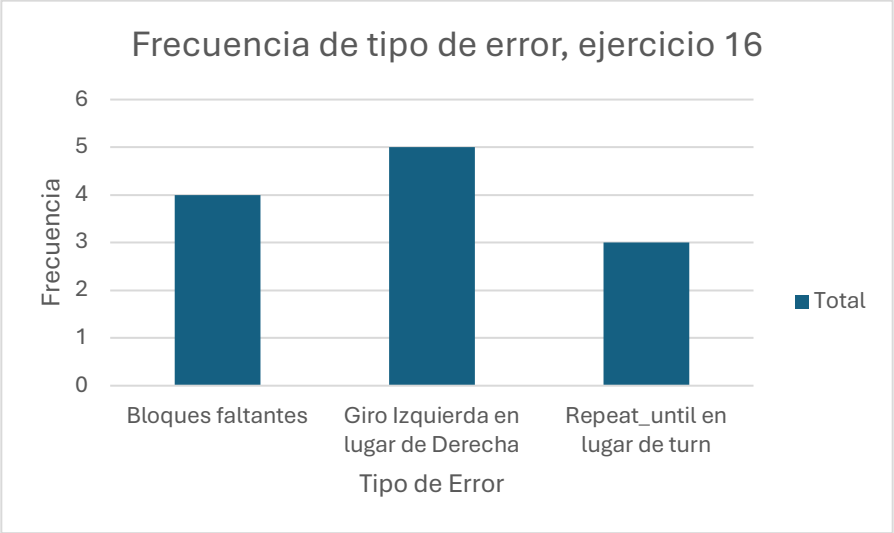


Ilustración 36. Frecuencia de tipo de error, ejercicio 16.

Para el ejercicio 17 (ver ilustración 37), el error más frecuente fue ***turn*** en lugar de ***move*** con con 12 repeticiones, seguido de ***bloques con faltantes*** con 11 y empatado con 7 registros los errores ***repeat until*** en lugar de ***move*** y ***giro a la izquierda*** en lugar de ***giro a la derecha***.

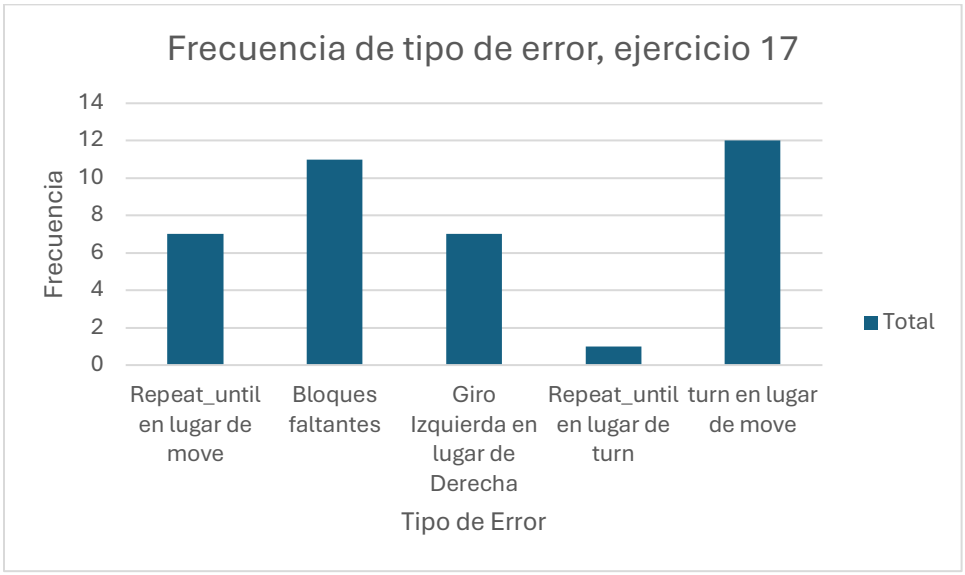


Ilustración 37. Frecuencia de tipo de error, ejercicio 17.

Para el ejercicio 18 (ver ilustración 38), el error más frecuente fue ***bloques faltantes*** con 17 registros, seguido de ***move*** en lugar de ***turn*** con 13 y ***giro a la izquierda*** en lugar de ***giro a la derecha*** con 9.

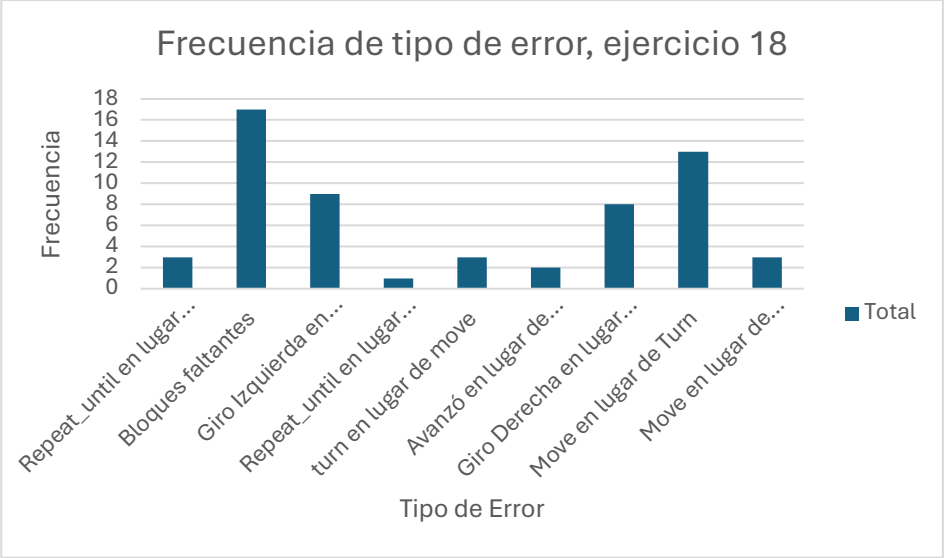


Ilustración 38. Frecuencia de tipo de error, ejercicio 18.

Para el ejercicio 19 (ver ilustración 39), el error más registrado fue ***bloques faltantes*** con 10 registros, seguido de ***turn*** en lugar de ***move*** y ***move*** en lugar de ***repeat until*** con 9 registros cada uno.

