UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

MateFun Infantil

Aplicación Android

Pedro Nicolás Chiaramello
Ana Lucía Etchart
Maximiliano Poses

Tutores: Gonzalo Tejera y Marcos Viera

Proyecto de Grado de Ingeniería en Computación
Instituto de Computación, Facultad de Ingeniería
Universidad de la República

24 de diciembre de 2020

Agradecimientos

A los tutores Marcos y Gonzalo, quienes nos brindaron su tiempo, conocimiento y apoyo durante todo el proyecto.

A nuestras familias y amigos, por su apoyo incondicional a lo largo de la carrera y en particular, durante la realización de este proyecto.

RESUMEN

MateFun es un lenguaje funcional creado por docentes del Instituto de Computación de la Facultad de Ingeniería, UdelaR. En sus inicios, fue diseñado con el objetivo de introducir la programación a estudiantes liceales y a su vez, fortalecer la apropiación del concepto de función matemática.

Tomando como base el intérprete (aplicación Haskell que implementa el lenguaje MateFun), varios proyectos de grado han extendido MateFun incorporando diferentes funcionalidades, tales como un IDE Web y la capacidad de graficar figuras y funciones.

En este contexto, surge el presente proyecto donde se aborda el estudio y realización de un lenguaje de programación visual. El mismo adapta MateFun para ser utilizado por niños en educación primaria, por medio de una aplicación Android.

Como resultado del análisis de diversos lenguajes visuales utilizados con fines educativos, se optó por utilizar Blockly como base para el desarrollo de MateFun Infantil. Blockly es una biblioteca desarrollada y mantenida por Google, la cual permite crear lenguajes visuales utilizando bloques.

Esta biblioteca fue adaptada para generar código en formato MateFun a partir de bloques, el cual debe ser evaluado en el intérprete. Para lograr esto, fue necesario integrar el intérprete a la aplicación Android.

A partir del código del IDE Web existente, fueron modificados y agregados a la aplicación los componentes encargados de generar gráficos en 2D y 3D.

Por ultimo, a partir de las pruebas realizadas, se concluye que el proyecto cumple con los objetivos planteados. En efecto, fue desarrollada una aplicación Android que permite crear programas utilizando componentes del lenguaje MateFun de forma gráfica, con bloques, y que es factible de ser utilizada en educación primaria por niños. Como parte del trabajo a futuro, se espera que lo anterior sea validado, probando la aplicación con niños y maestros en el aula.

Palabras clave: niños, programación funcional, programación visual, matemática, primaria, Android, bloques.

Lista de figuras

Lista de tablas

Tabla de contenidos

Capítulo 1

Introducción

1.1. Motivación y contexto

El lenguaje MateFun es un lenguaje de programación funcional puro dirigido al aprendizaje de funciones matemáticas, diseñado por investigadores del Instituto de Computación de la Facultad de Ingeniería.

El intérprete de MateFun es una aplicación de consola desarrollada en Haskell. Puede ser accedido a través de un entorno web de programación integrado que permite gestionar programas, programar, ejecutar programas y visualizar gráficas, figuras y animaciones.

En principio, fue diseñado para introducir la programación a estudiantes liceales y a su vez fortalecer la apropiación del concepto de función matemática.

En el Proyecto de Grado realizado por Cameto y Méndez (2017), MateFun fue probado con estudiantes liceales, demostrando ser de gran utilidad para la enseñanza de matemática en ese nivel.

Al día de hoy, también se utiliza en el curso "Taller de Introducción a la Computación" (2020), dictado en Facultad de Ingeniería. En particular, para el proyecto "Programando funciones matemáticas". Este curso está dirigido a estudiantes que recién ingresan a la carrera de Ingeniería en Computación. En el proyecto antes mencionado, se utiliza MateFun para introducir conceptos básicos que serán de utilidad en futuros cursos de programación y que ayudarán a comprender la relación entre matemática y programación.

Si bien se trata de un lenguaje de programación simple, tanto el uso del intérprete por línea de comandos o del entorno web de MateFun, requieren ciertos niveles cognitivos y conocimientos matemáticos.

1.2. Objetivos

El lenguaje MateFun fue diseñado para ser utilizado, en principio, por liceales que se encuentran estudiando el concepto de función. Se trata de un lenguaje textual, con una sintaxis minimal y semejante a la notación utilizada en matemática.

Con el fin de extender el alcance de MateFun, tomando los aciertos de los proyectos anteriores y las experiencias a nivel liceal y universitario, el objetivo de este proyecto es generar una versión de MateFun que pueda ser utilizada por niños en educación primaria. Por lo tanto, será deseable que el lenguaje se pueda asimilar rápidamente y que su relación con los conceptos matemáticos subyacentes pueda ser reconocido por los estudiantes.

En concreto, se espera contar con una aplicación Android que permita programar en una versión reducida de MateFun, utilizando un entorno visual. De esta manera, se intenta generar un entorno de programación más amigable y que pueda ser utilizado por niños en tablets.

1.3. Organización del documento

El presente documento se encuentra dividido en seis capítulos. A su vez, con el fin de presentar de forma ordenada y estructurada la información, cada capítulo se compone de distintas secciones y subsecciones con información específica.

En el Capítulo 2, se resume la investigación realizada al comienzo del proyecto respecto al estado del arte. Se hace énfasis en los distintos lenguajes de programación visual (VPL) existentes, utilizados con fines educativos.

En el Capítulo 3, se aborda el estado inicial de MateFun previo a comenzar el proyecto MateFun Infantil Android. Se estudian los distintos componentes de MateFun, profundizando en los aspectos que resultaron relevantes para el presente proyecto.

El Capítulo 4 es el más extenso. En él, se presentan los detalles respecto al desarrollo realizado en este proyecto. Se fundamentan las decisiones de diseño y arquitectura de la aplicación Android desarrollada, la forma en que fueron realizadas distintas funcionalidades, entre otros temas.

En el Capítulo 5, se presentan las tareas que fueron llevadas a cabo para validar el correcto funcionamiento de la aplicación. Se describen las pruebas

realizadas por el equipo utilizando *testing* exploratorio, la utilización de herramientas para el reporte y corrección de *bugs* y las pruebas no funcionales realizadas sobre la aplicación. Para cada una de estas, se detallan los resultados obtenidos.

Finalmente, en el Capítulo 6, se abordan las conclusiones y trabajo futuro.

Al final del documento se incluye un Glosario. Allí se definen aquellos conceptos utilizados a lo largo de este documento, para los cuales se entiende es necesario aclarar su significado.

El presente documento se complementa con los documentos anexos "Documento de Análisis y Diseño", "Manual de Usuario" y "Manual del Desarrollador". Cuando sea requerido, se hará referencia a ellos.

Capítulo 2

Estado del Arte

Este capítulo está enfocado en relevar y analizar los lenguajes de programación visual existentes (o *VPL* por sus siglas en inglés). Un *VPL* es cualquier lenguaje de programación que permite a los usuarios crear programas manipulando elementos gráficos en lugar de especificarlos textualmente (Jost y col., 2014). En concreto, un *VPL* permite programar con expresiones visuales, arreglos espaciales de texto y símbolos gráficos, utilizados como elementos de sintaxis o notación secundaria (craft.ai, 2015).

Según explica Repenning (2017), los *VPL* sirven para hacer más accesible la programación a quienes se inician en ella. Este tipo de lenguajes permite reducir o incluso eliminar los errores de sintaxis, utilizando íconos, bloques, formularios y diagramas. A su vez, haciendo uso de elementos visuales tales como iconos, formas y colores, se facilita la comprensión de la semántica de los componentes del lenguaje (proporcionan mecanismos para revelar el significado de las primitivas de programación).

El objetivo de este capítulo es conocer el estado actual de los *VPL*. En particular, interesa saber cuáles existen, qué características comparten, sus ventajas y desventajas. Una vez recabada y analizada la información al respecto, se buscará extraer ideas de usabilidad e implementación para la construcción de MateFun Infantil.

MateFun es un lenguaje de programación funcional dirigido al aprendizaje. Debido a esto, el foco de este capítulo estará en los *VPL* utilizados con fines educativos o aquellos que permitan representar un lenguaje de programación funcional.

2.1. Revisión de VPLs existentes

En esta sección, serán abordados los lenguajes de programación visual que resultaron de mayor interés y utilidad para el proyecto. Como fue mencionado, en su mayoría, serán presentados lenguajes de bloques utilizados con fines educativos. El relevamiento y análisis de estos lenguajes sirvió para obtener ideas generales y como base para la toma de decisiones en el diseño de MateFun Infantil.

2.1.1. Scratch

Scratch es un lenguaje de programación imperativo basado en bloques, de código abierto, desarrollado por el Grupo *Lifelong Kindergarten* del *MIT Media Lab*. Su primera versión fue desarrollada en 2003 (Marji, 2014).

La filosofía de Scratch se basa en el intercambio, la combinación y reutilización de código. En Scratch, los usuarios pueden crear sus propios proyectos así como también descargar proyectos públicos subidos y desarrollados por terceros.

El funcionamiento de la herramienta se basa en el uso de objetos provenientes de una biblioteca con objetos creados por el usuario o importados de otro proyecto. A tales objetos se les aplica bloques de instrucciones arrastrándolos desde la paleta de bloques hacia un área que contiene todos los pasos asociados con el objeto (Resnick y col., 2009).

En la Figura 2.1 se muestra un proyecto desarrollado con Scratch.

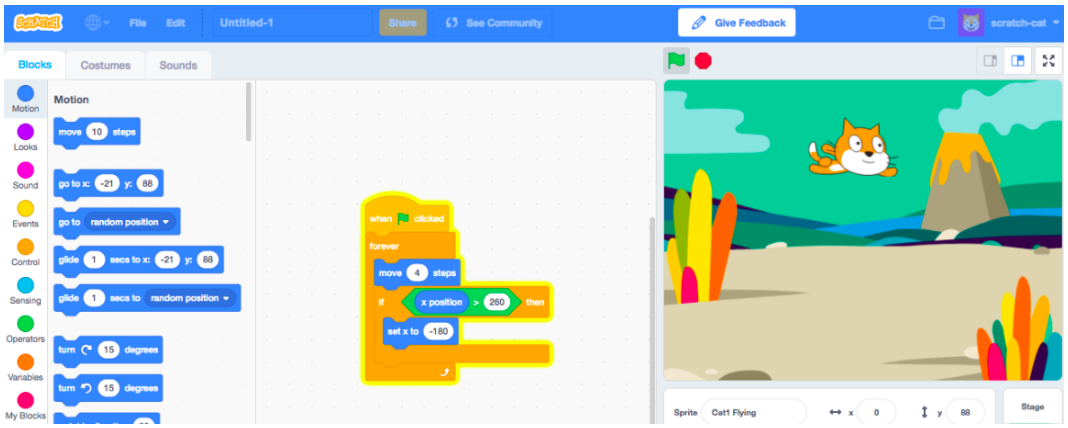

Figura 2.1: Interfaz de usuario de Scratch (ElObjetivo, 2019).

En la actualidad Scratch se encuentra en la versión 3.0 la cual se basa en HTML5, dejando de lado Adobe Flash utilizado en versiones anteriores. Además, la implementación utiliza una *Virtual Machine* (VM) que construye un árbol de sintaxis abstracto (MIT, s.f.). Esto permite que sea compatible con una gran cantidad de dispositivos.

Scratch utiliza tecnologías como Java, JavaScript, Node.js y GL Shader, entre otras.

2.1.2. Blockly

Blockly es una biblioteca gratuita y de código abierto creada por Google en 2002 bajo licencia Apache 2.0. Está disponible en versiones Web (basadas en JavaScript) y como librería nativa para Android e iOS.

Desde la perspectiva del usuario, Blockly es una forma visual e intuitiva de construir código (programando con bloques). Desde la perspectiva del desarrollador, Blockly es una interfaz de usuario lista para usar para crear un lenguaje visual que genera código sintácticamente correcto ("Blockly Overview", 2020).

Blockly tiene la capacidad de generar código (a partir de los bloques) en diferentes lenguajes, tales como JavaScript, Lua, Dart, Python o PHP. Permite además personalizar la herramienta para poder generar código en cualquier lenguaje de programación contextual ("Blockly Reference", s.f.).

La biblioteca incluye un conjunto de bloques predefinidos para operaciones comunes y dada su flexibilidad, permite a los desarrolladores personalizarlo para agregar nuevos bloques. Los nuevos bloques requieren una definición de bloque y un generador de código. La definición describe la apariencia del bloque (en la interfaz de usuario), mientras que el generador realiza la traducción del bloque al código ejecutable ("Blockly Generator", s.f.). Blockly permite escribir manualmente las definiciones de los bloques y el código retornado por los mismos. No obstante, posee la herramienta web "Blockly Factory" (s.f.), la cual permite definir de manera gráfica e intuitiva los bloques.

Otra característica útil es la verificación básica de tipos, de modo que los usuarios no tengan la posibilidad de conectar bloques que carezcan de sentido. De este modo, se evita que el usuario genere un error al ejecutar el programa.

En la Figura 2.2 se muestra la interfaz de usuario por defecto de Blockly.

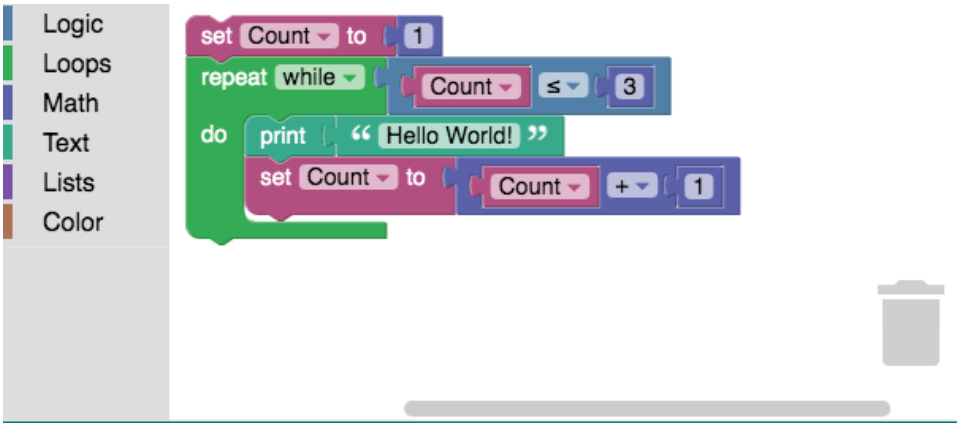

Figura 2.2: Interfaz de usuario de Blockly (“Blockly Reference”, s.f.).

2.1.3. Viskell

Viskell es un entorno de programación visual experimental para un lenguaje de programación funcional tipado (similar a Haskell). Este proyecto explora las posibilidades y desafíos de la programación visual interactiva en combinación con las fortalezas y debilidades de los lenguajes funcionales (Boeijink, 2016).

Como es mencionado en el “Repositorio en GitHub de Viskell” (2017), dentro de los objetivos de este proyecto se encuentran, entre otros, la creación de visualizaciones legibles y compactas para construcciones del lenguaje funcional, el desarrollo guiado por tipos (los fragmentos del programa muestran sus tipos y el error de tipo se visualiza localmente), entre otros.

Este lenguaje tiene algunas diferencias con los anteriores: el programa está diseñado en forma vertical, los bloques no tienen íconos (en su lugar se utilizan nombres), y se utilizan “puertos” para unir los bloques, todos con tipos especificados.

Viskell se encuentra implementado utilizando JavaFX, GHC y QuickCheck.

En la Figura 2.3 se muestran algunos programas realizados utilizando la interfaz de Viskell.

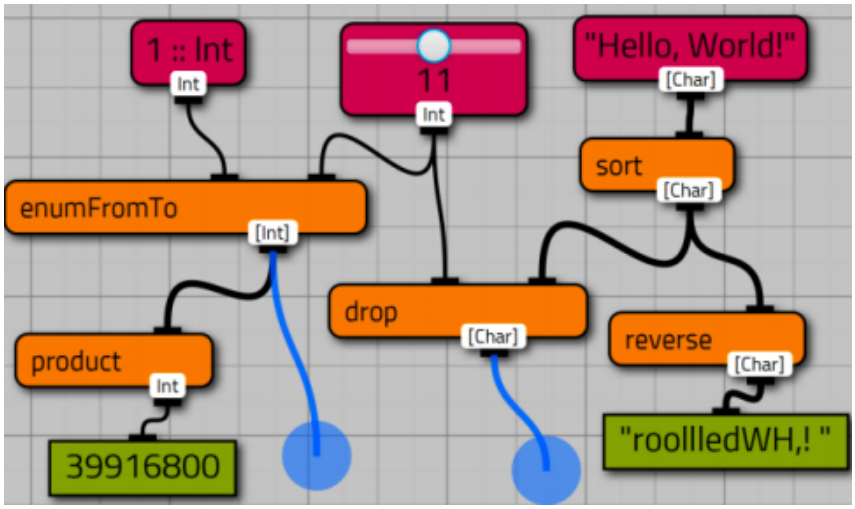

Figura 2.3: Interfaz de usuario de Viskell ("Repositorio en GitHub de Viskell", 2017)

2.1.4. Snap!

Snap! es un lenguaje de programación gráfica con fines educativos inspirado en Scratch, con algunas características avanzadas. Fue creado en 2011 por Jens Mönig y Brian Harvey de la Universidad de Berkeley (Harvey, s.f.). El editor de Snap! y los programas creados en él, son aplicaciones web que se ejecutan en el navegador sin necesidad de realizar instalación. El código fuente tiene licencia AGPL, está disponible en el "Repositorio en GitHub de Snap" (2019) y puede ser descargado y modificado con fines no comerciales.

En la Figura 2.4 se muestra su interfaz.

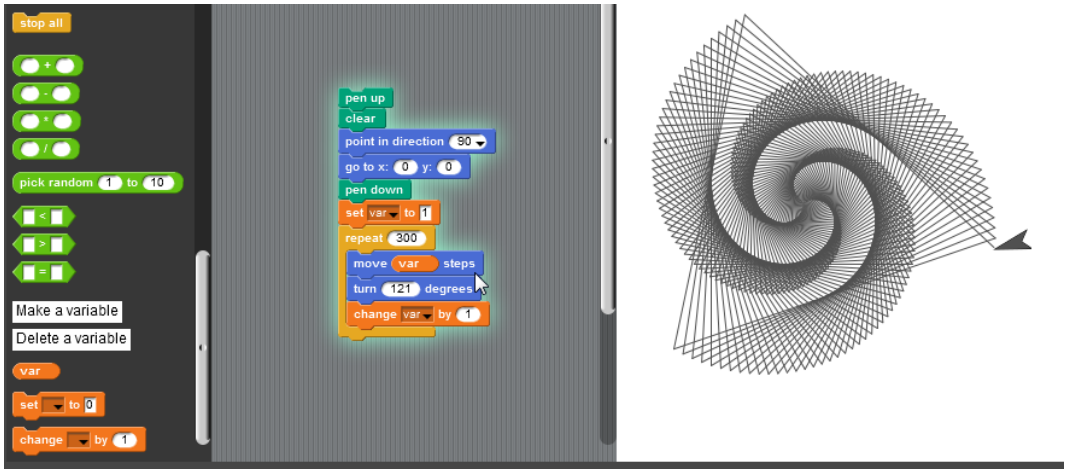

Figura 2.4: Interfaz de usuario de Snap! (Mönig y Harvey, 2020)

Snap! está implementado en JavaScript usando HTML5 Canvas APIs.