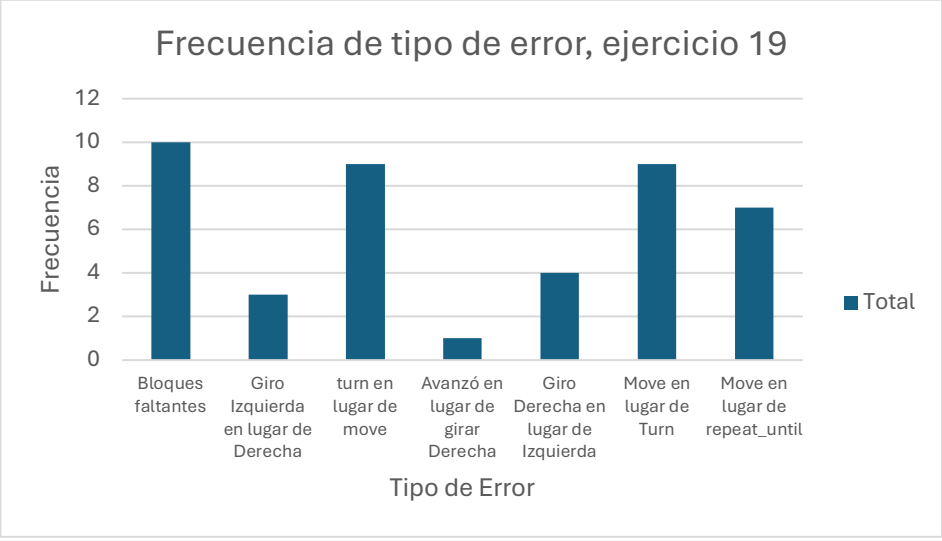


Ilustración 39. Frecuencia de tipo de error, ejercicio 19.

El ejercicio 20 (ver ilustración 40), registro como error más frecuente ***bloques faltantes*** con 19 registros, seguido de ***turn*** en lugar de ***move*** con 7.

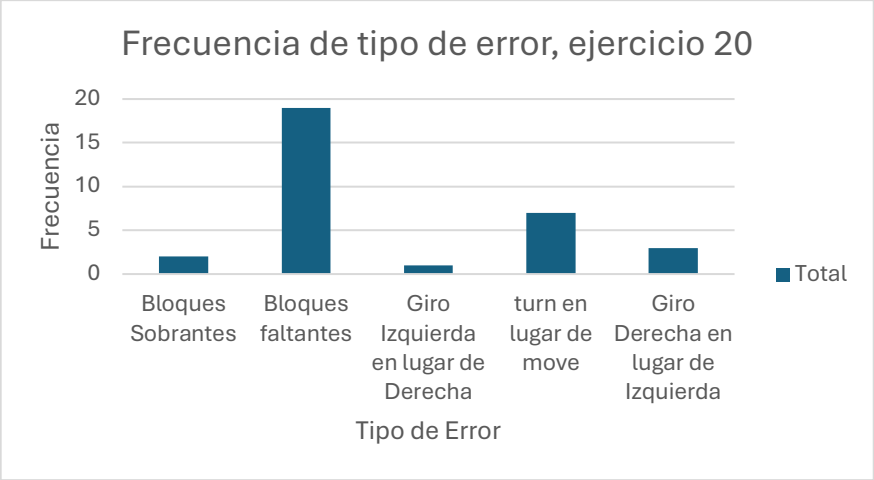


Ilustración 40. Frecuencia de tipo de error, ejercicio 20.

La ilustración 41, muestra la gráfica de tiempo promedio por ejercicio, en donde se puede observar que el usuario 19 es el que tiene mayor promedio de tiempo al realizar los ejercicios con más de 1,487 segundos, seguido del usuario 11 con 1476 segundos y el usuario 10 con 1271 segundos. Por su parte el usuario con menor promedio de tiempo es el usuario 23 con 376 segundos, seguido del usuario 25 con 422 segundos y del usuario 27 con 488 segundos.

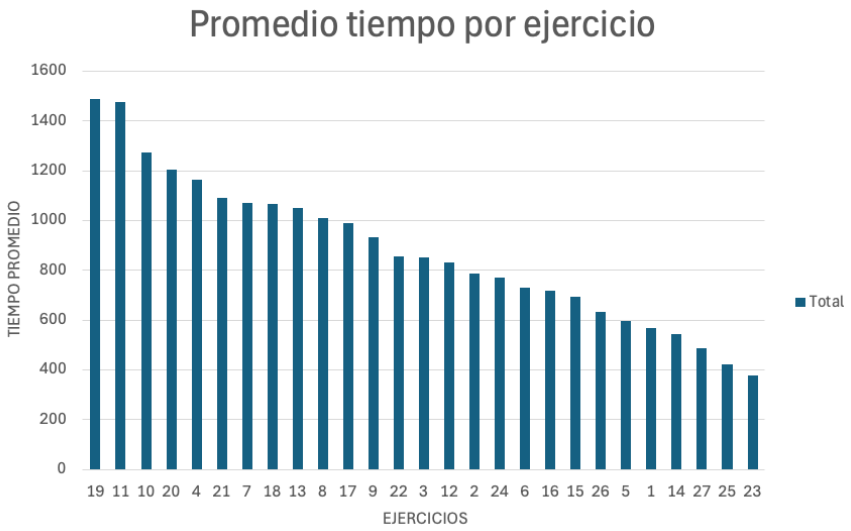


Ilustración 41. Tiempo promedio por ejercicio.

La ilustración 42 nos muestra el total de éxitos y no éxitos que se obtuvo por ejercicios. El ejercicio 10 fue el que mayor número de fallas tuvo con 255 intentos erroneos y sólo 20 intentos existosos, seguido del ejercicio 9 con 85 fallas y sólo 9 éxitos y el ejercicio 8 con 82 fallas y 9 intentos exitosos. Por su parte el ejercicio 11 no tuvo ninguna falla y 32 éxitos, seguido del ejercicio 13 y 12 con 12 éxitos y 0 fallos cada uno de ellos.

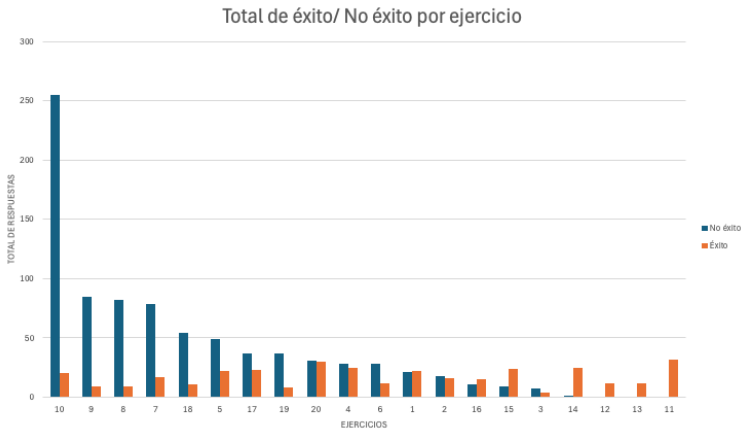


Ilustración 42. Total de ejercicios resueltos con éxito y no éxito.

En la ilustración 43, se muestran los ejercicios resueltos por el usuario 1, asi como el número de intento y el resultado de cada uno de ellos. La gráfica nos muestra que el usuario 1 resolvió correctamente 16 ejercicios de los 17 que resolvió, el único ejercicio incorrecto fuel el ejercicio 1.