Presenta diferencias con Scratch, agregando funcionalidades avanzadas tales como funciones lambda, posibilidad de generar código a Python, C, entre otras.

2.1.5. Turtle Blocks

Sugar es una plataforma de software educativo ("Sugar Labs", s.f.). En Sugar, una *activity* es una aplicación que puede ser ejecutada en dicho *software*.

"Turtle Blocks" (2014) surge en este contexto como una actividad Sugar, inspirada en Logo (LogoFoundation, 2014), la cual pone al alcance de niños conceptos de programación mediante una interfaz gráfica icónica, donde cada instrucción se mapea como un bloque. El objetivo que persigue es utilizar dichos bloques para controlar los movimientos de una tortuga y, por ejemplo, lograr dibujar figuras a partir de estos movimientos.

Actualmente existen dos versiones de Turtle Blocks, una implementada en Python y otra en JavaScript.

En la Figura 2.5 se presenta la interfaz de Turtle Blocks.

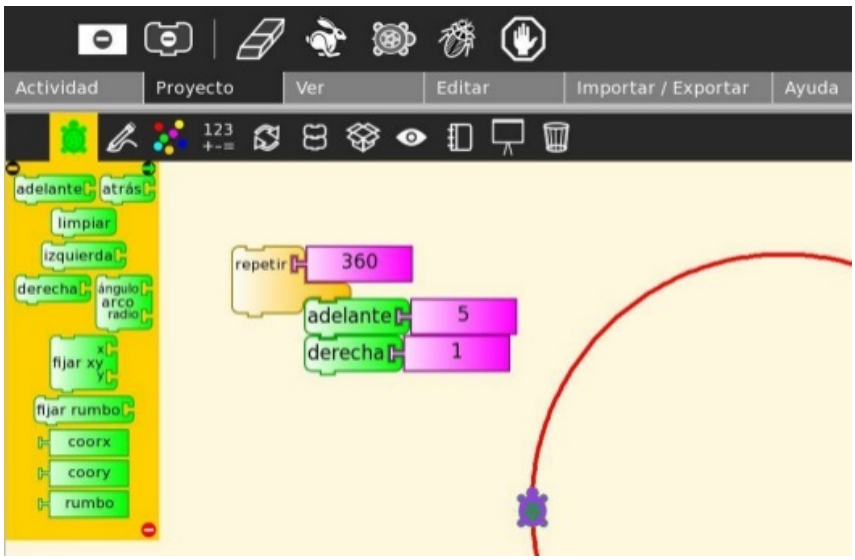

Figura 2.5: Interfaz de usuario de "Turtle Blocks" (2014)

El "Proyecto Butiá" (2016), llevado adelante en la Facultad de Ingeniería desde el año 2009, utiliza Turtle Blocks. Agregando *plugins* en forma de paletas que permiten controlar diferentes kits robóticos como Butiá, Lego NXT, WeDo, Fischer LT, FollowMe y Sumbot, crearon "TortuBots" (2019).

2.1.6. Pilas Bloques

Pilas Bloques es una aplicación orientada a niños en edad escolar, con el fin de incorporar la programación en el aula. En dicha aplicación se proponen desafíos con diversos niveles de dificultad, con el fin de acercar a los estudiantes a la programación por medio de la utilización de bloques.

La plataforma fue pensada para acompañar una secuencia didáctica para aprender a programar en la escuela. Una secuencia didáctica es la planificación mediante la cual se propone aprender un tema determinado. La secuencia didáctica incluida en Pilas Bloques fue ideada y probada por docentes e investigadores argentinos ("Web de Pilas Bloques", 2019).

En la Figura 2.6 se presenta la interfaz de Pilas Bloques.

Figura 2.6: Interfaz de usuario de Pilas Bloques ("Web de Pilas Bloques", 2019)

Como puede observarse en el "Repositorio en GitHub de Pilas Bloques" (2019), la aplicación se encuentra implementada en Python y Node.js. Además, utiliza Blockly para lograr su interfaz.

2.1.7. Polyup

"PolyUp" (2020) es una plataforma web que cuenta además con aplicaciones iOS y Android, la cual ofrece a estudiantes desafíos matemáticos gamificados. Utiliza como guía a un personaje llamado Poly, el cual presenta desafíos de cálculo llamados *Poly Machines*. Los estudiantes pueden utilizar la aplica-

ción para aprender, desde operaciones básicas como sumar, restar, multiplicar y dividir hasta secuencias y series más complejas como la serie de Fibonacci.

Polyup utiliza un modelo de arriba hacia abajo, o notación polaca inversa (*RPN*, por su nombre en inglés "Reverse Polish Notation") para hacer cálculos.

Si bien es subjetivo, tal como puede observarse en la Figura 2.7, la aplicación presenta una interfaz bien lograda. Debido a su notación, se alinea a un lenguaje funcional como lo es MateFun.

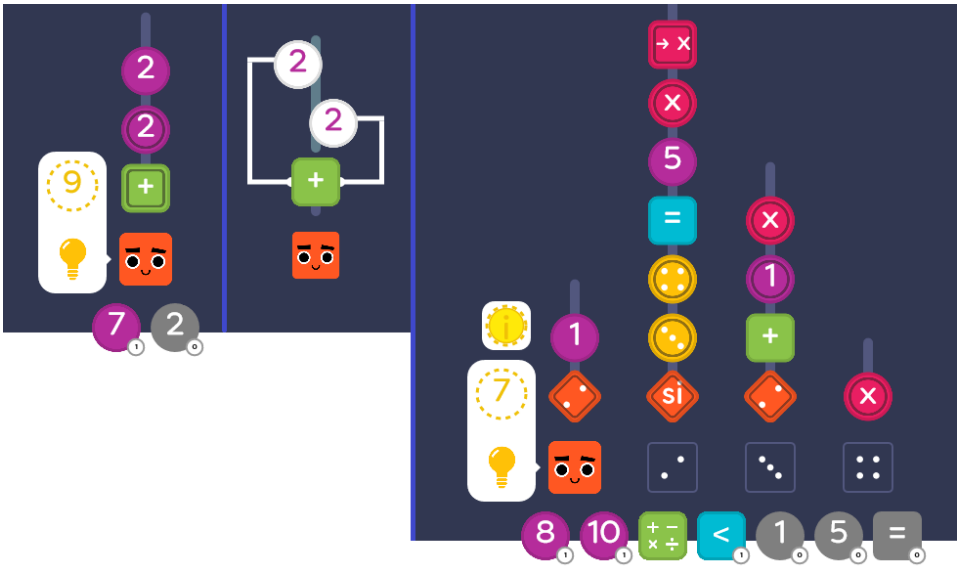

Figura 2.7: Interfaz de usuario de "PolyUp" (2020)

2.1.8. ScratchJr

ScratchJr está pensado para que niños de entre 5 y 7 años puedan programar historias interactivas y juegos. La idea es que en el proceso puedan aprender a resolver problemas, diseñar proyectos y expresar su creatividad programando ("Acerca de ScratchJr", s.f.).

En ScratchJr, un programa se crea arrastrando bloques en una zona de codificación y se encastran juntos. En particular, los bloques de ScratchJr están completamente basados en íconos (sin texto). De esta forma, los niños pueden utilizar este lenguaje antes de saber leer. Los bloques están conectados de izquierda a derecha, de igual manera a como se escriben las palabras (CBSNews, 2014).

La interfaz de usuario es mucho más simple en comparación con Scratch u otros similares. Además, las categorías de bloques de programación y el

número de bloques dentro de cada categoría son reducidos, apostando a hacer la aplicación lo más básica posible sin perder funcionalidad.

Como se menciona en el "Repositorio en GitHub de ScratchJr" (2019), el sistema está inspirado en Scratch y disponible como aplicación nativa para tablets con sistema operativo Android e iOS, así como también para Chromebook. A su vez, ScratchJr fue desarrollado por el MIT, está basado en Blockly y utiliza además tecnologías tales como JavaScript, Java y Objective-C.

En la Figura 2.8 se puede observar la interfaz de usuario de ScratchJr.

Figura 2.8: Interfaz de usuario de ScratchJr ("Acerca de ScratchJr", s.f.)

2.2. Análisis de VPLs estudiados

2.2.1. Características deseables

Al analizar los lenguajes y aplicaciones presentados en la sección Revisión de VPLs existentes, se observaron características en común, deseables para una aplicación como MateFun Infantil:

- Las imágenes o texto empleados deben entenderse fácilmente por el público objetivo.

- La forma en que se crea el programa con los componentes visuales debe ser comprensible rápidamente.
- Los bloques disponibles pueden ser mostrados por categorías, diferenciando por color y forma.
- Algunas cuentan con un personaje que guía la secuencia didáctica y propone retos.
- En caso de errores en los programas construidos con el lenguaje visual, tienen mecanismos de mensajes o formas de ilustrar los errores sobre los componentes visuales para así comprender mejor el error.

2.2.2. Comparación entre lenguajes estudiados

A continuación se presenta una tabla comparativa de algunas características de los distintos *VPL* estudiados.

	Licencia	Sistema Operativo	Tecnologías
Scratch	GPLv2	Windows/MacOS/Linux	JavaScript/HTML/Python/PHP/Java
Blockly	Apache 2.0	Android/iOS/Windows/ MacOS/Linux	JavaScript/HTML5/SVG
Viskell	MIT	Cross-Platform	Java 8/JavaFX
Snap!	AGPLv3	Cross-Platform	JavaScript/HTML5 Canvas API
TurtleBlocks	GNU	Cross-Platform	JavaScript/HTML5
Pilas Bloques	GNUv3	Windows/Linux/MacOS	Electron/JavaScript/NodeJS/Blockly
Polyup	N/A	Cross-Platform	N/A
Scratch Jr.	BSD-3	iOS, Android, Chromebook	HTML5/JavaScript/Java/Objective-C

Tabla 2.1: Comparación de lenguajes de bloques presentados en Revisión de VPLs existentes.

En base a estas características y considerando las cualidades deseables a partir del análisis anterior, se optó por uno de estos lenguajes visuales para integrar a MateFun Infantil, tal como se explica en las Opciones de implementación.

Fueron decisivos aspectos de licenciamiento, sistema operativo en el que podían ejecutarse y tecnologías en que se implementaba (debía poder integrarse a la aplicación Android).

2.2.3. Opciones de implementación

Luego del estudio realizado, considerando las buenas prácticas en lo que refiere a lenguajes visuales con fines educativos y orientados a lenguajes funcionales; fueron evaluadas tres opciones para implementar el proyecto MateFun Infantil:

1. Implementar un lenguaje visual desde el comienzo, sin utilizar otros existentes.

2. Modificar una aplicación de código abierto existente, adaptando su implementación a los requerimientos del proyecto.

3. Hacer uso de algún *framework*, como lo es Blockly, que brinde herramientas para crear un lenguaje de programación visual basado en bloques. Es deseable además, utilizarlo en conjunto con otras tecnologías que permitan completar el desarrollo de la aplicación MateFun Infantil para dispositivos Android.

Desarrollar una interfaz visual basada en bloques e implementar la unión de bloques, manejo de errores sintácticos, generación de código, entre otros; implica emplear esfuerzo del proyecto que podría ser destinado al desarrollo de otras funcionalidades.

La usabilidad de la aplicación a construir es muy importante. Los estudiantes deben poder familiarizarse con ella rápidamente y debe resultar natural su uso para la notación matemática que suelen utilizar. Aplicaciones existentes y bibliotecas como Blockly, solucionan parcialmente problemas de usabilidad y diseño, que podrían ocurrir en caso de no utilizarlas.

Utilizar software ya existente y modificarlo trae problemas, como pueden ser: el tiempo a emplear para entender su funcionamiento, código fuente y depender de las tecnologías utilizadas en su implementación, las cuales podrían no funcionar en Android o hacer más difícil la implementación de MateFun Infantil.

Blockly es una biblioteca que soluciona varios de los problemas de implementación del lenguaje visual y es relativamente fácil de integrar a un desarrollo propio. Se podría generar código en el lenguaje de MateFun definiendo los elementos necesarios en Blockly. Los retos restantes incluyen adaptarlo a un lenguaje funcional como lo es MateFun y mejorar su diseño para que sea atractivo a los estudiantes.

Como conclusión final del Estado del Arte, se realizaron pruebas con Blockly para generar código en MateFun a partir de bloques en Android. A su vez, para las decisiones de implementación a lo largo del proyecto, se tuvieron en cuenta los aciertos y fracasos del resto de las aplicaciones estudiadas.

Capítulo 3

Estado inicial del proyecto MateFun

En la sección Motivación y contexto se realizó una breve introducción sobre qué es el lenguaje MateFun, los objetivos para los cuales fue pensado y sus diferentes componentes.

El presente proyecto implica adaptar las funcionalidades existentes de MateFun, con el fin de que las mismas puedan ser utilizadas en un cierto contexto (en particular, por estudiantes de primaria). Por tal motivo, fue necesario tomar contacto en profundidad con las características de MateFun como lenguaje y con las distintas soluciones empleadas para su funcionamiento en sus diferentes versiones.

Como se menciona en el Capítulo 4, el intérprete y componentes de otras implementaciones de MateFun existentes fueron utilizados y adaptados para incluirse en la aplicación Android desarrollada.

Por tales motivos, en este capítulo se presentan características de MateFun como lenguaje y los distintos sistemas que dan soporte al mismo, fruto de proyectos de grado anteriores. En concreto, serán abordadas las características que resultaron de interés para el proyecto actual, haciendo referencia a la documentación existente del resto de los proyectos cuando así se requiera.

3.1. Intérprete de MateFun

El intérprete de MateFun es el componente principal para el proyecto. En él se define e implementa el lenguaje MateFun como tal.

Se trata de una aplicación de consola, desarrollada en Haskell por docentes del Instituto de Computación (InCo) de la Facultad de Ingeniería. Este componente, permite interpretar e interactuar con otros programas escritos en lenguaje MateFun.

El intérprete cuenta con diferentes módulos encargados de definir estructuras de datos, tipos, generación de gráficas, chequeos de errores, entre otros (Vázquez, 2019). Fue importante su estudio y se tuvieron presentes durante el desarrollo de la aplicación. Incluso dos módulos fueron quitados y otros modificados para poder integrar el intérprete directamente a la aplicación Android (tema abordado en detalle en la sección Integración del intérprete MateFun). A continuación se describen brevemente algunos de ellos.

Dentro de los módulos del intérprete se encuentran:

- **Core**: Contiene las estructuras y tipos necesarios para la ejecución del programa.
- **DomainIntersection**: Se encarga de chequear los dominios de cada función definida para resolver el problema de intersección de dominios.
- **Eval**: Módulo encargado de realizar evaluaciones de expresiones y condiciones.
- **InternationalizationHelper**: Módulo encargado de implementar la internacionalización. Por internacionalización se entiende la capacidad del sistema para adaptarse a diferentes idiomas y regiones.
- **Parser**: Se declaran las estructuras y funciones necesarias para realizar el parseo de los programas del usuario y llevarlos a estructuras adecuadas para ser manejados en tiempo de ejecución.
- **TypeCheck**: Módulo encargado de realizar el chequeo de tipos en funciones a ser cargadas.
- **Warning**: Módulo encargado de realizar el chequeo de advertencias en programas a ser cargados.
- **MateFun**: Módulo principal del intérprete MateFun.

- **Z3Helper**: Permite la interacción entre MateFun y el *solver* Z3.

En la Figura 3.1, se muestra un diagrama que representa las dependencias entre los módulos del intérprete. El sentido de la flecha indica la relación de dependencia entre éstos. A modo de ejemplo, es posible observar que el módulo *RenderFun* depende de los módulos *Core* y *ReservedNames*.

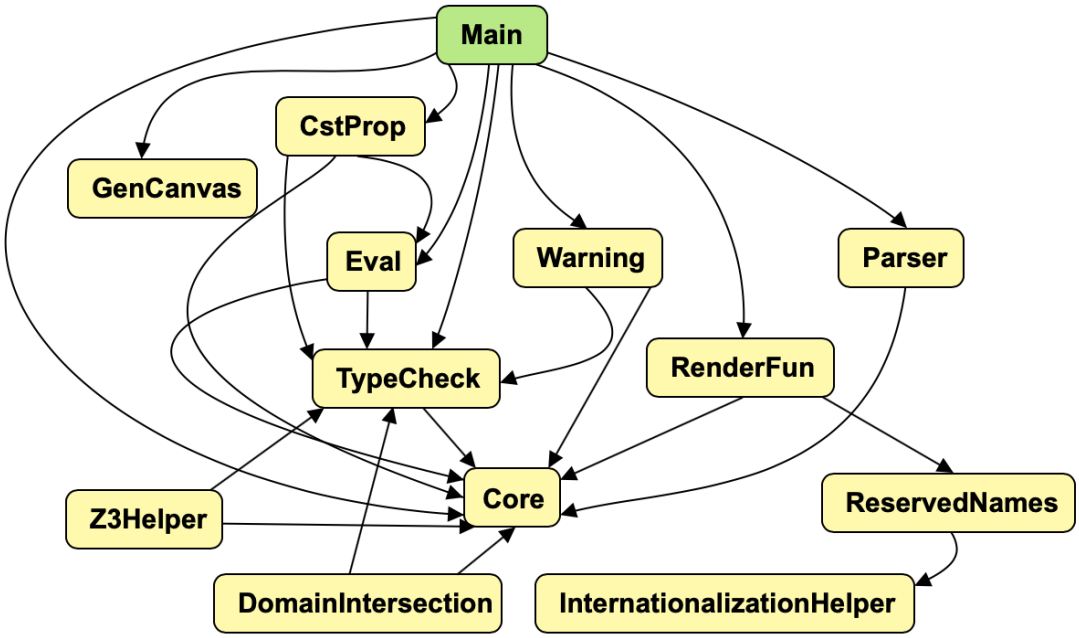

Figura 3.1: Diagrama de dependencias entre módulos del intérprete

3.1.1. Funciones predefinidas

El intérprete dispone de algunas funciones por defecto encargadas de realizar operaciones aritméticas, representar figuras tanto en dos como en tres dimensiones. En la Tabla 3.1 se muestran algunas de las funciones y figuras primitivas que dispone el lenguaje.

Función	**Firma**	**Descripción**
rect	(R x R) → Fig	Genera la figura rectángulo a partir de la dupla largo y ancho respectivamente
circ	R→Fig	Genera un círculo cuyo radio se recibe por parámetro
juntar	(Fig x Fig)→Fig	Genera una nueva figura que tiene como resultado la superposición de las figuras recibidas como parámetro
color	(Fig x Color)→Fig	Dados una figura y un color, colorea la figura con el color recibido como parámetro
mover	(Fig x (R x R)) → Fig	Mueve la figura recibida por parámetro al punto en el plano indicado por la dupla recibida en el segundo parámetro
rotar	(Fig x R)→Fig	Rota la figura en sentido horario de acuerdo a la cantidad de grados recibidos por parámetro
escalar	(Fig x R)→Fig	Escala la figura de acuerdo al factor del segundo parámetro
esfera	R→Fig3D	Genera una esfera de radio indicado en el primer parámetro
cilindro	(R x R x R)→Fig3D	Genera un cilindro de acuerdo a las dimensiones recibidas por parámetro
cubo	(R x R x R) → Fig3D	Genera un cubo de acuerdo a las dimensiones recibidas por parámetro

Tabla 3.1: Funciones primitivas en MateFun.

3.2. Aplicación Web de MateFun

MateFun cuenta con un entorno de desarrollo web. Se trata de una aplicación que permite a los usuarios interactuar con el intérprete desde un navegador web.

Desde el punto de vista del usuario, esta versión de MateFun consta de dos componentes principales: el intérprete interactivo y el área de visualización de figuras y animaciones.

El intérprete interactivo es el componente encargado de ejecutar las acciones que el usuario desea por medio de comandos que el lenguaje MateFun soporta. Tales acciones, generan respuestas que pueden ser visualizadas tanto en la consola del intérprete como en el área de visualización de figuras, dependiendo del tipo de respuesta.

Por otro lado, el componente responsable de la visualización de figuras se encarga de la representación en forma gráfica de los resultados que así lo requieran. Es el caso, por ejemplo, de la representación de una figura en 2D o una animación de figuras en 3D.

3.2.1. Tecnologías utilizadas

MateFun en su versión web utiliza diferentes tecnologías para poder resolver sus funcionalidades y manejar los distintos componentes.

La interfaz web corresponde a una *Single Web Aplication* (SPA) realizada en Angular 4.0, la cual es fácilmente modularizable y orientada a componentes. El estilo de la interfaz se logra mediante la tecnología Bootstrap 3, la cual permite generar una interfaz uniforme, amigable, de fácil utilización y adaptable a diferentes dispositivos (Cameto y Méndez, 2017).

Por otro lado, la consola interactiva se encuentra desarrollada en JavaScript utilizando la biblioteca JQConsole. Además, mediante la tecnología *WebSockets* se realiza la ejecución de comandos entre la capa de presentación y la capa de negocios (ver Figura 3.3).

La funcionalidad de graficar figuras fue mejorada a lo largo de distintos proyectos de grado. Originalmente, MateFun Web contaba con la posibilidad de representar figuras en 2D y funciones en una y varias dimensiones. Permitía crear y manipular figuras en 2D mediante funciones primitivas que el lenguaje MateFun tenía incorporado (Cameto y Méndez, 2017). Posteriormente, como

parte del proyecto de grado realizado por Rey y col. (2019), fueron incluidas mejoras al módulo 2D utilizando la biblioteca FunctionPlot.

Entre las carencias presentes en la versión inicial del proyecto web, se encontraba la imposibilidad de representar figuras en 3D. El proyecto mencionado agregó además, la creación de una nueva biblioteca basada en el *framework* Three.js, capaz de realizar el renderizado de diferentes figuras y animaciones en 3D.

En la Figura 3.2 se observa la interfaz de usuario del proyecto web.

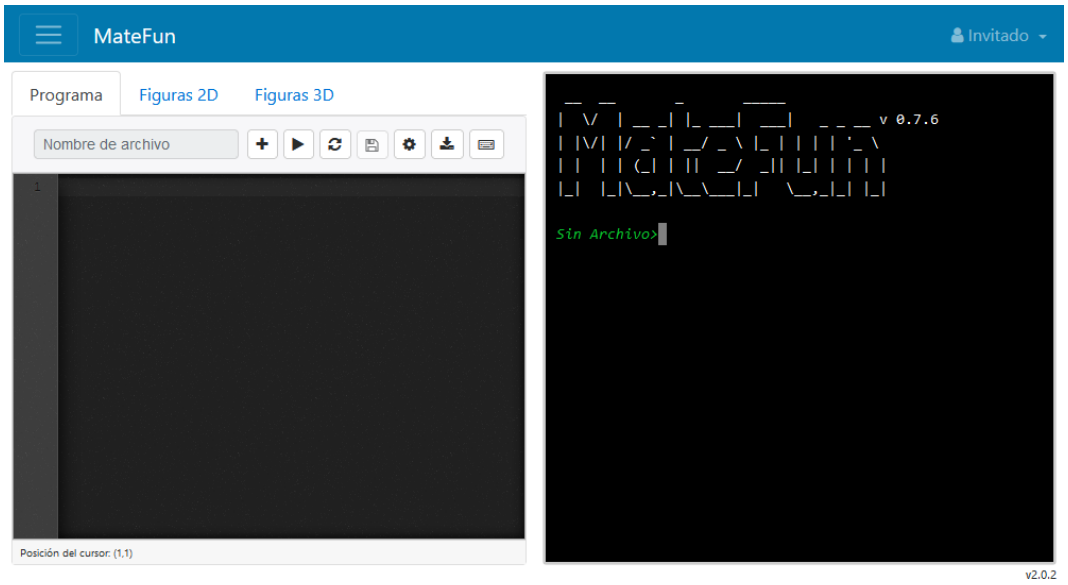

Figura 3.2: Interfaz de MateFun versión Web ("Matefun Interprete Web", s.f.)

3.2.2. Arquitectura

El entorno web de MateFun sigue una arquitectura cliente servidor. En los distintos proyectos involucrados se describe una arquitectura en capas, similar a la presentada en la Figura 3.3. A continuación, se detallan las responsabilidades de cada capa:

- **Capa de presentación:** Es ejecutada en los navegadores web de los usuarios. Permite que estos interactúen con el sistema, y se encarga de comunicar cuando es requerido la información hacia y desde la capa de negocios.

- **Capa de negocios:** Brinda servicios a la capa de presentación. Para el caso de la ejecución de código desde la consola web, se encarga de eje-

cutar un binario del intérprete de MateFun. A este le indica el comando enviado por el usuario. Posteriormente retorna el resultado a la capa de presentación.

- **Capa de persistencia:** Encargada de obtener y guardar los datos requeridos en una base de datos relacional.

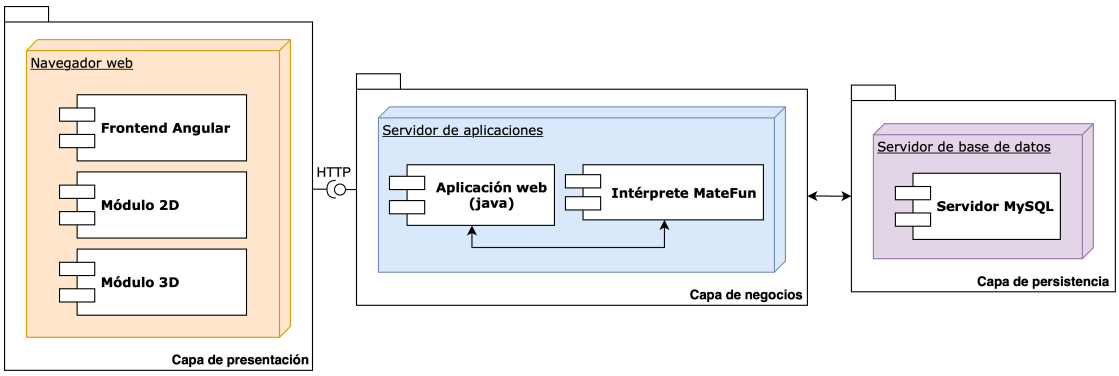

Figura 3.3: Diagrama de arquitectura de la aplicación web existente.

La comunicación entre el navegador web y el servidor de aplicaciones es realizada mediante *Web Services* REST y una conexión *WebSocket*. El servidor Java expone una API REST para las operaciones del *frontend* y el manejo de archivos, mientras que el *WebSocket* se utiliza para la comunicación bidireccional entre el navegador web y el intérprete de MateFun, utilizando al servidor Java como intermediario. Las acciones realizadas por el usuario en el navegador web son enviadas al servidor de aplicaciones por medio de mensajes en formato JSON, los cuales son recibidos por el intérprete MateFun (Cameto y Méndez, 2017). El intérprete, genera una respuesta de vuelta hacia el navegador web, desplegando mensajes en consola o representando tanto figuras como animaciones por medio de los módulos 2D y 3D.

Capítulo 4

Desarrollo de la aplicación

4.1. Prototipado

Uno de los componentes clave de la aplicación es la interfaz de usuario, la cual se espera sea atractiva e intuitiva para niños. Como una primera aproximación a la interfaz, se realizó un prototipo de baja fidelidad.

Inicialmente, en conjunto con los tutores, fue diseñada una interfaz basada en elementos con forma de tubería. Cada tubería estaba compuesta por una o más entradas y una salida. Como forma de identificar la funcionalidad de cada tubería, se consideró incluir una imagen dentro de la misma. A su vez, las tuberías tenían asignado un color teniendo en cuenta la categoría a la cual pertenecían. Inicialmente se consideraron categorías como operaciones binarias, condicionales, figuras, entre otras.

Además, puede observarse la distinción de colores por categoría y la utilización de iconos para indicar la operación o acción realizada por cada componente visual. Estas decisiones fueron tomadas siguiendo las conclusiones extraídas del estudio del Estado del Arte.

En la Figura 4.1 se observa el prototipo de baja fidelidad considerando el diseño de tuberías.

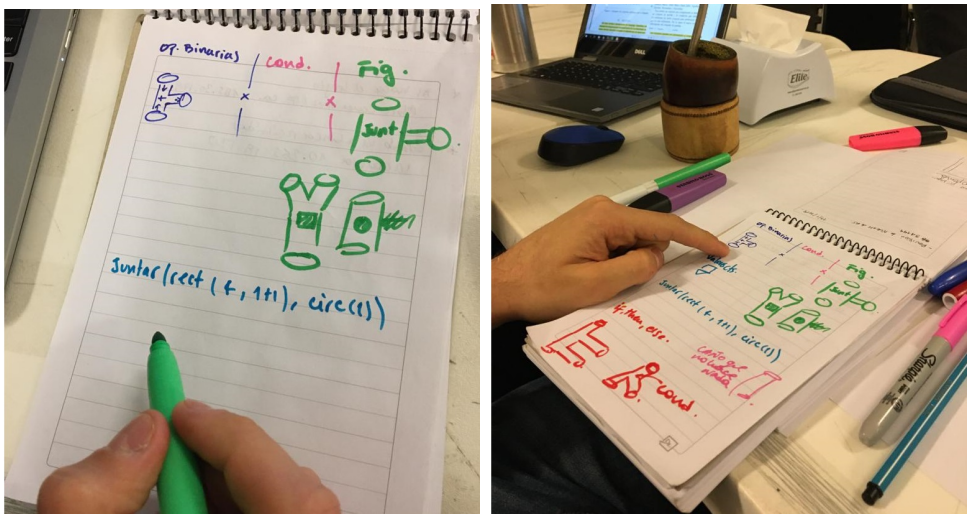

Figura 4.1: Prototipo de baja fidelidad, se muestra forma de tuberías, colores y categorías.

A pesar de que el diseño basado en tuberías parecía alentador, el equipo consideró que podría ser un riesgo alto para el proyecto debido a que el mismo no había sido probado en niños. Al mismo tiempo, el riesgo asociado a la elección de tecnologías para el desarrollo y la falta de experiencia por parte del equipo en desarrollo de interfaces visuales, fue decisivo para no considerar dicho diseño.

Por lo anterior, el equipo decidió la utilización de Blockly como base para el diseño de la interfaz visual, fundamentalmente por el hecho de ha sido utilizado en numerosos proyectos (incluidos Scatch y Snap!) incluyendo niños.

Teniendo en cuenta las funcionalidades ofrecidas por Blockly estudiadas hasta ese momento, se realizó un prototipo de media fidelidad. Este fue presentado a los tutores como prueba de concepto a través de un video. El prototipo en este caso, contaba con pocas funcionalidades, pero tenía como objetivo mostrar el diseño con bloques. Se presentaron además ciertas funcionalidades proporcionadas por Blockly como arrastrar, unir y borrar bloques utilizando la papelera, como se muestra en la Figura 4.2.

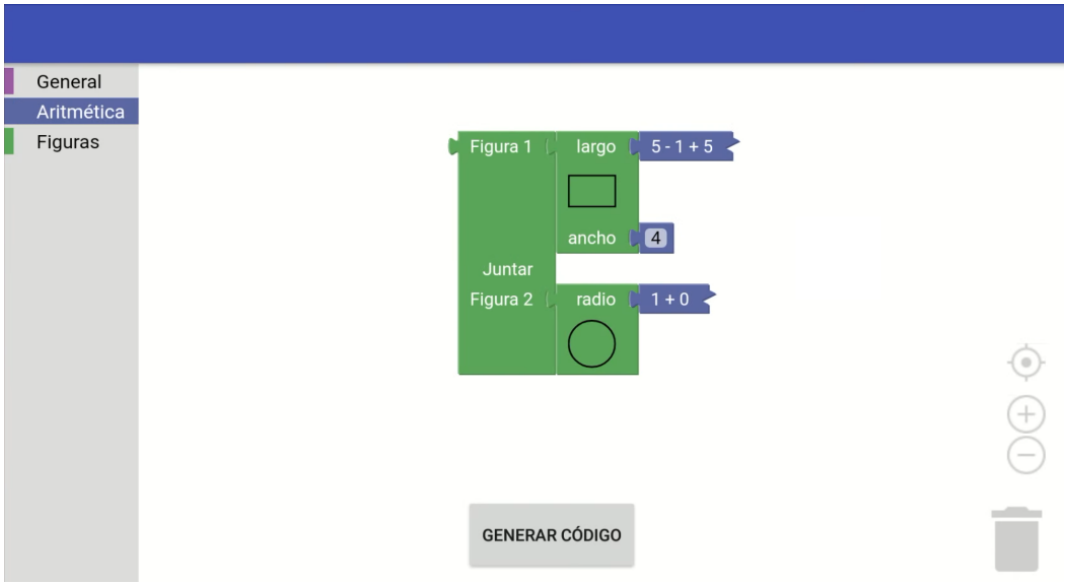

Figura 4.2: Prototipo de media fidelidad, se muestran funcionalidades de Blockly y forma de los bloques.

Una vez validado y aprobado en conjunto con los tutores el prototipo de media fidelidad, se avanzó hacia el prototipo de alta fidelidad. El mismo contó con un proceso iterativo.

En la primera iteración se agregaron todos los bloques disponibles en MateFun, agrupados por categorías. A su vez, fueron incorporadas imágenes dentro de los bloques para representar las acciones de los mismos. En la Figura 4.3 se puede observar el prototipo de alta fidelidad antes mencionado.

Este prototipo fue presentado en una reunión realizada en Julio de 2020 con el Centro Interdisciplinario en Cognición para la Enseñanza y el Aprendizaje (CICEA, 2020). De esta presentación se obtuvo una buena devolución sobre el prototipo mostrado. Además, surgieron comentarios y sugerencias en su mayoría respecto a las imágenes utilizadas en los bloques. Sugirieron uniformizar las imágenes y utilizar aquellas que no infligieran derechos de autor.

De esta forma, se llegó a la última iteración y al diseño de la interfaz final. En la Figura 4.4, se observa el resultado final de la interfaz de la aplicación.

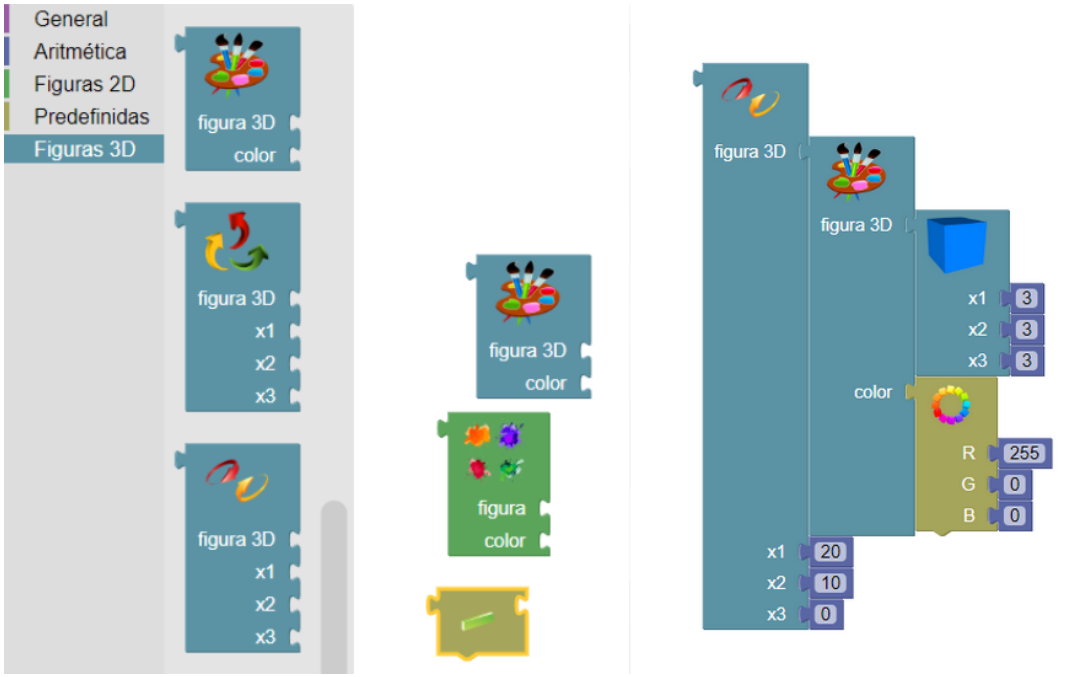

Figura 4.3: Prototipo de alta fidelidad, conteniendo todas las categorías disponibles, iconos y funciones de MateFun.

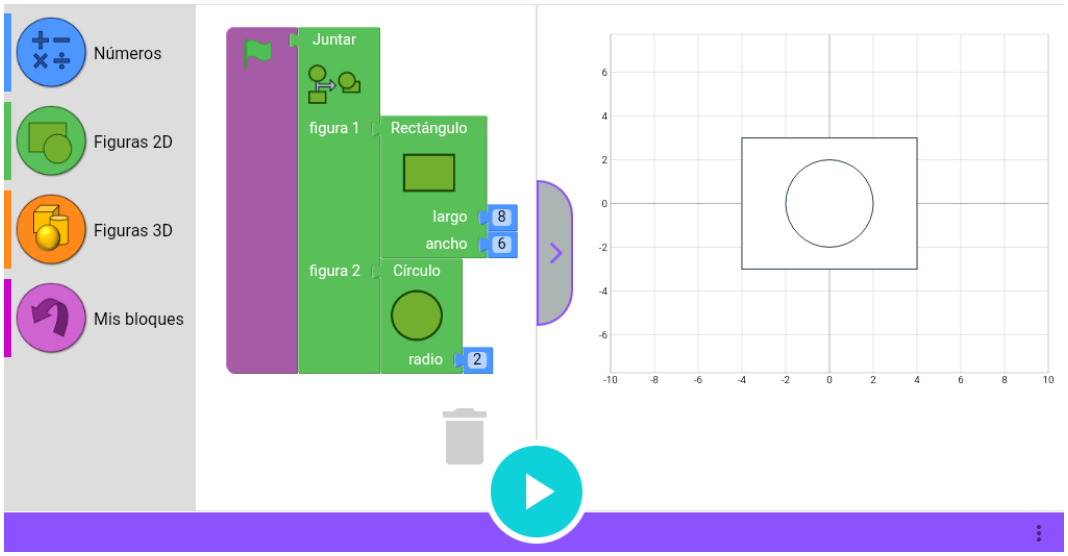

Figura 4.4: Resultado final de la interfaz, con paleta de bloques e iconos bien definidos, y con panel para mostrar resultados gráficos.

Para el diseño, fueron considerados los comentarios respecto a las imágenes en los bloques. En base a ellos, las imágenes necesarias fueron creadas específicamente para la aplicación y se permite modificarlas fácilmente en caso que sea requerido a futuro. A su vez, se modificaron los colores y se implementa-

ron mejoras en la Paleta de bloques y en las categorías, con el fin de que la aplicación en su conjunto resultase más atractiva para el publico objetivo.