Por su parte, el ejercicio con mayor número de intentos fue el ejercicio 10 con 4 intentos, seguido del ejercicio 8 y 4 con 3 intentos cada uno y finalmente el ejercicio 1 con dos intentos. El resto de los ejercicios los resolvió correctamente en la primera oportunidad.

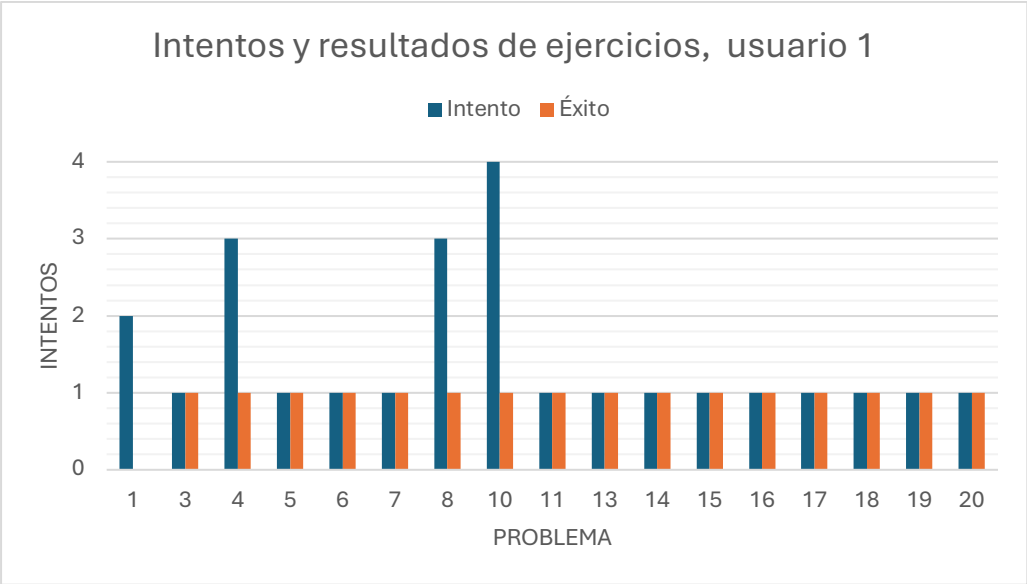


Ilustración 43. Intentos y resultados de los ejercicios de programación, usuario 1.

En la ilustración 44, vemos los ejercicios realizados por el usuario 7, el cual logró resolver de manera exitosa 8 de los 11 ejercicios que realizó; sólo los ejercicios 1, 5 y 7 tuvieron resultados negativos, el ejercicio que tuvo mayores intentos fue el ejercicio 10 con 13 intentos, seguido del ejercicio 9 con 7 intentos. El ejercicio con el menor número de intentos fue el ejercicio 11 con un intento.

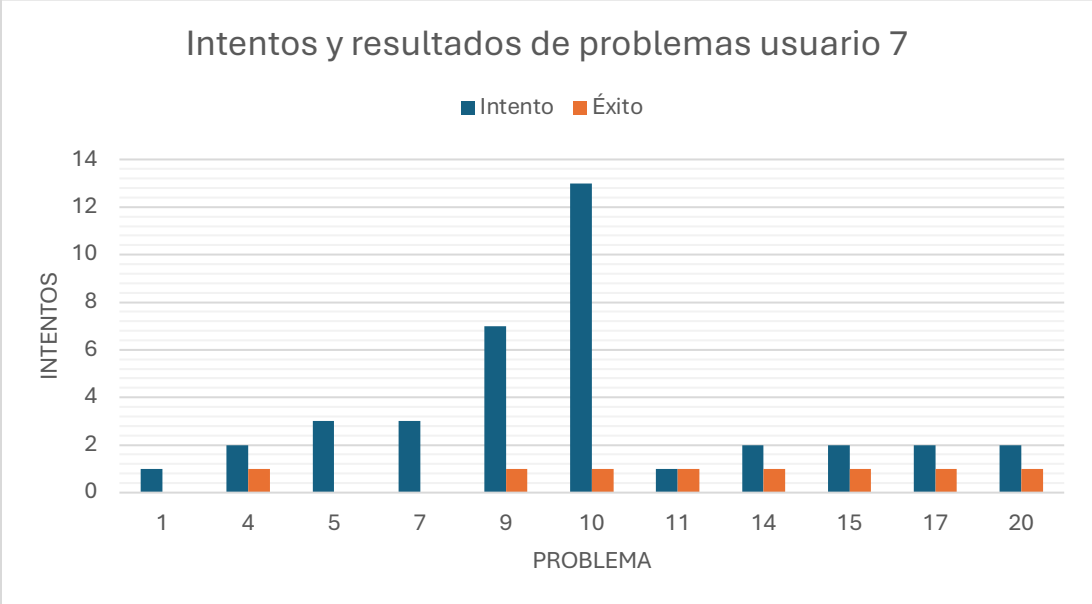


Ilustración 44. Intentos y resultados de los ejercicios de programación, usuario 7.

En la ilustración 45, se muestran los ejercicios realizados por el usuario 14, así como sus números de intentos y sus resultados. Se puede observar que de los 11 ejercicios realizados 4 no fueron contestados de manera correcta y los 7 restantes si, el ejercicio con más intentos fue el ejercicio 10 con 17 intentos, seguido del ejercicio 9 con 13 y del ejercicio 7 con 12. En cambio, los ejercicios con menores intentos fueron los ejercicios 1, 4, 11, 14 y 20 con un intento.

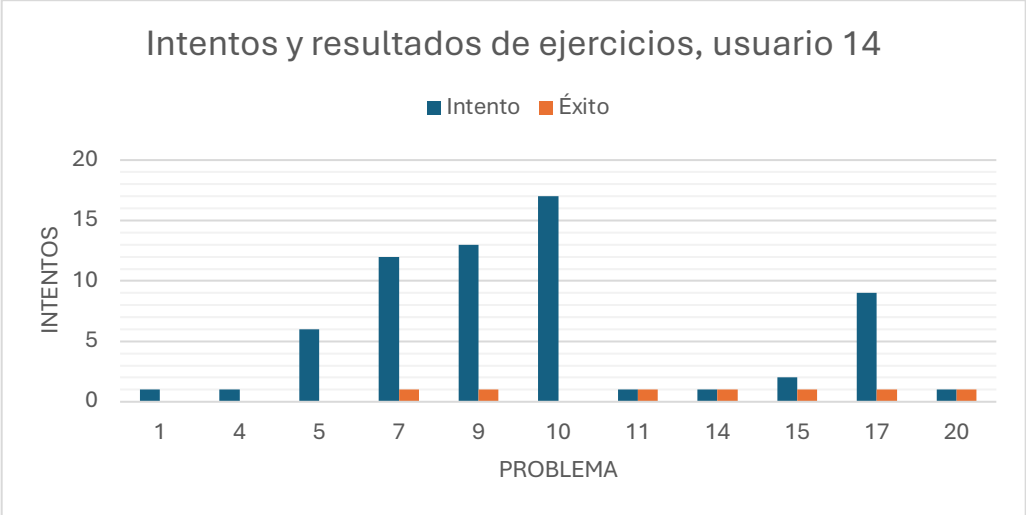


Ilustración 45. Intentos y resultados de los ejercicios de programación, usuario 14.