Respecto al diseño general de la aplicación, se optimizaron espacios, quitando elementos innecesarios como la barra superior (incluida por defecto en Android, como puede verse en la Figura 4.2) y botones innecesarios (para varias acciones se aprovechó el potencial de la pantalla táctil).

4.2. Componentes de la interfaz

En esta sección se presentan los diferentes componentes de la interfaz de usuario de la aplicación. Es importante que el lector se familiarice con ellos ya que serán mencionados en secciones posteriores.

El diseño de la aplicación se compone de diferentes elementos, presentados en la Figura 4.5 y que se detallan a continuación.

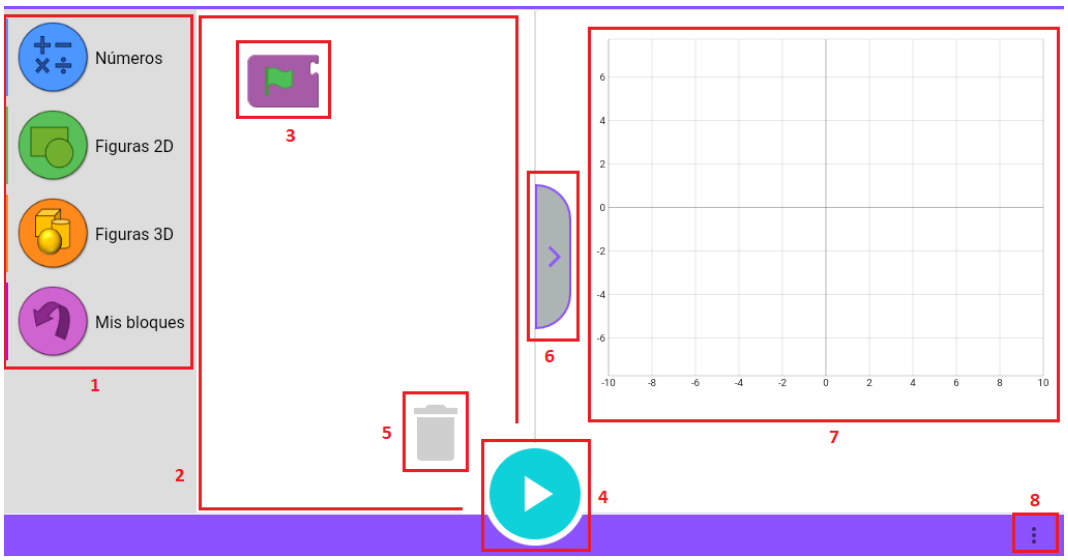

Figura 4.5: Secciones y elementos en pantalla principal.

1. **Paleta de bloques:** Contiene las cuatro categorías de bloques disponibles en la aplicación. Dentro de cada una de ellas, se encuentran los bloques que pueden ser utilizados para generar programas.

2. **Área de trabajo:** Es el lugar hacia donde se desplazan los bloques para generar un programa.

3. **Bloque bandera:** Es utilizado como bloque inicial. Los bloques que

serán considerados al momento de ejecutar un programa, guardar o eliminar bloques, son aquellos que están conectados a este bloque. Todo bloque que no este unido a él no será tenido en cuenta y se mostrará deshabilitado. No podrá ser borrado y siempre permanece en el área de trabajo.

4. **Botón ejecutar:** Al ser presionado, ejecuta el programa creado con bloques. Considera solamente los bloques conectados al bloque bandera.

5. **Papelera:** Se utiliza para borrar un bloque del área de trabajo. Todos los bloques desplazados hacia ella serán eliminados.

6. **Área de gráficas:** En caso que el resultado de ejecutar un programa realizado con bloques sea una figura o una animación, será mostrada en está área.

7. **Botón para desplegar gráficas:** Permite mostrar y ocultar el área de gráficas.

8. **Menú para "Mis Bloques":** Contiene funciones para ser utilizadas con los bloques creados por el usuario (Exportar, Importar, Guardar, Eliminar y Eliminar todos).

Existen otros bloques especiales que se presentan a continuación:

- **Bloque interrogante**: Se utiliza solo para el guardado de bloques. Sirve para asignar nombres a las entradas del nuevo bloque al momento de guardarlo. Se puede ver la representación del mismo en la Figura 4.6. La funcionalidad de este bloque se explica en detalle en la sección Bloque interrogante.

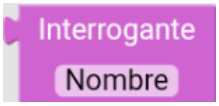

Figura 4.6: Ejemplo de bloque interrogante en la aplicación.

- **Bloque sombra**: Aparece en las entradas de los bloques con valores por defecto, permitiendo ingresar un valor, sin la necesidad de arrastrar un bloque a dicha entrada. Se puede ver la representación del mismo en la Figura 4.7.

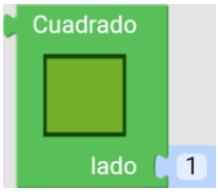

Figura 4.7: Bloque cuadrado con bloque sombra en su entrada.

Se puede ampliar la información presentada en esta sección, consultando el "Manual de Usuario".

4.3. Análisis y diseño

En esta sección se describen los requerimientos, casos de uso y arquitectura de la aplicación. Los requerimientos funcionales y no funcionales, así como los distintos casos de uso, surgen de varias reuniones periódicas con los tutores. Se fueron relevando en conjunto de forma iterativa e incremental.

A partir de los elementos de análisis relevados, se desprende la arquitectura de la aplicación y condicionantes respecto a las tecnologías a utilizar y al alcance del proyecto.

Con respecto a los requerimientos y casos de uso, se incluyen los más relevantes para el informe en las siguientes secciones. Se pueden ver en su totalidad consultando el "Documento de análisis y diseño".

4.3.1. Requerimientos

En esta sección, se describen algunos de los requerimientos que surgieron tanto en la etapa de análisis como en la etapa de desarrollo del proyecto.

Los requerimientos están escritos de forma genérica. Sin embargo, puede resultar útil aclarar que los conceptos **componentes** o **elementos** del lenguaje de programación visual, equivalen o se corresponden al concepto de bloque en la aplicación desarrollada.

4.3.1.1. Requerimientos funcionales

- **Generar programas en forma visual:** La aplicación debe permitir crear programas en lenguaje MateFun a partir de una interfaz visual.

- **Ejecutar programas generados:** Los programas generados podrán ejecutarse y la aplicación debe permitir mostrar el resultado.

- **Disponibilizar un subconjunto de expresiones y operaciones de MateFun:** Para crear programas de forma visual, se deben poner a disposición del usuario, elementos que representen las expresiones y operaciones de MateFun que sean apropiadas para un niño en edad escolar.

- **Disponibilizar nuevos elementos visuales que no estén en MateFun:** Sumado a lo mencionado en el ítem anterior, la aplicación debe ofrecer otras operaciones o componentes visuales para complementar las funciones definidas en MateFun. A modo de ejemplo, MateFun incluye las funciones para definir un rectángulo y un prisma rectangular, las cuales precisan dos y tres entradas respectivamente. Se debe definir cuadrado y cubo, ambos con una sola entrada.
  Se requiere agregar (al menos) los siguientes elementos que no se encuentran definidos en MateFun:

  - Cuadrado
  - Cubo
  - Cilindro
  - Cono

- **Indicar tipos en las entradas y salidas de los componentes visuales:** Los componentes del lenguaje visual deben tener un tipo asociado a cada una de sus entradas y salidas. Es deseable tener alguna indicación del tipo esperado en cada componente, por ejemplo mediante colores, para facilitar la comprensión y la creación del programa. Se deberá realizar un chequeo de tipos al momento de unir componentes.

- **Representar figuras:** La aplicación debe permitir generar o representar figuras 2D y 3D de forma análoga a como se hace en la interfaz web existente de MateFun.

- **Representar animaciones:** La aplicación debe permitir generar o representar animaciones 2D y 3D de forma análoga a como se hace en la interfaz web existente de MateFun.

- **Guardar programa en nuevo componente:** A partir de un programa creado con varios componentes del lenguaje visual, se debe poder generar un nuevo componente que encapsule la funcionalidad del programa realizado. La funcionalidad es análoga a la definición de una función. El nuevo componente debe quedar guardado y disponible para ser utilizado. Al momento de ser creado, debe ser posible asignar nombre a los parámetros, junto con un nombre y una imagen para el nuevo componente.

- **Exportar/importar componentes creados:** Debe ser posible exportar e importar (a través de archivos) los componentes creados a partir de programas realizados con el lenguaje visual.

4.3.1.2. Requerimientos no funcionales

- La aplicación debe concebirse para ser utilizada por niños en edad escolar.

- La aplicación debe funcionar en *tablets* Android. Es deseable que funcione en las *tablets* que entrega el Plan Ceibal.

- La aplicación debe funcionar sin requerir de servicios externos o conexión a Internet.

- En caso de ser requerido, los iconos y elementos visuales tienen que poder ser modificados fácilmente.

- La aplicación debe contar con manual de usuario.

- La aplicación debe proporcionar mensajes de error que sean informativos y orientados al usuario final.

4.3.2. Casos de uso

En esta sección se presenta un subconjunto de los casos de uso de la aplicación, los cuales ayudan a entender los flujos más importantes en esta. Los casos de uso fueron definidos a partir del relevamiento de requerimientos.

Respecto a los actores del sistema, la aplicación solo tiene un tipo de usuario, el cual interactúa con la interfaz de bloques y realiza distintas acciones a partir de los programas que genera.

A continuación, se especifican en alto nivel los casos de uso seleccionados.

En el "Documento de análisis y diseño" se especifican en su totalidad (también en alto nivel) y pueden encontrarse a su vez, los más importantes presentados en forma expandida.

En caso de ser necesario, para contextualizar los casos de uso, puede ser de utilidad consultar el "Manual de usuario". En ese documento, se presentan capturas de pantalla y se explica el uso de las funcionalidades implementadas.

4.3.2.1. Crear programa con bloques

El caso de uso comienza cuando el usuario quiere crear un programa con bloques. Para esto, debe arrastrar y unir bloques desde la Paleta de bloques al área de trabajo, creando una secuencia con origen en el Bloque bandera. Los bloques que no estén unidos a la bandera quedarán deshabilitados y no forman parte del programa.

Al momento que se mueva un bloque, serán resaltadas las entradas en el resto de los bloques del área de trabajo donde se pueda unir (según el tipo). En caso que se intenten unir dos bloques y los tipos de las uniones no coincidan, los bloques se separan automáticamente.

En algunos casos (por ejemplo, con bloques de tipo número), como alternativa a arrastrar y unir bloques desde la paleta de bloques, se cuenta con bloques sombra. Estos tienen un valor predefinido, que puede ser modificado o sobrescrito con otro bloque.

El caso de uso finaliza cuando el usuario no desea agregar más bloques a su programa y tiene al menos uno conectado al Bloque bandera.

4.3.2.2. Ejecutar programa creado

El caso de uso comienza cuando el usuario quiere ejecutar un programa creado con bloques. Como precondición, debe ser instanciado previamente el caso de uso Crear programa con bloques. Para ejecutar un programa, el usuario presiona el botón ejecutar. Si el programa es válido se muestra el resultado. Si no, muestra un error acorde. El resultado puede ser numérico, o la representación gráfica de una figura o animación. El caso de uso finaliza cuando se despliega el resultado o el error correspondiente.

4.3.2.3. Crear figuras 2D

El caso de uso comienza cuando el usuario quiere graficar figuras 2D. En este caso, se instancia en primer lugar al caso de uso Crear programa con bloques, con la particularidad de usar bloques que representan figuras 2D o aquellos que implican operaciones sobre estas (como por ejemplo, mover, rotar, colorear). A su vez será necesario usar bloques numéricos o bloques sombra para especificar los parámetros necesarios que definen a las figuras. Luego, se instancia al caso de uso Ejecutar programa creado. El caso de uso finaliza cuando se muestra la figura representada o un error de ejecución.

4.3.2.4. Crear animaciones 2D

El caso de uso comienza cuando el usuario quiere representar animaciones 2D. En este caso, se instancia en primer lugar al caso de uso Crear programa con bloques, con la particularidad de usar bloques que representan figuras 2D, aquellos que implican operaciones sobre estas (Ej. mover, rotar, colorear), bloques numéricos o bloques sombra para especificar los parámetros necesarios que definen a las figuras y los bloques de animación 2D. Luego, se instancia al caso de uso Ejecutar programa creado. El caso de uso finaliza cuando se muestra la figura representada o un error de ejecución.

4.3.2.5. Guardar programa creado (generar y guardar nuevo bloque)

El caso de uso comienza cuando el usuario quiere guardar un programa, creando un nuevo bloque. Debe ser instanciado previamente el caso de uso Crear programa con bloques. Al momento de crear el programa, se puede dejar entradas libres (pasarán a ser entradas -con nombre aleatorio- del nuevo bloque) o agregar un bloque interrogante para asignarle un nombre a los parámetros del nuevo bloque. Para guardar el programa, el usuario presiona el botón guardar en el menú Mis Bloques. Luego debe introducir el nombre para el nuevo bloque y opcionalmente una imagen. El caso de uso finaliza cuando se despliega un mensaje de éxito, se persiste el nuevo componente y se agrega a la categoría Mis bloques.

4.3.3. Arquitectura

Si bien la aplicación integra varios componentes o módulos, en cuanto al estilo arquitectónico, corresponde a una aplicación **Monolítica**. Esto implica que el sistema es autosuficiente, contiene todas las funcionalidades necesarias para realizar la tarea para el cual fue diseñado, sin contar con dependencias externas que complementen su funcionalidad.

En la Figura 4.8 se observan los distintos componentes de la aplicación. A continuación, se describen los mismos brevemente (serán abordados en detalle en las secciones Implementación y Diagramas de Comunicación):

- **Blockly:** permite generar código en formato MateFun a partir de bloques previamente definidos o guardados por el usuario. La definición y el manejo de los bloques son responsabilidad de este componente.

- **Intérprete de MateFun:** permite ejecutar código en lenguaje MateFun, generado a partir de los bloques y obtener la respuesta correspondiente en cada caso. Se utilizó una versión adaptada del intérprete de MateFun existente, tal como se explica en la sección Integración del intérprete MateFun.

- **Módulo de figuras:** permite representar gráficamente figuras 2D y 3D de forma similar a como es realizado en la versión web de MateFun. Requiere un archivo JSON generado por el intérprete de MateFun como entrada para poder generar las figuras. Dentro de este módulo, existe un componente encargado de representar figuras 2D y otro encargado de representar figuras 3D, teniendo cada uno sus responsabilidades y diferencias en la implementación. Dicho módulo fue adaptado desde la Aplicación Web de MateFun, tal como se explica en la sección Integración de componentes para representar figuras.

- ***Core* de la aplicación:** se utiliza este componente para agrupar y describir las funcionalidades de la aplicación Android como tal. Entre ellas se encuentran:
  - Manejo de *layouts*, botones, *WebViews* y otros componentes visuales.
  - Implementa la lógica necesaria para integrar los distintos componentes del sistema.

- Accede al sistema de archivos del dispositivo Android para las funcionalidades necesarias. A modo de ejemplo, permite persistir los bloques guardados por el usuario.

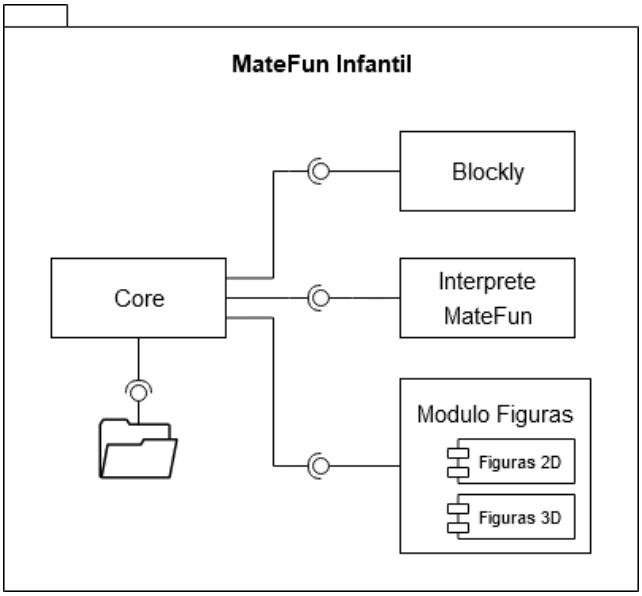

Figura 4.8: Diagrama de arquitectura.

Aunque los componentes presentados son parte de un sistema monolítico, internamente no presentan acoplamiento entre ellos. El acceso a los mismos se realiza mediante llamado a funciones, utilizando interfaces y sus responsabilidades están diferenciadas apropiadamente.

Desde el punto de vista lógico, pensando en un modelo de "programación por capas" (2020), Blockly y el módulo de figuras son componentes de la capa de presentación debido a la interacción del usuario con estos componentes.

Por otro lado, el intérprete de MateFun forma parte de la lógica de negocios ya que no hay interacción directa con el usuario.

Como fue mencionado en esta sección, el componente *Core* se encarga de aspectos de presentación e interacción con el usuario. A su vez, involucra funcionalidades del negocio e integración.

Con respecto a la capa de datos, la persistencia utilizada por la aplicación se realiza por medio de archivos JavaScript en el sistema de archivos del dispositivo Android donde se ejecuta. Los archivos JavaScript son utilizados para la funcionalidad del guardado de bloques.

4.3.4. Diagramas de Comunicación

En esta sección se presentan diagramas de comunicación de aquellas funcionalidades más relevantes y que a su vez permiten comprender la comunicación entre los distintos componentes descriptos en la sección Arquitectura.

4.3.4.1. Ejecutar programa realizado con bloques

El flujo general para crear y ejecutar un programa en la aplicación se da de la siguiente forma. En primer lugar el usuario crea su programa, arrastrando y uniendo bloques desde la Paleta de bloques al área de trabajo. Al ejecutar el programa creado, desde el componente *Core* se valida que el programa esté bien formado (llamando a funciones de Blockly) y en caso que así sea se obtiene a partir de los bloques, una expresión válida para ser evaluada en MateFun (si tiene errores se muestra un mensaje acorde al usuario). Dependiendo si el resultado es numérico o una figura, se despliega el resultado en un componente de texto o se invoca al módulo de figuras.

En la Figura 4.9, se presenta el diagrama de comunicación para esta funcionalidad.

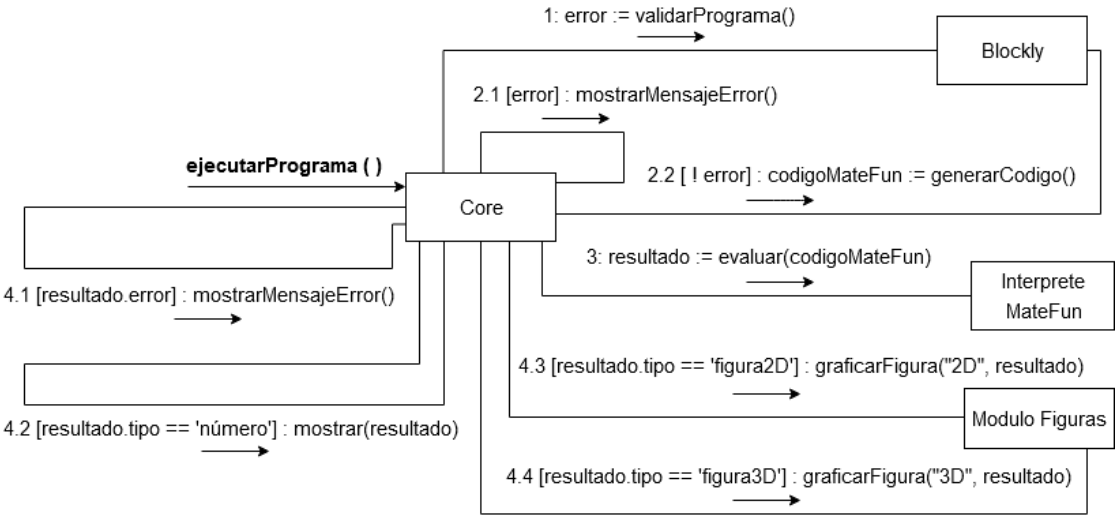

Figura 4.9: Diagrama de comunicación, representación de ejecutar programa.

4.3.4.2. Guardar un nuevo bloque en la Paleta de bloques

Para guardar un nuevo bloque (o dicho de otra forma, generar un bloque a partir del programa creado por el usuario con otros bloques), el flujo es el

siguiente. En primer lugar el usuario crea su programa, arrastrando y uniendo bloques desde la Paleta de bloques al área de trabajo. Al momento de guardar, se abre una ventana emergente en la cual se elige el nombre del nuevo bloque y un icono. En la aplicación, es el componente *Core* quien se encarga de obtener el código generado por los bloques a partir del componente Blockly, hacer un preprocesamiento y persistirlo en el sistema de archivos. Luego el bloque guardado queda disponible en la categoría "Mis bloques", desde donde podrá ser utilizado como cualquier otro.

En la Figura 4.10, se presenta el diagrama de comunicación para esta funcionalidad.

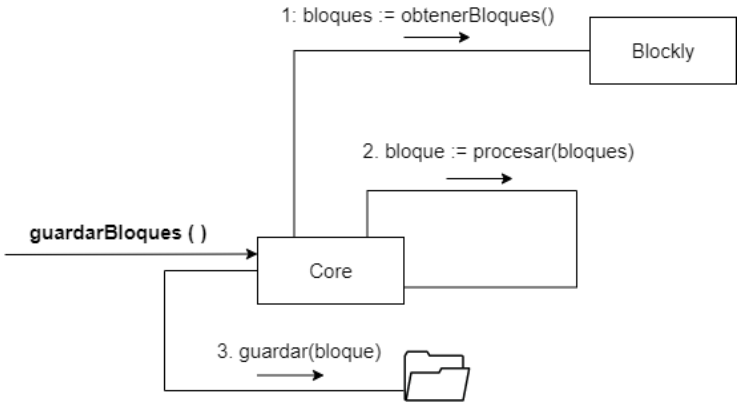

Figura 4.10: Diagrama de comunicación, representación del guardado de nuevos bloques.

4.3.5. Diseño de funcionalidad guardado de bloques

En esta sección se presenta el diseño del guardado de bloques creados por el usuario. Esta funcionalidad será abordada en detalle, dado que se trata de una de las funcionalidades más importantes que tiene MateFun Infantil.

El guardado de bloques permite, dado un conjunto de bloques, encapsular los mismos en un nuevo bloque y guardarlo. Al ejecutar el nuevo bloque, el resultado esperado será el mismo que el resultado de evaluar el conjunto de bloques inicial.

La funcionalidad en cuestión, intenta representar el concepto de función matemática o de "caja negra" en *software*, donde luego de tener definido el programa solo importan las entradas y la salida.

Para fijar ideas, en la Figura 4.11 se tiene un conjunto de bloques conectados al bloque bandera. Los mismos representan un programa creado con el

lenguaje MateFun.

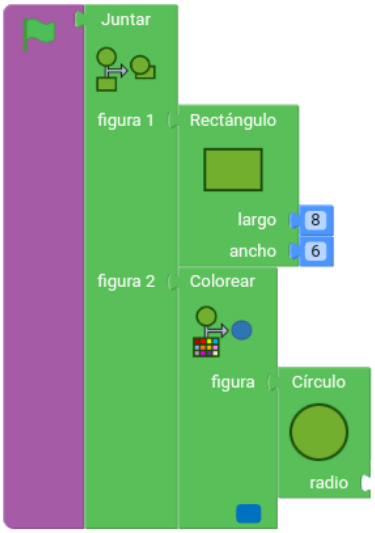

Figura 4.11: Programa MateFun previo a ser guardado.

Al guardar estos bloques, como se observa en la Figura 4.12, se genera un nuevo bloque el cual queda disponible en la Paleta de bloques para poder ser utilizado.

Figura 4.12: Bloque guardado disponible dentro de la categoría Mis bloques.

Finalmente, en la Figura 4.13, se observa el nuevo bloque que "encapsula" el conjunto de bloques inicial y el resultado de su ejecución.

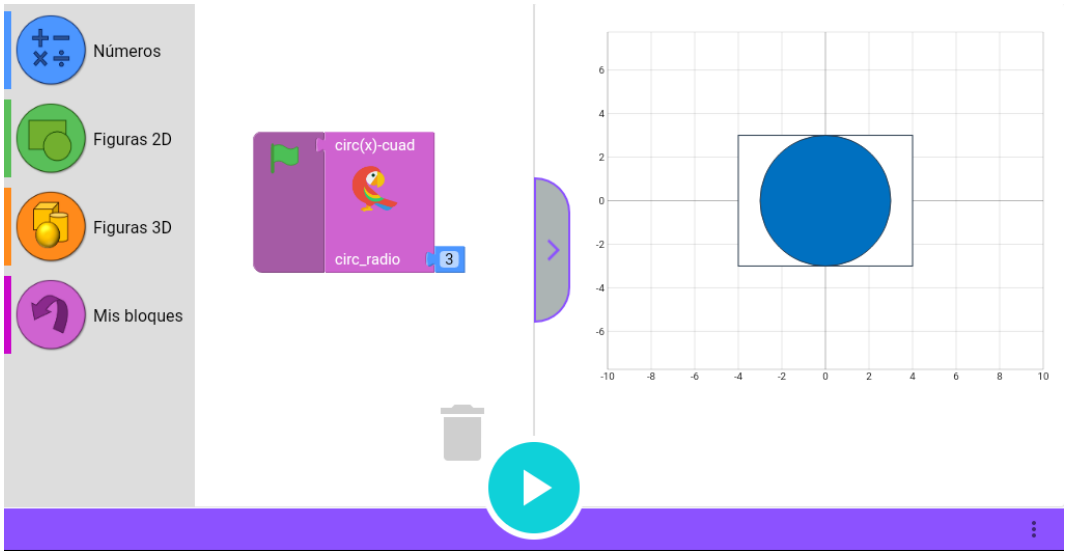

Figura 4.13: Resultado de la ejecución de un bloque guardado.

Desde el punto de vista de implementación, durante el desarrollo de esta funcionalidad fueron tomadas varias decisiones las cuales serán abordadas en la sección Implementación del guardado de bloques.

4.3.5.1. Bloque interrogante

El bloque interrogante únicamente se utiliza durante el guardado de bloques. Sus funcionalidades incluyen indicar el nombre de una entrada para un nuevo bloque y agrupar entradas del mismo tipo (se explica en detalle en la sección Entradas en nuevos bloques).

El bloque interrogante a su vez tiene un rol educativo para el alumno, ya que en primaria se trabaja con el concepto de variable, también llamada interrogante. Luego de analizar el "Programa de educación inicial y primaria" (2008) y consultar con maestros, a partir del artículo "Álgebra: aportes para nuevas reflexiones" (2009), se extrajeron ejemplos para trabajar el concepto en cuestión: "La variable como expresión del número generalizado".

En estos ejercicios se trabaja, por ejemplo, con múltiplos de 8 utilizando la expresión 8xV. En este caso V sería la variable generalizada. Como se muestra en la Figura 4.14, el ejemplo anterior se puede realizar con MateFun Infantil utilizando un bloque multiplicación y un bloque interrogante.

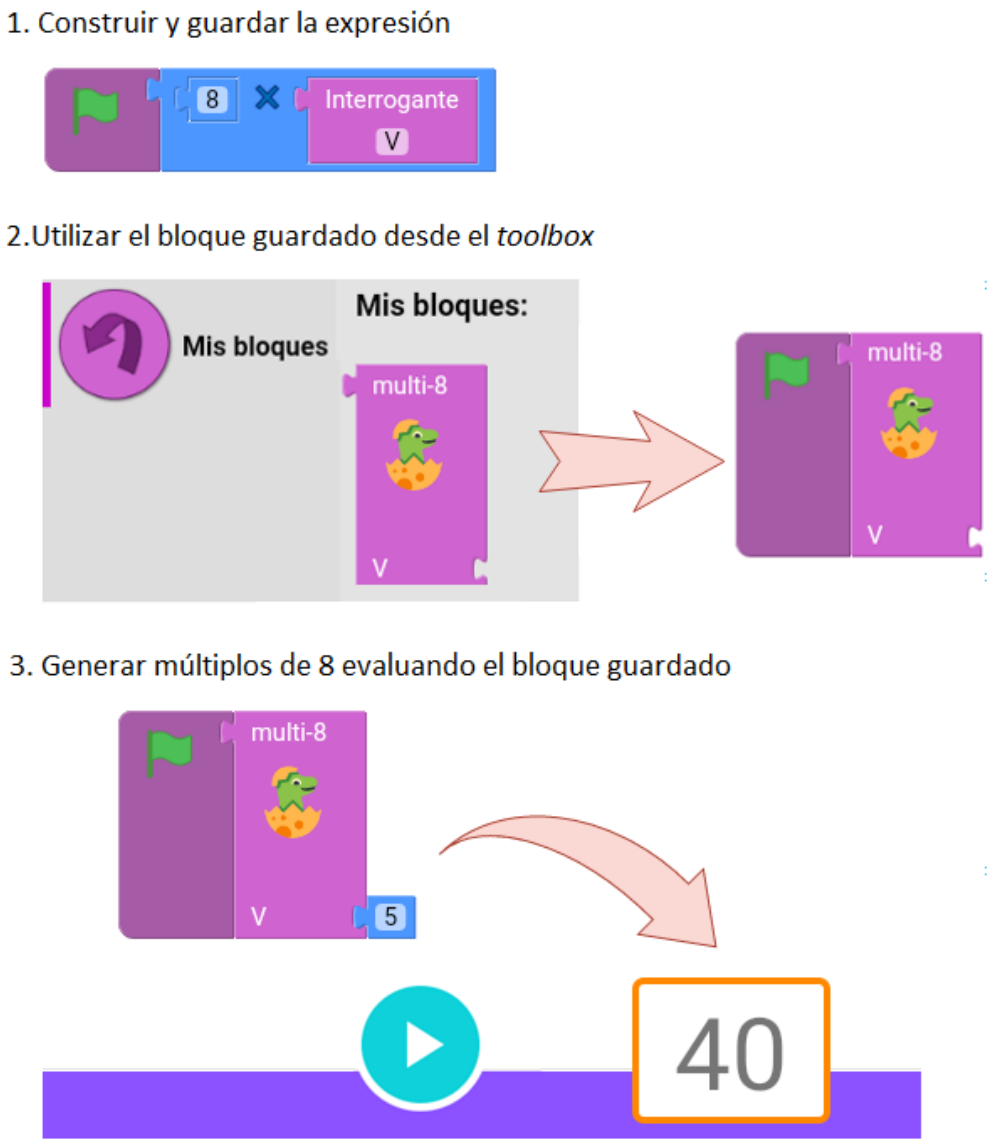

Figura 4.14: Ejemplo de creación y utilización de bloque que genera múltiplos de 8.

4.3.5.2. Entradas en nuevos bloques

Las entradas de los nuevos bloques, es decir, de aquellos que se generan al guardar un conjunto de bloques, pueden indicarse en el conjunto inicial de las siguientes formas:

1. Agregando un bloque interrogante
2. Dejando una entrada con un bloque sombra
3. Dejando una entrada sin bloque (libre)

A modo de ejemplo, en la Figura 4.15 se presentan las formas de indicar entradas para nuevos bloques. Los números presentados sobre la Figura se corresponden con los ítems enumerados.

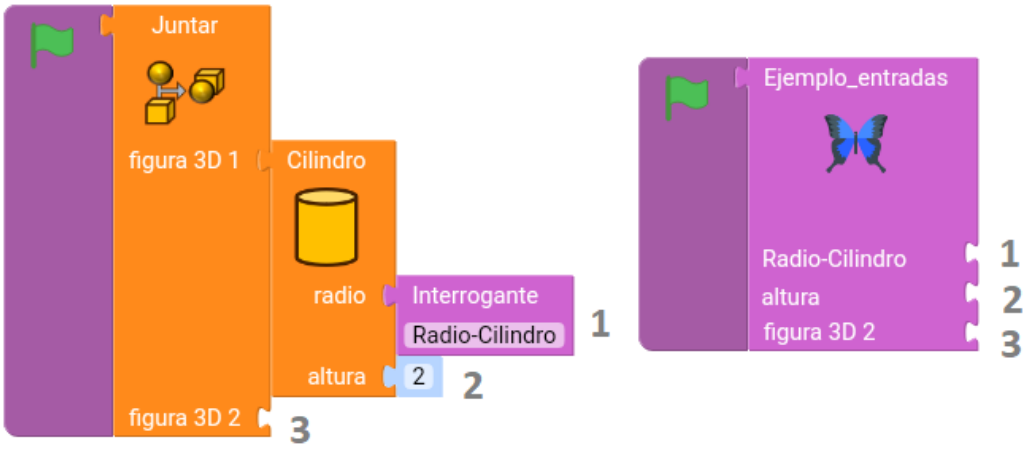

Figura 4.15: Formas de indicar entradas para un nuevo bloque.

Dejando una entrada libre o con un bloque sombra, las entradas del nuevo bloque toman como nombre el de la entrada que se está dejando libre. En cambio, con el bloque interrogante, se puede indicar explícitamente el nombre que debe ser asignado a esa entrada en el nuevo bloque.

Es importante destacar que la entrada del nuevo bloque conserva el tipo de la entrada original.

Para agrupar entradas del mismo tipo, se debe agregar un bloque interrogante con el mismo nombre en cada una de las entradas a agrupar. Al crear un nuevo bloque, este tendrá una entrada con el nombre elegido, que luego "replicará" el valor asignado a esta entrada en todas las que tenían el bloque interrogante en la agrupación inicial guardada. En la Figura 4.16 se observa la funcionalidad presentada.

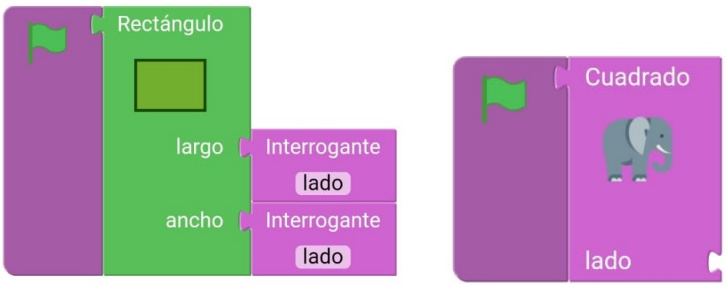

Figura 4.16: Ejemplo de bloque interrogante con mismo nombre.

4.3.5.3. Edición de nuevo bloque

Otra funcionalidad implementada sobre los bloques guardados consiste en ver cómo están compuestos, es decir, ver cuál es la secuencia inicial de bloques que lo generó.

Esta funcionalidad se puede utilizar presionando sobre un bloque y eligiendo la opción Editar. Al hacer esto, se muestra sobre el área de trabajo dicha secuencia.

Luego, si se guarda la secuencia nuevamente, se sugiere el nombre del bloque que se esta editando para permitir actualizar el mismo. El detalle del paso a paso se puede ver en el "Manual de Usuario".

4.3.5.4. Actualización y borrado

Teniendo un bloque guardado se permite su actualización o borrado.

Para actualizar un bloque, al momento de guardarlo, el nombre debe ser el mismo que el bloque tenía anteriormente. Otra forma es editando el mismo, tal como fue descrito en la sección Edición de nuevo bloque.

Para eliminar un bloque guardado, se debe arrastrar hacia el Bloque bandera y presionar sobre el botón Eliminar en el Menú "Mis bloques".

4.4. Implementación

En esta sección se presentan detalles de implementación de la aplicación MateFun Infantil.

El *Core* de la aplicación se encuentra implementado en Java, utilizando componentes nativos de la API de Android ("Android API reference", 2020). El resto de los módulos de la aplicación, presentados en la sección Arquitectura, se integran utilizando componentes de Android como *WebView* (permite integrar a una aplicación Android, elementos de una aplicación web) y *NDK* (permite utilizar desde la aplicación Android, bibliotecas de código escritas en otros lenguajes).

Para construir la aplicación fueron reutilizados desarrollos de proyectos de grado anteriores como es el caso del intérprete de MateFun y de los componentes de figuras 2D y 3D del IDE Web de MateFun. Estos desarrollos fueron modificados y adaptados para ser utilizados en la aplicación Android. Al pensar en desarrollos futuros o en el mantenimiento de la aplicación, se debe tener

en cuenta lo anterior. El código modificado del intérprete y el nuevo módulo de representación de figuras se mantienen en el repositorio de la aplicación Android, con lo necesario para funcionar en esta y con diferencias respecto a sus repositorios de origen las cuales serán abordadas en detalle en las siguientes secciones.

4.4.1. Integración y uso de Blockly

En el Capítulo 2, dentro de la sección Blockly, se realizó una introducción a las características básicas de este *framework*. A su vez, allí fue justificada su elección al comienzo del proyecto.

El objetivo de esta sección es describir en detalle las características de esta librería que fueron utilizadas, las modificaciones realizadas y aspectos de integración que tienen relevancia para entender el funcionamiento de la aplicación.

4.4.1.1. Integración con Android

Si bien Blockly tiene una biblioteca nativa para Android, la misma se encuentra sin soporte por parte de Google. El "Proyecto blockly-android disponible en Github" (2018) ya no es desarrollado activamente y la recomendación para nuevas aplicaciones es utilizar la versión web de Blockly. Pensando en el mantenimiento de la aplicación MateFun Infantil Android y en las posibles mejoras a futuro, fue acordado desde el inicio del proyecto seguir la recomendación e integrar la versión web de Blockly a la aplicación desarrollada. Esto se realiza a través de un *WebView*. Fueron seguidos los lineamientos del ejemplo "Blockly in an Android WebView" (2019).

Para acceder a funciones que provee Blockly, se debe llamar desde una clase Java (en el componente *Core*) a funciones implementadas en JavaScript que residen en el *WebView* de Blockly. Se amplía información en la sección Funciones de Blockly invocadas.

4.4.1.2. Bloques definidos

La funcionalidad clave de Blockly es definir bloques para integrar en una aplicación como lo es MateFun Infantil. A su vez, problemas relacionados a los bloques como son definir la forma, cómo se unen, chequeo de tipos en entradas, entre otros, ya están resueltos por la biblioteca.

Para crear un bloque se debe generar por un lado la definición del bloque (incluyendo sus entradas, salida, imagen asociada, color, etc); y por otro lado, al bloque se le debe asociar el código que retorna. En el caso de la aplicación creada, el código asociado a cada bloque se especifica en lenguaje MateFun.