Por su parte en la ilustración 46, se observa el porcentaje de ejercicios resueltos correcta o incorrectamente de los 20 ejercicios de programación en su primera oportunidad. Los ejercicios 11, 12, y 13 tienen el 100% de efectividad al momento de realizar estos ejercicios, seguido del ejercicio 14 con un poco más del 90% de éxito. Por el contrario, el ejercicio 10 tiene menos del 10% de éxito, seguido del ejercicio 8 y 9 con un 10% y continuando con el ejercicio 7 con un poco más del 15 de éxito.

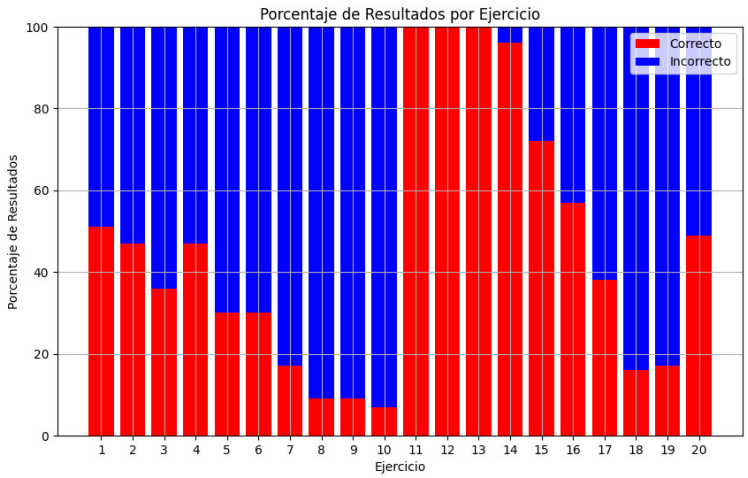


Ilustración 46. Porcentaje de resultados correcto/incorrecto por ejercicio.

En la ilustración 47 se observa el porcentaje de acierto que tuvieron los usuarios al momento de resolver todos sus ejercicios de programación. El usuario 1 fue el que obtuvo el mayor porcentaje de ejercicios resueltos correctamente con un 65% de éxito, seguido del usuario 16 con un 58% y el usuario 6 con 43% de respuestas correctas. Por el contrario, el usuario 14 fue el que menos porcentaje de respuestas correctas obtuvo con un 10% de éxito, seguido del usuario 11 con un 15% y del usuario 20 con un 17% de éxito.

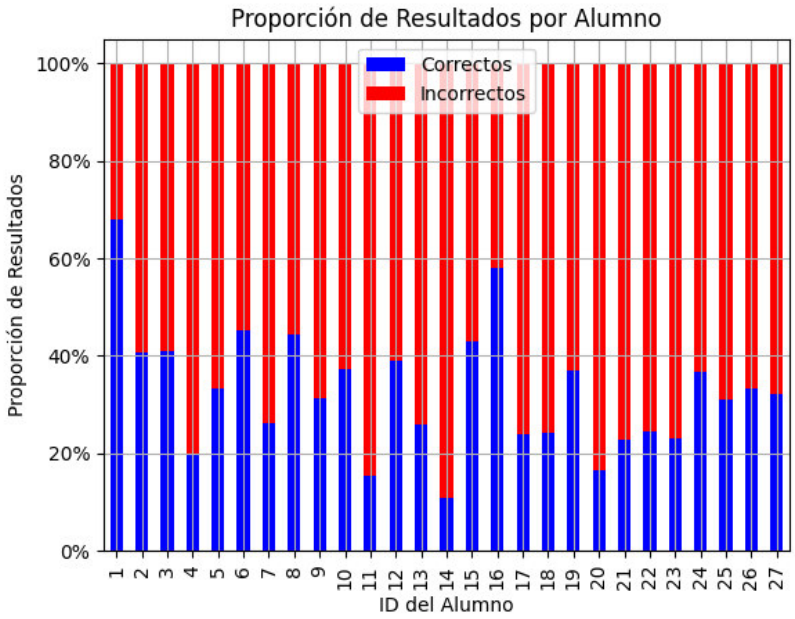


Ilustración 47. Porcentaje de resultados correctos/incorrectos por alumno.

Por otro lado, se analizaron los AST óptimos y los AST intermedios. Como resultado se obtuvieron los errores más frecuentes, los cuales son:

- Bloques faltantes con 383 ocurrencias
- Bloques sobrantes con 215
- Error en campo con 177
- Orden incorrecto con 135
- Y tipo no coincidente con 96

Mientras que en la tabla 51, se muestra la estadística descriptiva de la base de datos. Donde se puede notar que la media por intentos es de 3.87 mientras que la mínima y máxima son de 1 y 29 respectivamente. Por otro, el lado la media de la altura error es de 4.56 y la mínima y máxima son de 0 y 26.

Tabla 51. Estadística descriptiva de la base de datos.

Variable	Media	Mínimo	Máximo	Desviación estándar
Intento	3.87	1	29	3.77
Tiempo	937.9 seg. (15.6min)	5.9	2905.9	617.61
Dificultad S.O.	3.56	1	6	1.56
Dificultad S.A.	3.22	0	25	1.54
Altura S.O.	11.96	2	25	8.22
Altura S.A.	7.97	0	31	6.18
Altura Error	4.56	0	26	5.77
Total Intentos	6.72	1	29	5.61

Por su parte, la ilustración 48 muestra los ejercicios que resolvieron los participantes durante su sesión de programación. Siendo de color azul marino los ejercicios resueltos y de color gris los ejercicios sin resolver.