Tanto la definición de los bloques como el código que retornan, se especifican en JavaScript. A modo de ejemplo, en la Figura 4.17 se puede ver la definición del bloque cuadrado. El código que retorna se especifica en la Figura 4.18.

```
Blockly.Blocks['block_cuad'] = {
  init: function() {
    this.appendDummyInput()
        .setAlign(Blockly.ALIGN_CENTRE)
        .appendField("Cuadrado");
    this.appendDummyInput()
        .setAlign(Blockly.ALIGN_CENTRE)
        .appendField(new Blockly.FieldImage("img/iconos_bloques/figuras2D/cuadrado.png"));
    this.appendValueInput("cuad_lado")
        .setCheck("Number")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField("lado");
    this.setInputsInline(false);
    this.setOutput(true, "Fig");
    this.setColour(color_figuras_2D);
  }
};
```

Figura 4.17: Definición de la estructura del bloque cuadrado.

```
Blockly.JavaScript['block_cuad'] = function(block) {
  var value_cuad_lado = Blockly.JavaScript.valueToCode(block, 'cuad_lado');

  var code = 'rect( ' + value_cuad_lado + ' , ' + value_cuad_lado + ' )';

  return [code, Blockly.JavaScript.ORDER_NONE];
};
```

Figura 4.18: Definición del código a retornar por el bloque cuadrado.

Los bloques pueden ser definidos directamente escribiendo el código JavaScript, o utilizando la herramienta "Blockly Factory" (s.f.). Durante el proyecto resulto más fácil codificarlos directamente en JavaScript.

Respecto a los bloques que fueron definidos, se puede ver el listado y su funcionalidad detallada en el "Manual de Usuario". Sin embargo, es importante destacar que algunas funcionalidades de MateFun fueron removidas y otras agregadas. Las decisiones fueron tomadas consultando informalmente a maes-

tros familiares y a su vez, apoyando las mismas en el "Programa de educación inicial y primaria" (2008).

En base a esto, se optó por ejemplo, por dejar las figuras 3D, introducir al Bloque interrogante y definir bloques con operaciones no disponibles directamente en MateFun (Cuadrado, Cono, modificaciones en mover, en la forma de definir un color, entre otros).

Por el contrario, fueron excluidas las funciones trigonométricas y la figura 3D anillo o toroide.

4.4.1.3. Funcionalidades utilizadas

Respecto a las funcionalidades de Blockly utilizadas, además del poder definir bloques, destacan las siguientes.

Blockly permite definir un *toolbox* (se corresponde con la Paleta de bloques en MateFun Infantil), donde los bloques definidos se agrupan por categorías. Es personalizable y los bloques se agregan al mismo editando un archivo XML, añadiendo el nombre del bloque creado previamente ("Blockly Toolbox", 2020). En la Figura 4.19 se puede ver un fragmento de la definición del *toolbox* de Blockly utilizado en MateFun. El resultado en la aplicación se muestra en la Figura 4.20.

```
<xml xmlns="https://developers.google.com/blockly/xml" id="toolbox" style="display: none">
  <category name="Números">
    <label text="Números:" web-class="labelStyle"/>
    <block type="block_numero"/>
    <label text="Operaciones:" web-class="labelStyle"/>
    <block type="block_suma"/>
    <block type="block_resta"/>
    <block type="block_multiplicacion"/>
    <block type="block_division"/>
  </category>
  ...
  <category name="...">
  ..
  </category>
</xml>
```

Figura 4.19: Fragmento de la definición del *toolbox* utilizado en MateFun.

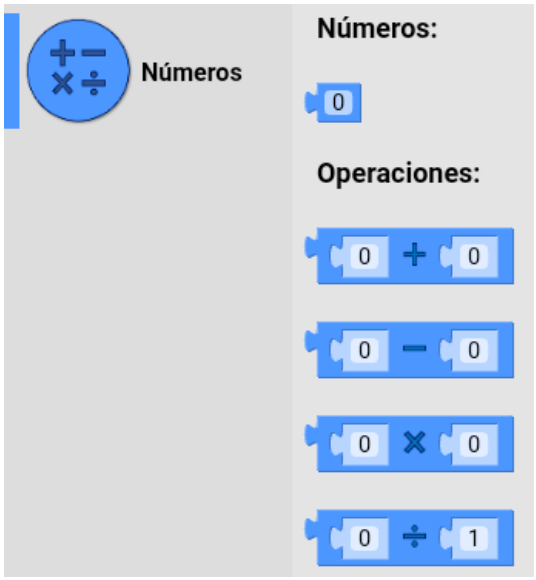

Figura 4.20: Categoría Números visualizada a partir del código de la Figura 4.19.

A su vez, para la sugerencia de valores en las entradas, se utilizó la funcionalidad "Blockly shadows" (2020). Los bloques sombra, aquellos que utilizan la funcionalidad mencionada, se agregan definiendo valores en el mismo XML del *toolbox* para las entradas que los utilicen.

Estos bloques permiten evitar usar bloques desde el *toolbox* para operaciones sencillas (por ejemplo una suma). Se comportan de la siguiente forma:

- El valor de un bloque sombra puede ser modificado directamente.
- En caso de cambiar el valor, el bloque sombra se transforma en un bloque convencional.
- Si se une otro bloque a la entrada donde se encuentra el bloque sombra, este es reemplazado por el bloque unido.

Otras funcionalidades utilizadas fueron el *zoom* y *scroll* sobre el *workspace* (o área de trabajo) y la papelera para borrar bloques. A su vez, se habilitó la funcionalidad de Blockly que genera sonidos al unir los bloques.

4.4.1.4. Modificaciones realizadas

Se realizaron modificaciones a Blockly respecto a lo que ofrecía en aspectos visuales. Entre ellas destacan los iconos agregados en el *toolbox*, ocultar componentes como las barras de *scroll* y los botones de zoom (las funcionalidades

se realizan con la pantalla táctil), cambios en colores y fuentes. Las modificaciones mencionadas se realizan sobrescribiendo estilos CSS de la librería.

En el documento "Manual del Desarrollador" se explica detalladamente el archivo en el cual se definen estos cambios.

4.4.1.5. Funciones de Blockly invocadas

Blockly cuenta con una API de funciones que son de utilidad al momento de crear una aplicación utilizando dicho *framework* ("Blockly JavaScript library APIs", 2018).

Entre las más relevantes, utilizadas en la aplicación creada, se encuentran las siguientes:

- **workspaceToCode**: permite generar el código en formato MateFun del conjunto de bloques que estén en el *workspace*.
- **getBlocksByType**: permite obtener bloques de un tipo específico. Se utiliza, por ejemplo, para obtener una referencia al Bloque bandera, la cual se utiliza luego para chequeos previos a la ejecución de un programa.
- **allInputsFilled**: se utiliza para comprobar que no existan bloques con entradas vacías al ejecutar un programa.
- **domToBlock**: permite obtener un bloque (y todos sus bloques conectados) a partir de un archivo XML.
- **blockToDomWithXY**: dado un bloque (y todos sus bloques conectados), devuelve el archivo XML correspondiente.

4.4.1.6. Plugins de terceros incorporados

Los *Plugins* de terceros incorporados a Blockly proveen de nuevas funcionalidades a MateFun Infantil, permitiendo mejorar la experiencia de usuario. Existen numerosos *Plugins* disponibles en la web para extender funcionalidades, algunos referenciados desde la documentación oficial de Blockly y otros disponibles en repositorios de terceros.

En el caso particular de MateFun Infantil, se utilizan dos *Plugins* desarrollados por "SPE Systemhaus GmbH" (2020):

- "Blockly Type Indicator" (2017): Consiste en un indicador que permite resaltar con color las entradas en las que un bloque puede ser conectado.
- "Blockly Shadow Autoreplacer" (2016): Automáticamente reemplaza los bloques de tipo *shadow* cuando son modificados con un valor.

SPE Systemhaus GmbH, dispone de un repositorio público en *GitHub* el cual contiene otros *Plugins* que pueden ser descargados e incorporados a proyectos que utilicen el *framework* Blockly.

4.4.2. Integración del intérprete MateFun

Integrar el intérprete de MateFun en la aplicación Android fue una de las tareas más difíciles enfrentadas en el transcurso del proyecto. A partir de los requerimientos y considerando cómo fue pensada la aplicación en conjunto con los tutores, era de gran importancia que el intérprete quedara integrado a la misma. Es decir, lo esperado desde el principio era lograr utilizarlo directamente en la aplicación, sin necesidad de, por ejemplo, consumir servicios de un servidor web o instalar *software* adicional en el dispositivo Android.

El intérprete de MateFun es usado para obtener el resultado de la ejecución de los programas generados con bloques. Luego que se tiene un programa, utilizando llamadas a funciones de Blockly (como fue explicado en la sección Funciones de Blockly invocadas), la aplicación obtiene la expresión (en lenguaje MateFun) a ser evaluada. Se trata de un *string* que debe ser comunicado al intérprete para poder ser evaluado, para luego obtener el resultado que será mostrado en la aplicación.

Como fue abordado en la sección Intérprete de MateFun, el mismo se encuentra implementado en Haskell. Utiliza a su vez varias bibliotecas, manejadas con el *software* Cabal, el cuál se encarga de descargar las dependencias y compilarlas según sea necesario.

4.4.2.1. Compilación cruzada

Los dispositivos Android utilizan tres arquitecturas de CPU básicas: arm, arm64 y x86 ("Tipos de procesador en Android", 2019). Integrar el código Haskell de MateFun con el resto de la aplicación Android implica en primer lugar, compilar el programa Haskell para las arquitecturas mencionadas desde una arquitectura diferente a la destino (generalmente desde x86_64).

Para poder realizar lo anterior, fue requerido un compilador cruzado. Se optó por utilizar el proyecto "Mobile haskell" (2018). Los binarios del compilador están disponibles en el sitio referenciado, sin embargo, su configuración no es trivial y se explica en el "Manual del desarrollador".

La compilación cruzada del código Haskell del intérprete fue especificada en un archivo makefile, definiendo la misma de forma tal que el resultado sean tres bibliotecas (una correspondiente a cada arquitectura de Android). Estas serán incluidas y accedidas por la aplicación como se describe en la sección Comunicación de Android con el intérprete.

4.4.2.2. Modificaciones al código del intérprete

Como punto de entrada a la biblioteca compilada del intérprete, se requiere una función que reciba un *string* con la expresión a ser evaluada por MateFun. Esta función debe retornar en otro *string* el resultado de evaluar la expresión que se le pasa por parámetro. No existía una función con estas condiciones implementada, por lo tanto tuvo que ser creada.

Se realizaron a su vez modificaciones para retornar un JSON en el caso de evaluar expresiones con figuras.

Las dos modificaciones descritas, junto con otros pequeños cambios en el código del intérprete, permitieron dejar de lado la comunicación por línea de comandos que este tenía implementada y hacer uso de la función creada.

En la práctica, el archivo MateFun.hs fue reemplazado por otro con el nombre MateFunAndroid.hs donde se efectuaron en gran parte las modificaciones mencionadas.

Las funcionalidades de internacionalización e intersección de dominios tuvieron que ser quitadas del intérprete para lograr su funcionamiento sobre Android. Esto debido a fallos en la compilación cruzada de bibliotecas utilizadas en estas funcionalidades y manejadas por Cabal. Debido a que se trataban de errores de bajo nivel del compilador, se optó en conjunto con los tutores por quitar las bibliotecas problemáticas y seguir adelante con la implementación.

En el repositorio del proyecto se incluye la versión modificada del intérprete de MateFun para funcionar con la aplicación Android.

4.4.2.3. Comunicación de Android con el intérprete

La comunicación de la aplicación (componente *Core*) con la biblioteca del intérprete, se apoya en primer lugar en la funcionalidad provista por el Kit de desarrollo nativo "NDK de Android" (s.f.). El NDK es un conjunto de herramientas que permiten implementar partes de la aplicación en código nativo mediante lenguajes como C y C++. Además, permite reutilizar bibliotecas de código escritas en estos lenguajes.

El NDK permite en el proyecto, incluir las liberías compiladas del intérprete de MateFun e importarlas desde un archivo .cpp. Desde este archivo se llama a la función creada en el intérprete de MateFun, tal como fue descrito en la sección Modificaciones al código del intérprete. Esta función está declarada como una "Foreign Function Interface (FFI)", 2020, lo que permite que los programas de Haskell cooperen con programas escritos en otros lenguajes, en este caso, con un programa escrito en C++.

Los eventos disparados por los botones, como por ejemplo el de ejecutar el programa creado con bloques, son "capturados" por una clase Java y allí se le asocia la lógica. Para invocar a un método dentro del archivo .cpp, escrito en C++ (desde donde puede invocarse a la FFI del intérprete MateFun), se utiliza "JNI" (2020). JNI es la interfaz nativa de Java, la cual define una forma para que el código escrito en Java interactúe con el código nativo (escrito en C/C++).

A modo de resumen, cuando el usuario presiona el Botón ejecutar, el evento *onClick* de Android es capturado por una clase Java. Allí se utiliza JNI para llamar a una función definida en C++ en un archivo .cpp dentro del proyecto. Desde C++, se importa la biblioteca previamente compilada del intérprete de MateFun, donde fue definida una *Foreign Function Interface*; la función escrita en C++ invocada desde Java llama a la FFI escrita en Haskell. Luego de evaluada la expresión en el intérprete, el camino para retornar el resultado es el mismo pero a la inversa; todas las funciones retornan un *string*.

En la Figura 4.21 se muestra gráficamente las interacciones que fueron descritas en los párrafos anteriores.

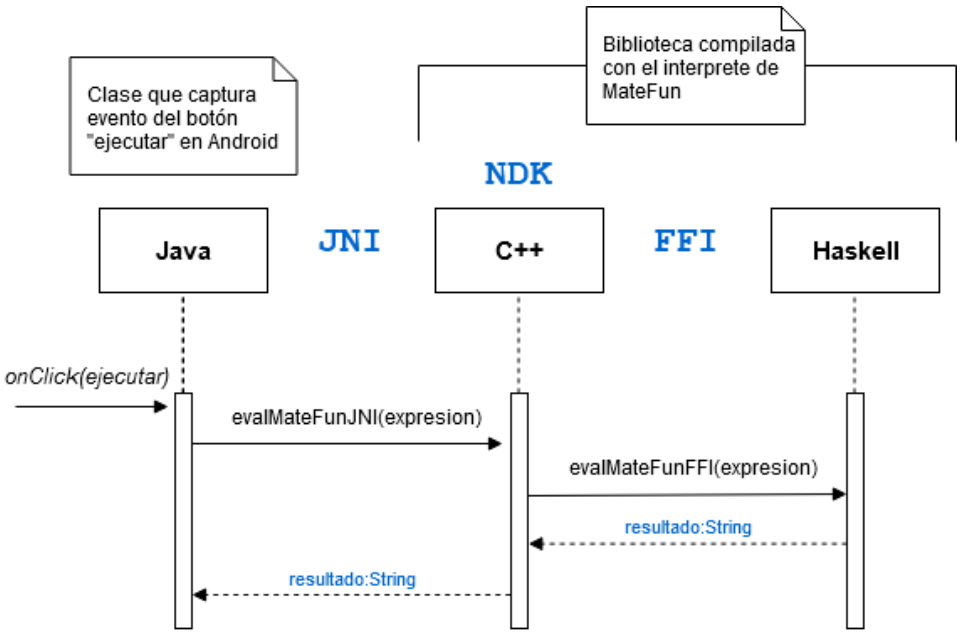

Figura 4.21: Integración con el intérprete de MateFun.

4.4.3. Integración de componentes para representar figuras

Para poder representar gráficamente figuras y animaciones en la aplicación Android, el punto de partida fue la Aplicación Web de MateFun. En el *frontend* de esta aplicación, realizada en Angular, ya se integraban las funcionalidades que permiten por ejemplo graficar figuras.

Un proyecto en Angular se estructura en varios componentes que conforman la aplicación. Un componente es un elemento reutilizable (Aristizabal, 2019).

En base a lo anterior, se optó por tomar los componentes que se encargaban de representar figuras 2D y 3D. Luego, fue generado un nuevo proyecto Angular donde se agregaron estos componentes, quitando elementos como botones y menús avanzados de personalización. A su vez, se agregaron imágenes para ilustrar la funcionalidad de animaciones y se modificó la barra que controla la velocidad (con el fin que se muestre de igual forma en animaciones 2D y en 3D, ya que en la versión web se muestran diferentes). Como ventaja, en cada uno de los componentes no fue necesario implementar nuevas funcionalidades en la lógica que empleaban.

Además de las modificaciones visuales, se implementó la lógica para llamar a un componente o a otro, y para elegir ciertos parámetros como el tamaño del gráfico. El punto de entrada al nuevo módulo de figuras es una función en

JavaScript; ésta, dependiendo de parámetros que recibe, se encarga de llamar a un componente o a otro y darle ciertas directivas para que la figura o animación se muestre como se espera.

Debido a las modificaciones realizadas, el mantenimiento de este módulo debe ser independiente a la evolución de los componentes Angular del IDE web de MateFun. Como desventaja, en caso de realizar una modificación en los componentes de gráficos de la aplicación web, no será impactada automáticamente en la aplicación Android de MateFun Infantil, y viceversa.

Sin embargo, adaptar los componentes Angular ya existentes tuvo como ventaja poder reutilizar el código y las bibliotecas estudiadas y probadas en proyectos anteriores de MateFun (Rey y col., 2019), reduciendo el tiempo empleado en investigación e implementación.

Los cambios y la forma en que fue realizado el nuevo proyecto en Angular contemplaron desde un primer momento que debía ser posible integrarlo a la aplicación Android. La forma de incluirlo es por medio de un *WebView*.

La aplicación se comunica con el módulo de figuras de la siguiente forma: una vez recibida la respuesta del intérprete con un resultado de tipo figura, se llama a la función del módulo de figuras utilizando funcionalidades que provee Android para comunicarse con un componente de tipo *WebView*. Android permite llamar desde una clase Java donde se maneja la lógica de la aplicación (componente *Core*) a la función JavaScript creada en el módulo de figuras.

Entre los parámetros con los que se invoca la función mencionada se encuentran: las dimensiones de la pantalla (permite adaptar el gráfico a este tamaño), si se trata de una figura o animación, si es 2D o 3D, y un JSON con la estructura de la figura a ser representada (esta es la respuesta que retorna el intérprete de MateFun).

Luego de invocada la función, la aplicación Angular se encarga de llamar al componente 2D o 3D. De ahí en más funciona de forma similar a como lo hace en el IDE Web de MateFun. Cada componente usa bibliotecas mencionadas en la sección Aplicación Web de MateFun, las cuales permiten representar las figuras o animaciones según corresponda y mostrarlas en pantalla.

4.4.4. Implementación del guardado de bloques

Desde el punto de vista de implementación, la funcionalidad de guardado de bloques tiene especial complejidad. Es a su vez, una de las funcionalidades

más importantes de la aplicación debido a su finalidad. Por lo anterior, en esta sección se presentan detalles de la implementación del guardado de bloques y funcionalidades relacionadas.

En la Figura 4.10, fue presentado el diagrama de comunicación para la funcionalidad de guardar bloques. Como puede apreciarse en la figura referenciada, el flujo comienza en el componente *Core*, desde allí se llama a la función obtenerBloques, creada en el componente Blockly.