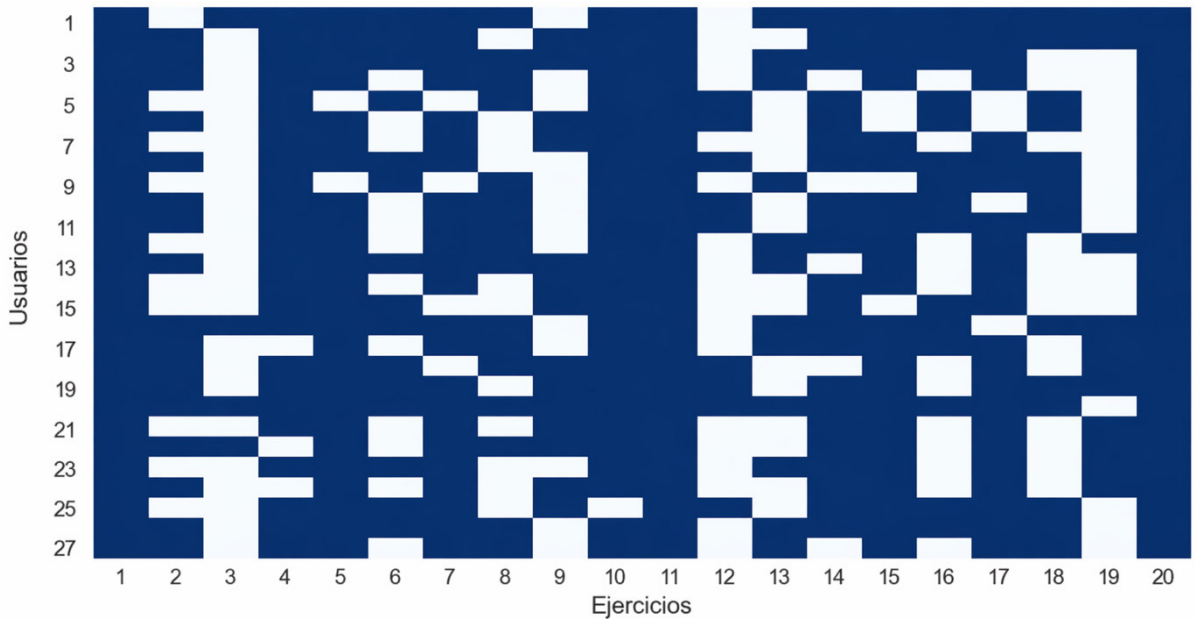


Ilustración 48. Ejercicios resueltos por participantes.

Mientras que la ilustración 49, se muestra los ejercicios completados por los usuarios de la plataforma (color rojo o verde ejercicios realizados, color gris ejercicio sin realizar) así como los resultados de cada ejercicio. En color verde tenemos los ejercicios correctos, mientras que en color rojo los incorrectos. El número dentro de los rectángulos representa los intentos para realizados para cada ejercicio. La gráfica fue realizada en Python tomando en cuenta los resultados de los ejercicios y el número de intentos realizados.

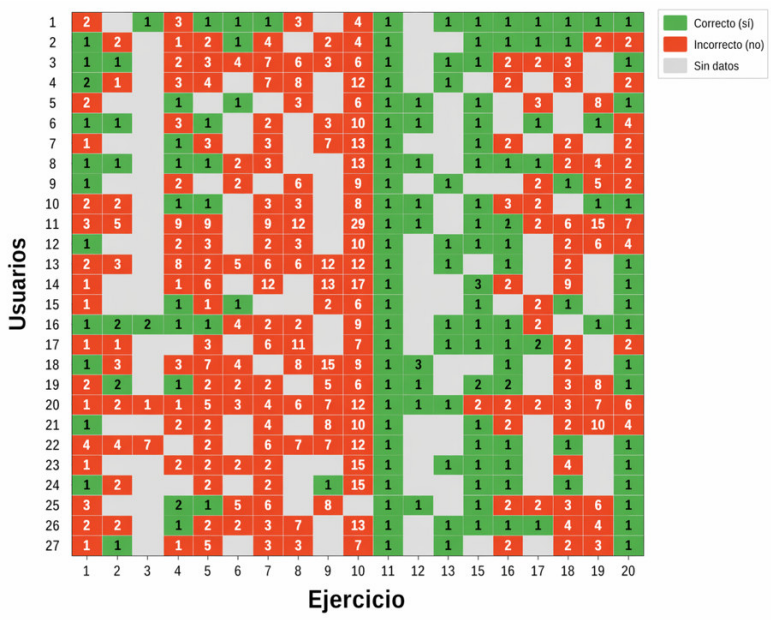


Ilustración 49. Problemas realizados por usuarios de la plataforma.

La ilustración 50, muestra la superficie generada por nuestro sistema difuso. En el cual se modela la relación entre el número de intentos, el error y la acción recomendada. La acción recomendada toma valores de entre 0 y 1, donde los valores más cercanos a 1 recomiendan pasar al ejercicio siguiente, mientras que los valores más cercanos a 0 sugieren repetir el ejercicio y los valores intermedios recomienda los ejercicios de refuerzo. El punto color verde representa el valor máximo de acción con un valor de 0.96, mientras que el punto rojo nos muestra el valor mínimo de acción con un valor de 0.33.

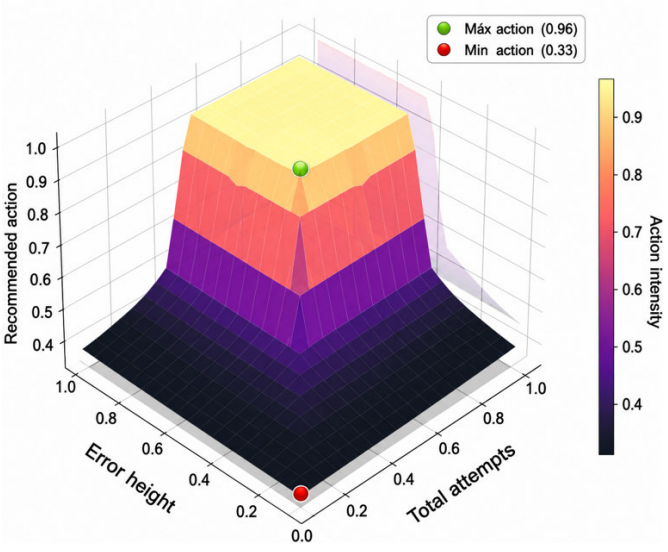


Ilustración 50. Superficie Difusa de la relación entre intentos, error y acción.

Cuando el número de intentos es bajo y el error también muestra un valor bajo, la acción recomendada alcanza su valor máximo (0.96). Esto implica que el sistema considera que el estudiante ha comprendido adecuadamente el ejercicio y puede avanzar a uno de mayor dificultad. Por el contrario, cuando hay más intentos, incluso si el error es bajo, la acción recomendada disminuye significativamente. Esto nos indica que, aunque el estudiante resolvió el ejercicio correctamente, necesitó múltiples intentos, lo que sugiere que aún no ha dominado completamente el concepto y, por lo tanto, debería realizar un ejercicio de refuerzo.

La región más obscura del gráfico, con una acción cercana a 0.0, corresponde a situaciones con muchos intentos y errores altos, donde el sistema decide no avanzar y da como recomendación repetir el mismo ejercicio.