Esta función es la encargada de obtener todos los datos que se necesitan para lograr la definición del nuevo bloque. Estos son:

- Nombre
- Icono
- Entradas y sus tipos
- Salida y su tipo
- Código resultante
- XML del programa
- Bloques guardados usados

El nombre, icono, entradas y salida, se utilizan para definir el bloque a guardar. El código resultante, es el código en formato MateFun que se le asocia a este bloque.

El XML del programa, se utiliza para asociar al nuevo bloque la secuencia de bloques que lo generó. Esta información es necesaria para la edición de un bloque guardado.

Por último, bloques guardados usados, lleva la trazabilidad de los bloques guardados que se utilizaron para generar el nuevo bloque. Esto sirve al momento de actualizar o eliminar un bloque guardado.

Estos datos son retornados en un JSON que llega al componente *Core*, y allí se pasa a la función que realiza la persistencia de los bloques.

4.4.4.1. Persistencia

En esta sección se describen los pasos necesarios para persistir los bloques creados por el usuario. Esta funcionalidad permite que dichos bloques estén disponibles luego de cerrar la aplicación y puedan ser utilizados una vez que el usuario vuelve a abrirla.

A partir de un programa creado con bloques, se debe obtener la definición del nuevo bloque y persistir la misma. Para esto, es necesario conocer la información de los bloques detallada al comienzo de la sección Implementación del guardado de bloques. A partir de estos datos, un bloque se define en formato JavaScript.

Una vez generado un nuevo bloque, este debe quedar disponible en la Paleta de bloques. Lo anterior se logra definiendo los nuevos bloques en archivos JavaScript (guardados en el sistema de archivos) que se importan en Blockly. Luego, se modifica el archivo XML donde se define la Paleta de bloques, agregando el nombre del nuevo bloque.

4.4.4.2. Edición, actualización y borrado de bloques guardados

Para la funcionalidad de edición, se guarda el XML de secuencia de bloques. Luego se utiliza una función llamada deplegarXml, que recorre los elementos del XML para formar el programa de bloques en el área de trabajo. La función fue adaptada del ejemplo "Blockly Pitch Fields" (s.f.).

En el caso de la actualización y borrado de bloques, para no agregar complejidad a la implementación, se impone la restricción de que no sea posible eliminar o actualizar un bloque que esta siendo utilizado en la composición de otro bloque guardado. Esto es, si se tiene un bloque guardado llamado "BloqueA", y luego se guarda un "BloqueB" formado con al menos un "BloqueA"; luego "BloqueA" no se podrá actualizar o eliminar.

Para el control descrito en el párrafo anterior, se utiliza la trazabilidad mencionada al comienzo de la sección Implementación del guardado de bloques. Se trata de un diccionario que tiene como clave el nombre de cada bloque guardado y los valores son los bloques que lo usan. Si el bloque a borrar o actualizar tiene otros bloques que lo usan, entonces no se permiten estas acciones.

4.4.4.3. Compartir bloques

Por último, es posible compartir los bloques que el usuario creó y quedaron guardados en su aplicación. Para esto se implementaron las funcionalidades de exportar e importar los bloques guardados.

Al momento de exportar, la aplicación genera un archivo "matefunBlocks.mf" que agrupa la información de los archivos JavaScript utilizados

en la implementación del guardado de bloques.

Luego de generado el archivo, se despliega una notificación de Android, que permite acceder al gestor de archivos para encontrar el archivo y así compartirlo por los medios que permita el dispositivo.

Para importar los bloques, se debe seleccionar un archivo "matefunBlocks.mf", lo cual hace que se escriba el contenido del mismo en los archivos encargados de la persistencia de lo nuevos bloques.

4.5. Gestión del proyecto

En esta sección se presentan en forma breve, aspectos relacionados a la gestión del proyecto. En particular, serán descritas las etapas del proyecto, el cronograma de las actividades realizadas, herramientas de gestión y metodología empleadas para el desarrollo de la aplicación entre otros aspectos.

4.5.1. Etapas en el proyecto

El proyecto fue planificado en las siguientes etapas:

1. Estado del arte, estudio de lenguajes de programación visual.
2. Estudio de MateFun.
3. Diseño del lenguaje visual.
4. Implementación de prototipos.
5. Estudio de herramientas y lenguajes de programación Android.
6. Diseño e implementación de la aplicación Android.
7. Finalización de la implementación y documentación del proyecto.

Las primeras cuatro etapas fueron realizadas sin grandes dificultades y acorde al cronograma inicial sugerido para el proyecto.

El estudio de herramientas y tecnologías relacionadas a programar en Android fue una etapa importante ya que ninguno de los integrantes del equipo contaba con experiencia desarrollando en esta plataforma.

Durante la etapa de implementación de prototipos fue detectado un problema para el que inicialmente no se tuvo una solución clara: ejecutar el intérprete de MateFun sobre el sistema operativo Android e integrarlo dentro de la propia aplicación. Debido a la complejidad del mismo y a la poca información disponible para resolverlo, el tiempo dedicado en la etapa de Diseño e implementación se extendió más de lo planificado.

Se realizaron documentos durante todo el transcurso del proyecto, aunque en la última etapa fue donde se dedicó mayor esfuerzo a las tareas referentes a la documentación.

Respecto a los *tests* realizados sobre las distintas funcionalidades de la aplicación, fueron llevados a cabo durante la implementación de la misma, a medida que nuevas funcionalidades eran liberadas. Sin embargo, durante la etapa final, se invirtió tiempo considerable para realizar pruebas de regresión y corregir errores encontrados.

En la Figura 4.22 se presenta un diagrama con la duración final de cada etapa.

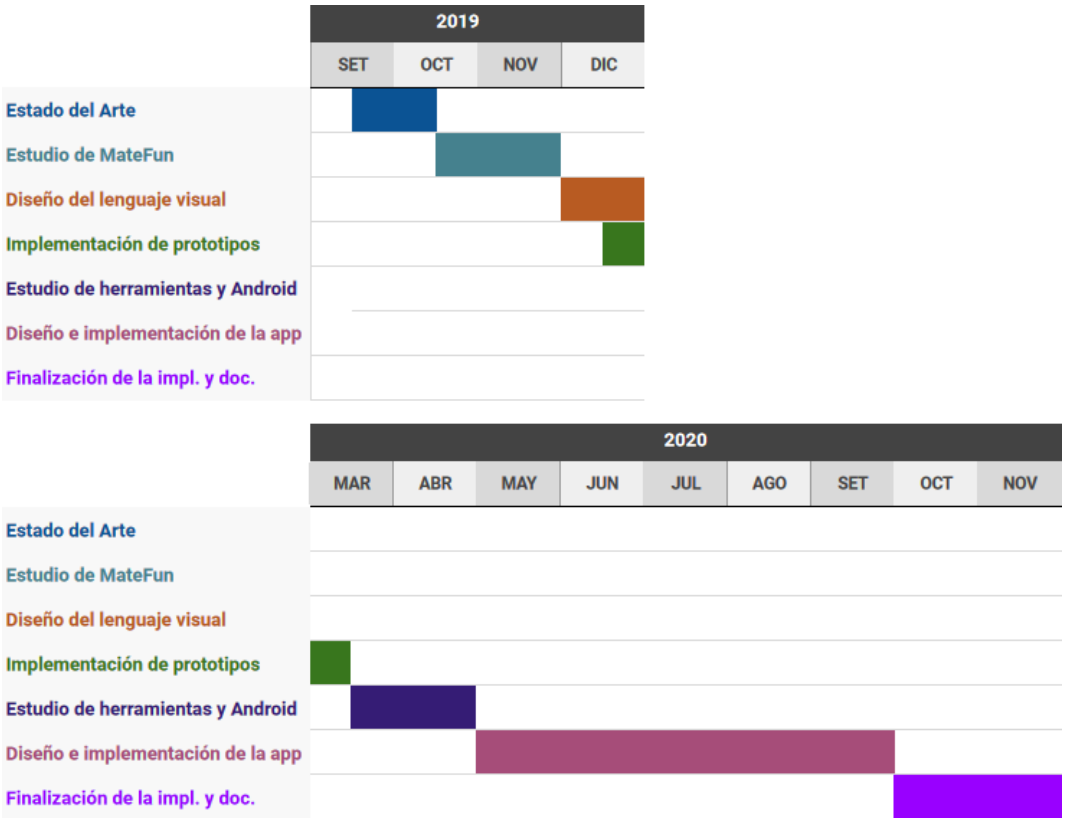

Figura 4.22: Avance del proyecto en el tiempo.

4.5.2. Metodología

Para el desarrollo de la aplicación, si bien no fue utilizado ningún *framework* en específico, se siguió un enfoque ágil. Tanto el relevamiento de requerimientos como la liberación de nuevas funcionalidades se realizaron de forma iterativa e incremental.

Las iteraciones fueron por lo general de dos semanas, realizando reuniones con los tutores al comienzo de cada iteración para mostrar los avances de implementación y relevar requerimientos. Luego de cada reunión con los tutores, se realizaba una reunión del equipo para planificar el trabajo de las próximas dos semanas. Por el contexto de pandemia durante el año 2020, las reuniones fueron vía "Zoom" (2020).

Además, las comunicaciones del equipo en el transcurso de las iteraciones fueron mediante "Whatsapp" (2020) y realizando videollamadas en caso de ser

necesario.

Para la organización de las tareas de implementación e investigación se utilizó un tablero en "Trello" (2020). Los requerimientos asociados a cada funcionalidad fueron escritos en tarjetas, las que luego eran asignadas a los integrantes del grupo en las reuniones de planificación.

Las tarjetas eran movidas entre distintas columnas, las cuales representaban su estado en ese momento. Para cada iteración se tenían las columnas *InProgress*, *Revisión* y *Done*, tal como se ve en la Figura 4.23.

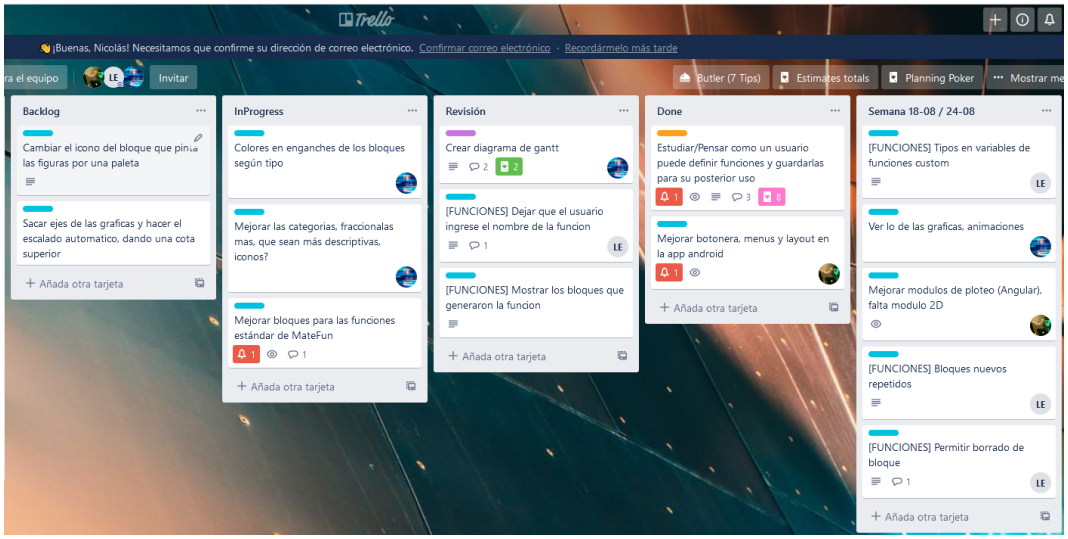

Figura 4.23: Uso de "Trello" (2020).

Luego se tenía una columna *backlog* con funcionalidades relevadas y pendientes por implementar.

El flujo de las tarjetas en Trello entre columnas, fue llevado a cabo de la siguiente forma: al momento de comenzar a implementar una nueva funcionalidad, la tarjeta asociada a esta era tomada del *backlog* por su responsable y pasada a *InProgress*. Al finalizar la misma, se pasaba a *Revision*. Por último, otro miembro del equipo (que no implementó la funcionalidad en cuestión), la debía validar y posteriormente, pasar la tarjeta a *Done*, indicando que la tarea de implementación fue completada. En caso de no pasar la validación, la tarjeta se retornaba a la columna *InProgress*, con una nota indicando porque no pasó la revisión.

4.5.3. Otras herramientas utilizadas

Además de las herramientas ya mencionadas, se destacan el uso de las siguientes:

- "GitLab" (2020) como gestor de repositorio y manejo de *issues*.
- "Overleaf" (2020) para redacción de documentos.
- "Gmail" (2020) para intercambio de mails entre tutores y el equipo.
- "Google Drive" (2020) para compartir archivos y guardar documentos no oficiales.
- "Android Studio" (2020) como IDE utilizado para el desarrollo de la aplicación.
- "Icons8" (2020) fue el sitio desde donde se tomaron los iconos para los bloques guardados.

Capítulo 5

Testing de la aplicación

En este capítulo, se presentan las pruebas realizadas sobre MateFun Infantil y se analizan los resultados obtenidos. Se describen las sesiones de *testing* exploratorio, la forma de reportar y corregir los bugs encontrados durante las mismas. Por último, se comentan las pruebas no funcionales y se discuten los resultados.

5.1. Testing exploratorio

Según explican Lamancha y col. (2007), el *testing* exploratorio es un enfoque para la realización de pruebas de *software* que se caracteriza por la ejecución en forma simultánea del aprendizaje, diseño y ejecución de los casos de pruebas sobre la aplicación. Al realizar *testing* exploratorio, se diseñan y ejecutan las pruebas con un objetivo concreto.

Al hacer *testing* exploratorio, se debe:

1. Tener un objetivo al que se quiere alcanzar luego de finalizada la sesión (en el caso de MateFun Infantil, probar el funcionamiento de la aplicación tomando como referencia los Casos de uso).
2. Establecer un tiempo límite dedicado a la sesión de *testing*.

La estructura externa del *testing* exploratorio es sencilla. En el transcurso de la sesión, el *tester* interactúa con la aplicación teniendo como objetivo el cumplimiento de una misión. Una misión consiste en probar diferentes funcionalidades de la aplicación considerando los tipos de incidentes que se buscan y los riesgos involucrados.

Para el caso particular del proyecto MateFun Infantil, el *testing* exploratorio fue realizado por los integrantes del equipo con sesiones de duración de 30 minutos. Las misiones fueron definidas considerando la sección de Casos de uso y para cada una de ellas, se determinó la cantidad de sesiones, dependiendo de la complejidad y que tan crítico es el caso de uso.

Un listado con las misiones definidas en MateFun Infantil puede observarse en la Tabla 5.1. La primer columna de la tabla corresponde a las misiones creadas a partir de los Casos de uso, mientras que la segunda indica el número de sesiones llevadas a cabo para probar el funcionamiento de los mismos.

Considerando los Requerimientos no funcionales, era deseable que el sistema fuese capaz de ejecutar en las "Tablets de Plan Ceibal" (s.f.). Debido a que el equipo no contaba con un dispositivo físico, se decidió realizar la ejecución de las misiones en un emulador con características similares a las soportadas por los dispositivos de Plan Ceibal. El *hardware* asignado al dispositivo del emulador corresponde a la "Tablet T8 U800 B" (s.f.), la cual presenta las siguientes características:

- Sistema operativo: Android 8.1.0
- Memoria RAM: 2 GB
- Tamaño de pantalla: 8 pulgadas
- Resolución: 1280 x 800
- Almacenamiento interno: 16 GB

Luego de realizadas cada una de las misiones fueron detectados errores, para los cuales se realizó el Reporte de errores mediante herramienta Issues de GitLab.

5.1.1. Reporte de errores mediante herramienta Issues de GitLab

"GitLab" (2020) es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Además de ser un gestor de repositorios, ofrece la posibilidad de reportar problemas detectados en una aplicación mediante los denominados *Issues*. Los *Issues* poseen un estado, el cual puede ser: *abierto*, en cuyo caso aún no se ha resuelto, o *cerrado*, lo que significa que ha sido corregido.

Misión	**Cant. de sesiones**
Crear programa con bloques	6
Ejecutar programa creado	6
Crear programa con resultado numérico	4
Crear figuras 2D	3
Crear figuras 3D	3
Crear animaciones 2D	4
Crear animaciones 3D	4
Ocultar/desplegar figuras o animaciones	3
Duplicar bloques del *workspace*	1
Borrar bloques del *workspace*	2
Guardar programa creado	4
Eliminar bloque guardado	2
Mostrar programa de un bloque guardado	3
Exportar bloques guardados	4
Importar bloques guardados	4

Tabla 5.1: Misiones de *testing* exploratorio.

Al realizar *testing* exploratorio, los integrantes del equipo detectaron errores y realizaron mejoras en la aplicación. Al poder tener visibilidad global de los problemas encontrados por el equipo, antes de agregar un nuevo *Issue*, se verificaba que no haya sido reportado por otro integrante, evitando de esta manera la duplicación de los mismos.

Al reportar un nuevo error en la aplicación por alguno de los integrantes, se acordó que el mismo debía contener un título descriptivo, una descripción detallada del problema encontrado, los pasos para reproducirlo y opcionalmente, una imagen o video. De esta manera, la corrección de los *Issues* era realizada por algún integrante del equipo y una vez resueltos, mediante *testing* cruzado, se verificaban que los mismos fueran corregidos. En la Figura 5.1, se observan todos los *Issues* reportados por el equipo.

Se definieron algunas etiquetas para indicar el tipo de error encontrado (como por ejemplo: "Error funcional" para errores de tipo funcional o "Mejora interfaz" para mejoras visuales). Algunos incidentes no pudieron ser reproducidos por otros integrantes del equipo, por lo que temporalmente algunos *Issues* fueron asignados con la etiqueta "No reproducido". Una vez corregido el problema y revisado por otro integrante del equipo, se asignaba la etiqueta "Revisado" para finalmente cambiar a estado "Cerrado" por quien lo reportó originalmente, indicando de este modo que el mismo había sido correctamente solucionado y revisado.

5.1.2. Corrección de *Issues* en MateFun

Luego de realizadas las misiones, se detectaron un total de 31 errores en la aplicación, distribuidos de la siguiente manera:

- Mensajes desplegados: 9 defectos
- Eliminar bloques: 6 defectos
- Representación de figuras: 6 defectos
- Guardar bloques: 4 defectos
- Interfaz de usuario: 3 defectos
- Ejecutar programa: 2 defectos
- Exportar bloques: 1 defecto

Todos los errores reportados mediante *testing* exploratorio fueron solucionados. En etapas posteriores, se verificó que el comportamiento de las misiones

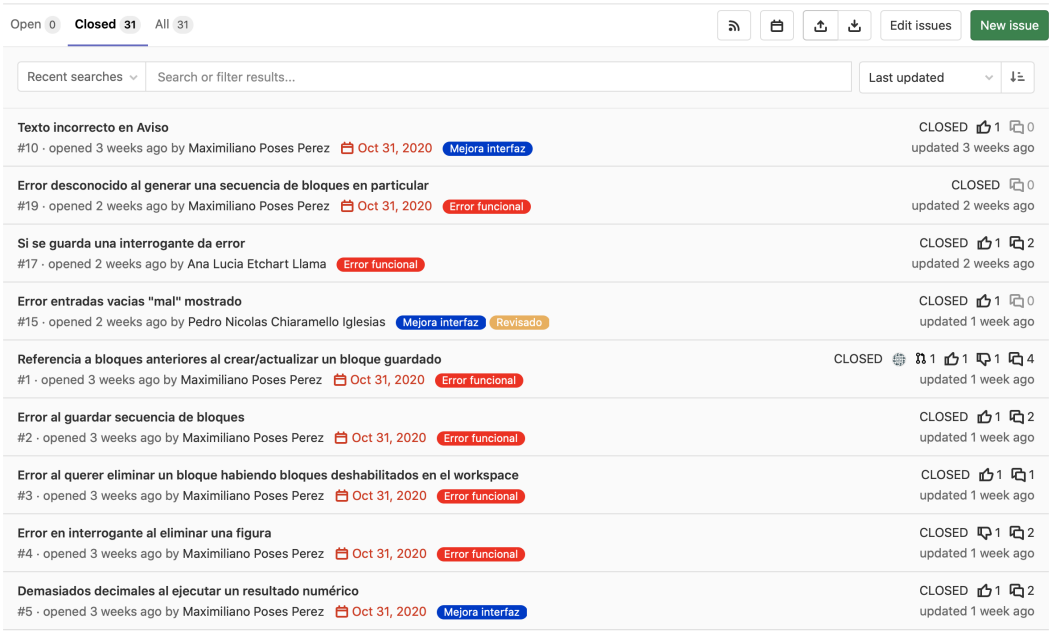

Figura 5.1: *Issues* reportados utilizando GitLab.

en su conjunto fuera el correcto.