Por otro lado, en la ilustración 51 se presentan los resultados de nuestro sistema difuso. Dicho sistema, genera rutas de aprendizaje personalizadas para cada uno de nuestros usuarios. Por ejemplo, para nuestro usuario 27 nuestro sistema recomienda realizar el ejercicio de refuerzo 1 ya que aún no cuenta las habilidades necesarias para tomar un ejercicio más avanzado, para el ejercicio 2, sugiere pasar al ejercicio 4, y para el ejercicio 4 recomienda realizar los ejercicios de refuerzo 3 y 4. Y así, sigue generando las rutas de aprendizaje para nuestro usuario 27 y el resto de ellos.

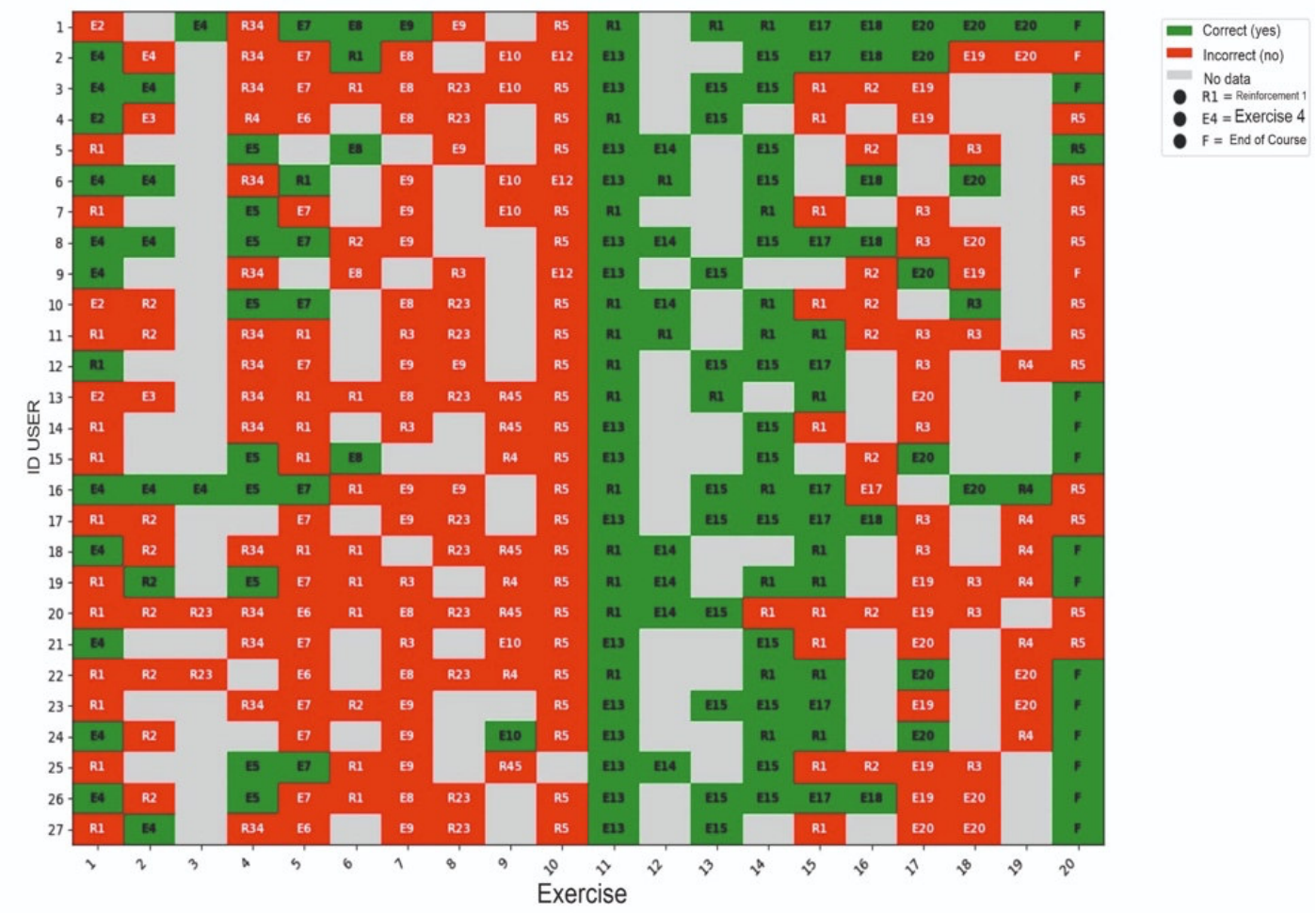


Ilustración 51. generación de las rutas de aprendizaje por usuario.

Los resultados obtenidos cumplen con el objetivo principal de nuestro sistema difuso. El cual busca adaptarse al ritmo de cada estudiante priorizando un aprendizaje significativo antes de avanzar a ejercicios más complejos. Esto da como resultado el desarrolló gradual del pensamiento computación en cada usuario a medida que avanza en cada uno de los ejercicios de programación.

Capítulo 6

6. Conclusiones y Trabajo Futuro

La programación fomenta el desarrollo del pensamiento computacional en los niños, razón por la cual muchos países alrededor del mundo han optado por incluirla en sus currículos educativos. Sin embargo, surgen dudas sobre las formas más efectivas de su enseñanza. Por lo anterior, varios investigadores han utilizado plataformas de programación como: Scratch y CODE para la realización de sus estudios. Estos trabajos tienen como resultados: las predicciones sobre el siguiente ejercicio o las categorías recomendadas en Scratch para comenzar a aprender programación.

Para responder la pregunta de investigación ¿Existen conjuntos de datos públicos que contengan datos de la interacción para ser usados en este proyecto? Se hizo un estudio y análisis de la literatura y como resultado se obtuvo que no se tiene un conjunto de datos públicos que contengan interacciones o información acerca de los datos recaudados por la plataforma.

Para la pregunta ¿Qué datos de la interacción de los niños con una plataforma de programación a bloques pueden usarse para la predicción de rutas de aprendizaje? Se hizo un análisis de la literatura y de los datos que se habían utilizado anteriormente y como respuesta se obtuvo que el tiempo de realización de los ejercicios, el número de intentos para cada uno de ellos, la altura de la solución del usuario, la altura de error y si el ejercicio fue correcto o no, son datos que se pueden usar para la predicción de rutas de aprendizaje.

Para la última pregunta ¿Qué herramientas de aprendizaje automático e inteligencia artificial se pueden usar para predecir rutas de aprendizaje en ejercicios de programación para niños? Se realizó también un análisis de la literatura y de las herramientas que mencionan diferentes autores y se concluyó que por el tipo de datos que se tenían la lógica difusa era la herramienta ideal para la generación de rutas de aprendizaje.

Con las preguntas y respuestas anteriores se formuló el objetivo general de esta investigación, el cual fue: predecir rutas de aprendizaje de programación utilizando la lógica difusa y los datos de las

interacciones de los niños con la herramienta de programación, para que la experiencia de aprendizaje se ajuste a su experiencia de uso y desempeño, el cual se logró de manera total ya que con los datos recolectados y el sistema difuso se lograron encontrar los ejercicios adecuados para cada niño según su nivel de programación.

Para lograr el objetivo general se realizaron los objetivos específicos, como el diseño de la herramienta de programación, la recolección de las interacciones de los usuarios como el tiempo de realización de los ejercicios, el número de intentos para cada uno de ellos, la altura de la solución del usuario, la altura de error y si el ejercicio fue correcto o no.

Se analizaron los datos anteriores dentro de un sistema difuso. Este sistema toma estos datos como características de los ejercicios completados por los usuarios y devuelve como salida el siguiente ejercicio a resolver. Con ayuda del sistema difuso se pudieron generar rutas de aprendizaje personalizadas ,ya que de acuerdo con la solución generada por el usuario en cada uno de los ejercicios el sistema le recomendará el ejercicio idóneo para el, lo que permite que su experiencia de aprendizaje se ajuste a su desempeño a lo largo de los ejercicios.

Por lo anterior se logró comprobar la hipótesis principal: Los datos de interacción de los niños con la plataforma de programación si ayudan a la predicción de rutas de aprendizaje utilizando lógica difusa. Además de la aceptación de un articulo JCR titulado “Recommending Computational Thinking Learning Paths using Fuzzy Logic”.

Las limitantes de esta investigación son:

1. aunque se obtuvo una buena cantidad de datos para analizar se necesitan un conjunto mayor para validar aún más este experimento y probar que el sistema difuso funciona bien con una mayor cantidad de datos.
2. Otra limitante es que la plataforma solo esta alojada en una computadora, se necesita subir a un servidor para que más gente pueda acceder a ella y la falta de infraestructura en escuelas no sea un problema, con esto se puede asegurar una mayor recolección de datos en un menor tiempo.
3. El número de bloques de código, para esta investigación se utilizaron bloques de código sencillos (movimiento, giro y repetir hasta) lo que permite medir el desempeño de usuarios

de nivel principiante, se necesitan agregar bloques de código más desafiantes que permitan a los usuarios de un nivel avanzado medir sus conocimientos de programación y su pensamiento computacional, además con estos bloques de código permitirían a los usuarios de nivel básico comenzar a subir a un nivel más avanzado.

Como trabajo a futuro se planea resolver las limitantes anteriores, subir la herramienta a un servidor para que más usuarios puedan acceder a ella y así obtener más datos que permitan mejorar el modelo difuso. Además de agregar nuevos bloques de código y ejercicios de programación que permitan mejorar el pensamiento computacional y los fundamentos de programación. También se implementarían los ejercicios de refuerzo en la plataforma, con esto se podría probar las rutas de aprendizaje generadas por el sistema para verificar su nivel acierto.

También se implementarían otros métodos de análisis como las redes neuronales o los algoritmos genéticos, los cuales ayudarían a comparar los resultados de estos métodos con los resultados de esta investigación, con esto se podría descubrir qué método predice mejor las rutas de aprendizaje.

7. Referencias

[1] M. Guzdial, “Education: Paving the way for computational thinking,” Aug. 01, 2008. doi: 10.1145/1378704.1378713.

[2] J. M. Wing, “Computational thinking and thinking about computing,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 366, no. 1881, pp. 3717–3725, Oct. 2008, doi: 10.1098/rsta.2008.0118.

[3] J. M. Wing, “Computational Thinking,” *Commun. ACM*, vol. 49, pp. 33–35, Mar. 2006.

[4] F. Kalelioglu, Y. Gulbahar, and D. Doğan, “Curriculum Integration Ideas for Improving the Computational Thinking Skills of Learners through Programming via Scratch.” 7th International Conference on Informatics in Schools: Situation, Evolution and Perspectives [Online]. Available: https://www.researchgate.net/publication/280942805

[5] A. Balanskat and K. Engelhardt, “Computing our future Computer programming and coding Priorities, school curricula and initiatives across Europe,” 2015. [Online]. Available: http://creativecommons.org/licenses/by-sa/3.0/

[6] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, “Habits of Programming in Scratch,” 2011. [Online]. Available: http://web.mit.edu/~axch/www/programming_habits.html

[7] F. Kalelioɣlu, “A new way of teaching programming skills to K-12 students: Code.org,” *Comput. Human Behav.*, vol. 52, pp. 200–210, Jul. 2015, doi: 10.1016/j.chb.2015.05.047.

[8] T. Saito and Y. Watanobe, “Learning path recommendation system for programming education based on neural networks,” *International Journal of Distance Education Technologies*, vol. 18, no. 1, pp. 36–64, Jan. 2020, doi: 10.4018/IJDET.2020010103.

[9] A. Repenning, “Agentsheets: A Tool for Building Domain-Oriented Visual Programming Environments (Demonstration),” Colorado, Apr. 1993.

[10] M. Guenaga, A. Eguíluz, P. Garaizar, and J. Gibaja, “How do students develop computational thinking? Assessing early programmers in a maze-based online game,” *Computer Science Education*, vol. 31, no. 2, pp. 259–289, Apr. 2021, doi: 10.1080/08993408.2021.1903248.

[11] S. Grover, M. Bienkowski, J. Niekrasz, and M. Hauswirth, “Assessing problem-solving process at scale,” in *L@S 2016 - Proceedings of the 3rd 2016 ACM Conference on Learning at Scale*, Edinburgh, Scotland UK: Association for Computing Machinery, Inc, Apr. 2016, pp. 245–248. doi: 10.1145/2876034.2893425.

[12] S. Grover and S. Basu, “Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and Boolean logic,” in *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*, Association for Computing Machinery, Mar. 2017, pp. 267–272. doi: 10.1145/3017680.3017723.

[13] Robles Gregorio, Moreno-León Jesús, Aivaloglou Efthimia, and Hermans Felienne, *Software Clones in Scratch Projects: On the Presence of Copy-and-Paste in Computational Thinking Learninng*. klagenfurt, Austria, 2017.

[14] N. A. Escherle *et al.*, “Piloting computer science education week in Mexico,” in *SIGCSE 2016 - Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, Association for Computing Machinery, Inc, Feb. 2016, pp. 431–436. doi: 10.1145/2839509.2844598.

[15] D. Brent, “El concepto del Banco Mundial de la participación en el desarrollo y la gobernanza de la educación: un análisis de su acercamiento y resultados.,” Distrito Federal, México, 2014.

[16] J. Mendoza González, “El Rezago Educativo. Un Problema de Construcción Social,” *A&H*, pp. 44–57, Nov. 2019, [Online]. Available: www.upaep.mx/revistaayh

[17] Barr Valerie and Stephenson Chris, “Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community?,” *acm inroads*, vol. 1, no. 1, pp. 48–54, Mar. 2011, doi: 10.1145/1929887.1929905 ©.

[18] R. S. N. Lindberg, T. H. Laine, and L. Haaranen, “Gamifying programming education in K-12: A review of programming curricula in seven countries and programming games,” *British Journal of Educational Technology*, 2018, doi: 10.1111/bjet.12685.

[19] A. Vee, “Understanding Computer Programming as a Literacy,” Pittsburgh, Oct. 1999.

[20] J. Moreno-León, G. Robles, and M. Román-González, “Towards Data-Driven Learning Paths to Develop Computational Thinking with Scratch,” *IEEE Trans. Emerg. Top. Comput.*, vol. 8, no. 1, pp. 193–205, Jan. 2020, doi: 10.1109/TETC.2017.2734818.

[21] S. Atmatzidou and S. Demetriadis, “Advancing students’ computational thinking skills through educational robotics: A study on age and gender relevant differences,” *Rob. Auton. Syst.*, vol. 75, pp. 661–670, Jan. 2016, doi: 10.1016/j.robot.2015.10.008.

[22] S. Y. Lye and J. H. L. Koh, “Review on teaching and learning of computational thinking through programming: What is next for K-12?,” 2014, *Elsevier Ltd*. doi: 10.1016/j.chb.2014.09.012.

[23] D. Pérez-Marín, R. Hijón-Neira, A. Bacelo, and C. Pizarro, “Can computational thinking be improved by using a methodology based on metaphors and scratch to teach computer programming to children?,” *Comput. Human Behav.*, vol. 105, Apr. 2020, doi: 10.1016/j.chb.2018.12.027.

[24] F. J. García-Peñalvo and A. J. Mendes, “Exploring the computational thinking effects in pre-university education,” Mar. 01, 2018, *Elsevier Ltd*. doi: 10.1016/j.chb.2017.12.005.

[25] I. M. Ramos, D. B. Ramos, B. F. Gadelha, and E. H. T. De Oliveira, “An Approach to Group Formation in Collaborative Learning Using Learning Paths in Learning Management Systems,” *IEEE Transactions on Learning Technologies*, vol. 14, no. 5, pp. 555–567, Oct. 2021, doi: 10.1109/TLT.2021.3117916.

[26] Y. Zhang and M. Funk, “The Four Steps to Creative Programming with the Processing Language Coding Art.” [Online]. Available: https://www.springer.com/

[27] M. Rouhani, “Programming for Teachers: Supporting Participants in Defining Their Learning Path in a Flexible Online Learning Trajectory Course,” *International Journal of Childhood Education*, vol. 1, no. 1, pp. 13–23, Dec. 2020, doi: 10.33422/ijce.v1i1.10.

[28] Y. Higo, A. Ohtani, and S. Kusumoto, “Generating Simpler AST Edit Scripts by Considering Copy-and-Paste,” 2017.

[29] J. R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, “Fine-grained and accurate source code differencing,” in *ASE 2014 - Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, Association for Computing Machinery, Inc, 2014, pp. 313–323. doi: 10.1145/2642937.2642982.

[30] U. D«, A. V Aho, M. S. Lam, R. Sethi, J. D. Ullman, and W. Ji, “Compiladores principios, técnicas y herramientas Segunda edición.”

[31] M. M. Burnett, “Visual Programming,” John Wiley & Sons Inc, 1999.

[32] P. P. Ray, “A Survey on Visual Programming Languages in Internet of Things,” 2017, *Hindawi Limited*. doi: 10.1155/2017/1231430.

[33] C. H. Knight, “Intelligent, artificially,” May 01, 2024, *Cambridge University Press*. doi: 10.1017/S0022029924000499.

[34] L. D. Hollebeek, C. Menidjel, M. Sarstedt, J. Jansson, and S. Urbonavicius, “Engaging consumers through artificially intelligent technologies: Systematic review, conceptual model, and further research,” Apr. 01, 2024, *John Wiley and Sons Inc*. doi: 10.1002/mar.21957.

[35] J. H. Fetzer, “What is Artificial Intelligence,” *Artificial Intelligence: its Scope and limits*, vol. 1, pp. 3–27, 1990.

[36] R. M. Granados, “Modelos de regresión lineal múltiple.”, Granda, España, (2016), pp. 1-61

[37] C. Laguna, “CORRELACIÓN Y REGRESIÓN LINEAL”, instituto Aragonés de Ciencias y Salud (2014).

[38] R. E. Barrientos Martínez, N. Cruz Ramírez, and H. G. Acosta Mesa, “Árboles de decisión como herramienta en el diagnóstico médico,” Sep. 2009. [Online]. Available: www.uv.mx/rm

[39] P. A. Cardona Hernández, “Aplicación de árboles de decisión en modelos de riesgo crediticio,”, Revista Colombiana de estadistica Dec. 2004.

[40] Klir J. George, “fuzzy logic Unearthing its meaning and significance,” Binghamton, Sep. (1995).

[41] J. M. Mendel, “Fuzzy Logic Systems for Engineering: A fitorial.”, congreso IEEE, Oct (1995).

[42] A. Vee, “Understanding Computer Programming as a Literacy,” Pittsburgh, Oct. 1999.

[43] M. Pankiewicz, Y. Shi, and R. S. Baker, “srcML-DKT: Enhancing Deep Knowledge Tracing with Robust Code Representations from srcML,” in *Proceedings of the International Conference on Educational Data Mining*, Palermo, Italia: International Educational Data Mining Society, 2025, pp. 541–548. doi: 10.5281/zenodo.15870306.

[44] M. Hoq, P. Brusilovsky, and B. Akram, “Explaining Explainability: Early Performance Prediction with Student Programming Pattern Profiling,” 2024.

[45] M. Zhu, S. Han, P. Yuan, and X. Lu, “Enhancing Programming Knowledge Tracing by Interacting Programming Skills and Student Code,” in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Mar. 2022, pp. 438–443. doi: 10.1145/3506860.3506870.

[46] T. Jin, L. Dou, G. Yang, A. Zhou, X. Zhu, and C. Huang, “Dynamics-Aware Gated Graph Attention Neural Network for Student Program Classification and Knowledge Tracing,” 13th ed., ICEKIM, Ed., 2023, pp. 1640–1652. doi: 10.2991/978-94-6463-172-2_182.

[47] B. Jiang, Y. Ye, and H. Zhang, “Knowledge Tracing Within Single Programming Exercise Using Process Data,” Philippines, 2018.

[48] B. Jiang, S. Wu, C. Yin, and H. Zhang, “Knowledge Tracing within Single Programming Practice Using Problem-Solving Process Data,” *IEEE Transactions on Learning Technologies*, vol. 13, no. 4, pp. 822–832, Oct. 2020, doi: 10.1109/TLT.2020.3032980.