5.2. Pruebas no funcionales

La calidad de software refiere, entre otros aspectos, a la calidad de la programación y el diseño, los cuales impactan principalmente en los costos de mantenimiento de un producto. Su utilidad radica en la posibilidad de realizar un diagnóstico rápido y de bajo coste de los componentes del sistema, permitiendo evaluar tanto la mantenibilidad como la escalabilidad del mismo.

Para el caso particular de MateFun Infantil, era deseable utilizar herramientas que permitieran evaluar la calidad de software desarrollado utilizando métricas que fueran automatizables y de bajo coste, a modo de obtener una visión general de los diferentes componentes de la aplicación.

5.2.1. Complejidad ciclomática

La complejidad ciclomática es una métrica del *software* que es utilizada en ingeniería de software para medir la complejidad de un programa, siendo posible calcularla con respecto a módulos, clases, métodos y funciones (McCabe, 1976).

En el caso del proyecto MateFun Infantil, fue utilizada la herramienta "MetricsReloaded" (2020), la cual puede ser descargada e integrada con Android Studio. Incorpora una gran cantidad de métricas sobre el código fuente, muchas de las cuales no serán consideradas en este proyecto debido al alcance.

Al calcular la complejidad ciclomática de un fragmento de código, se puede determinar el riesgo que este supone utilizando los rangos definidos en la Tabla 5.2, cuya primer columna corresponde al valor de la complejidad ciclomática y la segunda al riesgo asociado:

v(G)	**Riesgo**
1-10	Programa simple, sin mucho riesgo
11-20	Más complejo, riesgo moderado
21-50	Complejo, programa de alto riesgo
50	Programa no testeable, muy alto riesgo

Tabla 5.2: Evaluación de riesgo según complejidad ciclomática.

A partir del análisis realizado por expertos en diferentes proyectos de software, se encontró que un valor 10 es un límite superior práctico para el tamaño de un módulo. Cuando la complejidad supera dicho valor se hace muy difícil probarlo, entenderlo y modificarlo. La limitación deliberada de la complejidad en todas las fases del desarrollo ayuda a evitar los problemas asociados a proyectos de alta complejidad (Cardacci, 2016).

En el caso particular de MateFun Infantil, los Resultados de la complejidad ciclomática en los métodos en el sistema arrojan, en promedio, un valor de **3.62**. Considerando lo anterior, parece indicar que MateFun Infantil es un programa con baja complejidad asociada, lo que permite hacerlo mantenible y simple de testear.

5.2.2. LCOM - Métrica de falta de cohesión en los métodos

La métrica de Falta de Cohesión en los Métodos o LCOM (*Lack of Cohesion*), es una medida de lo cohesionada que es una clase, a partir del número de atributos comunes usados por diferentes métodos de la misma. En otras

palabras, se refiere al grado en que los elementos de un módulo permanecen juntos (Yourdon y Constantine, 1979).

En la programación orientada a objetos, si los métodos que sirven a una clase tienden a ser similares en muchos aspectos, entonces se dice que la clase tiene una alta cohesión. En un sistema altamente cohesivo, la legibilidad y reusabilidad del código es mayor, mientras que la complejidad se mantiene manejable (Hitz y Montazeri, 1995). La cohesión es mayor si:

- Las funcionalidades embebidas en una clase, accedidas a través de sus métodos, tienen mucho en común.
- Los métodos realizan un pequeño número de actividades relacionadas.

Para analizar la cohesión en MateFun se utilizaron las métricas de Chidamber y Kemerer (1994), en particular, la métrica LCOM. La métrica, tiene como referencia un puntaje con las siguientes características:

- El valor de 1 indica una clase cohesiva, lo que es considerado correcto.
- Un valor de 2 o superior indica que la clase debería ser dividida en clases más pequeñas. Suele indicar un problema.
- El valor de 0 indica que no hay métodos en la clase. Esto es considerado una mala práctica.

Para el caso de MateFun Infantil, a partir de los Resultados de LCOM en las clases se obtiene un valor promedio de **1.86**, lo que indica que el sistema posee alta cohesión. La distribución de las clases de acuerdo al valor obtenido es de la siguiente manera:

- Valor de 0: 2 clases
- Valor de 1: 9 clases
- Valor mayor o igual a 2: 3 clases

Analizando los resultados, se encontró que las clases cuyo valor es 0 corresponden a enumerados (Enums.MessagesEnum y Enums.ErrorEnum). Son clases definidas de manera centralizada que contienen un conjunto de constantes, cuyos valores son utilizados para el despliegue de errores y mensajes hacia el usuario. Por dicho motivo, se considera importante no eliminar ni unir dichas clases con otras como sugiere la métrica. La mayor parte de las clases definidas arrojan un valor de 1, lo que indica un resultado óptimo. Los módulos con un

resultado mayor a 2 corresponden a *WebChromeClient*, *GraphHelper* y *MainActivity2*. Separar las clases anteriores en clases más pequeñas podría generar errores en la aplicación e implicaría un riesgo, dado que son módulos críticos encargados de la representación gráfica heredados de proyectos anteriores, de capturar eventos ocurridos en la aplicación y del manejo de la interacción entre la aplicación Android y Blockly. Por este motivo, se decide no separar dichas clases. Tener módulos con baja cohesión implica nuevos riesgos y comportamientos indeseados en la aplicación, así como también problemas en la mantenibilidad y reutilización de dichos módulos.

Cabe destacar que un sistema con alta cohesión resulta en un sistema más robusto, fiable, reutilizable y mejor comprendido.

Capítulo 6

Conclusiones y trabajo futuro

Este capítulo tiene como objetivo detallar las conclusiones obtenidas y el trabajo a futuro.

6.1. Conclusiones

El resultado obtenido extiende los trabajos realizados en otros proyectos de grado a partir de MateFun.

El aporte principal de este proyecto consiste en el desarrollo de una aplicación Android orientada a niños de educación primaria, que utiliza la potencialidad de MateFun a través de un lenguaje visual basado en bloques.

En este sentido, fue desarrollada una aplicación que permite utilizar el lenguaje MateFun a partir de una interfaz visual, en su mayoría icónica. La misma puede ser utilizada para crear y evaluar funciones matemáticas, permitiendo al usuario abstraer la sintaxis del lenguaje para enfocarse en la lógica del programa a construir. Además de lo anterior, es posible visualizar figuras y animaciones tanto en 2D como en 3D, las cuales pueden ser de utilidad para reforzar conceptos contenidos en el "Programa de educación inicial y primaria" (2008).

Adicionalmente, se generaron documentos que se espera sean de utilidad para comprender el funcionamiento de la aplicación. Además de lo anterior, se incluye documentación técnica que permite extender el desarrollo de la aplicación en caso de ser necesario. Dentro de los mismos se encuentran el "Manual de Usuario", el "Documento de análisis y diseño" y el "Manual del Desarrollador".

La investigación realizada en el marco de este trabajo, incluye el conocimiento generado respecto a la ejecución del intérprete de MateFun en Android. Se logró compilar (desde una arquitectura diferente a la de destino) un programa escrito en Haskell para que funcione en dispositivos Android con arquitecturas ARM y x86. Este punto fue un gran desafío presentado en el transcurso del proyecto, debido a la escasa documentación existente.

Cabe destacar que resultaron de especial importancia los aportes realizados por maestros en diferentes etapas del proyecto, quienes con su experiencia y visión crítica, sugirieron mejoras que fueron consideradas por el equipo en el producto final. En este sentido, se concluye que sería de gran utilidad incluir a maestros y alumnos directamente en el desarrollo de aplicaciones similares a MateFun Infantil. En concreto (pensando en una metodología similar a la utilizada), deberían ser partícipes desde el comienzo, participando del relevamiento de requerimientos y de las *demo* o instancias donde se liberen nuevas versiones de la aplicación para que puedan aportar su visión sobre las funcionalidades y aspectos de interfaz. De esta forma, se tendrían respuestas mas precisas a las siguientes cuestiones: ¿Esta funcionalidad desarrollada puede ser usada por niños?, ¿La interfaz desarrollada es comprendida por un niño?, ¿Cuál es el nombre más apropiado para determinado concepto?

6.2. Trabajo futuro

En primer lugar sería conveniente realizar un trabajo interdisciplinario contando con docentes, integrantes del Instituto de Computación (InCo) y otras personas que puedan acercar MateFun Infantil a las aulas. Este trabajo debería ir de la mano con un análisis minucioso del "Programa de educación inicial y primaria" (2008). El objetivo sería evaluar la aplicación actual, adaptarla e incorporar funcionalidades que sean consideradas necesarias.

Como ya se ha indicado, tomando como base el contenido del programa de educación inicial, se propone agregar bloques que representen figuras, tales como triángulo y pirámide.

A su vez, podría evaluarse la posibilidad de renombrar las categorías existentes en la barra de herramientas para hacerlas consistente con el contenido del programa en cuestión (por ejemplo, cambiar el nombre de la categoría Figuras 2D a "Figuras en el plano" o Figuras 3D a "Figuras en el espacio") tanto en la aplicación como en la documentación generada.

Es importante tener en cuenta que a futuro la aplicación podría extenderse para ser usada en edades aún más tempranas. Por este motivo, sería necesario reducir la cantidad de texto presente en la interfaz y desplegar tanto advertencias como errores directamente sobre los bloques.

Se sugiere entonces, evaluar la opción de agregar niveles de dificultad en la aplicación. Para cada nivel, se espera que sean visualizados aquellos bloques que permitan comprender conceptos trabajados según el año escolar.

Podría ser de utilidad además incluir alguna forma de recolección de datos de uso de la aplicación, con el fin de conocer en profundidad la interacción de los usuarios con la herramienta (tiempo de uso de sesión, bloques usados, errores cometidos, entre otros).

Sería conveniente llevar a cabo una validación del uso de la aplicación con niños, mediante una experiencia en el aula. Teniendo en cuenta las características deseables analizadas en el capítulo Estado del Arte, podría ser de utilidad incorporar un personaje que guíe en la aplicación. Al mismo tiempo, se cree conveniente la posibilidad de integrar tutoriales paso a paso que guíen la secuencia didáctica.

Finalmente, tanto Blockly como los componentes de Android utilizados en la aplicación permiten ser modificados y adaptados fácilmente. Tomando en cuenta lo anterior, se espera que sea aprovechada la flexibilidad que ofrecen estas tecnologías para mejorar la presentación visual de la aplicación, aspecto de suma importancia teniendo en cuenta el público objetivo. Además de estas mejoras, como se mencionó en Integración y uso de Blockly, sería interesante lograr una comunicación desde Blockly hacia el *Core* de la aplicación para poder realizar acciones según el estado de los bloques, como por ejemplo, habilitar solamente el Botón ejecutar en caso de que el programa este bien formado.

Referencias

Acerca de ScratchJr. (s.f.). Recuperado el 17 de diciembre de 2019, desde https://www.scratchjr.org/about/info

Álgebra: aportes para nuevas reflexiones. (2009). Recuperado el 1 de noviembre de 2020, desde https://fumtep.edu.uy/aportes-para-la-reflexion-docente/item/download/350_e408243b082db6c1b5c98dc357d03479

Android API reference. (2020). Recuperado el 6 de diciembre de 2020, desde https://developer.android.com/reference

Android Studio. (2020). Recuperado el 15 de noviembre de 2020, desde https://developer.android.com/studio

Aristizabal, V. (2019). *Componentes en Angular*. Recuperado el 24 de octubre de 2019, desde https://medium.com/@vanessamarely/componentes-en-angular-f25138b00c83

Blockly Factory. (s.f.). Recuperado el 15 de noviembre de 2019, desde https://blockly-demo.appspot.com/static/demos/blockfactory/index.html

Blockly Generator. (s.f.). Recuperado el 20 de noviembre de 2019, desde https://blockly-demo.appspot.com/static/demos/index.html

Blockly in an Android WebView. (2019). https://github.com/google/blockly/tree/develop/demos/mobile/android

Blockly JavaScript library APIs. (2018). Recuperado el 6 de noviembre de 2020, desde https://developers.google.com/blockly/reference/overview#javascript_library_api

Blockly Overview. (2020). Recuperado el 20 de noviembre de 2019, desde https://developers.google.com/blockly/guides/overview

Blockly Pitch Fields. (s.f.). Recuperado el 15 de noviembre de 2019, desde https://blockly-demo.appspot.com/static/demos/custom-fields/pitch/index.html

Blockly Reference. (s.f.). Recuperado el 20 de noviembre de 2019, desde https://developers.google.com/blockly

Blockly Shadow Autoreplacer. (2016). Recuperado el 14 de noviembre de 2020, desde https://github.com/SPE-Systemhaus/blockly-shadow-autoreplacer

Blockly shadows. (2020). Recuperado el 5 de noviembre de 2020, desde https://developers.google.com/blockly/guides/configure/web/toolbox#shadow_blocks

Blockly Toolbox. (2020). Recuperado el 5 de noviembre de 2020, desde https://developers.google.com/blockly/guides/configure/web/toolbox

Blockly Type Indicator. (2017). Recuperado el 14 de noviembre de 2020, desde https://github.com/SPE-Systemhaus/blockly-type-indicator

Boeijink, A. (2016). *Experimenting with an interactive visualfunctional programming environment*. Recuperado el 23 de noviembre de 2019, desde https://raw.githubusercontent.com/viskell/viskell/master/viskell-nlfpday.pdf

Cameto, G. & Méndez, M. (2017). MateFun (IDE Web + Lenguaje Funcional Específico). Proyecto de grado. *Universidad de la República (Uruguay). Facultad de Ingeniería*.

Cardacci, D. G. (2016). REFACTORIZACIÓN DE CÓDIGO Y CONSIDERACIONES SOBRE LA COMPLEJIDAD CICLOMÁTICA. *Documentos de Trabajo*, (592), 1-8. http://proxy.timbo.org.uy/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=asn&AN=117994974&lang=es&site=eds-live

CBSNews. (2014). *Coding for kindergarteners: App teaches kids computer basics*. Recuperado el 17 de diciembre de 2019, desde https://www.cbsnews.com/news/scratch-jr-app-teaches-kindergarteners-the-basics-of-computer-coding/

Chidamber, S. R. & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, *20*(6), 476-493. https://doi.org/10.1109/32.295895

CICEA. (2020). *Centro Interdisciplinario en Cognición para la Enseñanza y el Aprendizaje*. Recuperado el 11 de noviembre de 2020, desde http://www.cicea.ei.udelar.edu.uy/

craft.ai. (2015). *The maturity of visual programming*. Recuperado el 1 de noviembre de 2019, desde https://www.craft.ai/blog/the-maturity-of-visual-programming

ElObjetivo. (2019). *Scratch 3.0! Es hora de enseñar a los niños a programar*. https://elobjetivo.com.ar/contenido/4766/scratch-30-es-hora-de-ensenar-a-los-ninos-a-programar

Foreign Function Interface (FFI). (2020). Recuperado el 18 de octubre de 2019, desde https://wiki.haskell.org/Foreign_Function_Interface

GitLab. (2020). Recuperado el 10 de noviembre de 2020, desde https://about.gitlab.com

Gmail. (2020). Recuperado el 15 de noviembre de 2020, desde https://www.gmail.com/

Google Drive. (2020). Recuperado el 15 de noviembre de 2020, desde https://drive.google.com/

Harvey, B. (s.f.). *HomePage for Brian Harvey*. Recuperado el 27 de noviembre de 2019, desde https://people.eecs.berkeley.edu/~bh/

Hitz, M. & Montazeri, B. (1995). Measuring coupling and cohesion in object-oriented systems.

Icons8. (2020). Recuperado el 15 de noviembre de 2020, desde https://icons8.com/

JNI. (2020). Recuperado el 18 de octubre de 2019, desde https://developer.android.com/training/articles/perf-jni?hl=es

Jost, B., Ketterl, M., Budde, R. & Leimbach, T. (2014). Graphical Programming Environments for Educational Robots: Open Roberta - Yet Another One?, En *2014 IEEE International Symposium on Multimedia*. https://doi.org/10.1109/ISM.2014.24

Lamancha, B. P., Pittier, A., Travieso, M. & Wodzislawski, M. (2007). Testing Exploratorio en la Práctica, En *JIISIC*.

LogoFoundation. (2014). *Logo Programming*. Recuperado el 27 de noviembre de 2019, desde https://el.media.mit.edu/logo-foundation/what_is_logo/logo_programming.html

Marji, M. (2014). Learn to Program with Scratch. *No Starch Press*, 1-9, 13-15.

Matefun Interprete Web [Online: Accedido 14/11/2020]. (s.f.). https://matefun.math.psico.edu.uy/#/matefun

McCabe, T. J. (1976). A Complexity Measure. *IEEE Transactions on Software Engineering*, *SE-2*(4), 308-320. https://doi.org/10.1109/TSE.1976.233837

MetricsReloaded. (2020). Recuperado el 14 de noviembre de 2020, desde https://plugins.jetbrains.com/plugin/93-metricsreloaded

MIT. (s.f.). *Scratch - Imagine, Program, Share*. Recuperado el 20 de noviembre de 2019, desde https://scratch.mit.edu

Mobile haskell. (2018). Recuperado el 18 de octubre de 2019, desde http://hackage.mobilehaskell.org/

Mönig, J. & Harvey, B. (2020). *Snap! online*. Recuperado el 27 de noviembre de 2019, desde https://snap.berkeley.edu/snap/snap.html

NDK de Android. (s.f.). Recuperado el 18 de octubre de 2019, desde https://developer.android.com/ndk

Overleaf. (2020). Recuperado el 15 de noviembre de 2020, desde https://www.overleaf.com/

PolyUp. (2020). Recuperado el 16 de diciembre de 2019, desde https://www.polyup.com/

Programa de educación inicial y primaria. (2008). Recuperado el 28 de octubre de 2020, desde https://www.ceip.edu.uy/documentos/normativa/programaescolar/ProgramaEscolar_14-6.pdf

programación por capas. (2020). Recuperado el 3 de octubre de 2019, desde https://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas

Proyecto blockly-android disponible en Github. (2018). Recuperado el 4 de noviembre de 2020, desde https://github.com/google/blockly-android

Proyecto Butiá. (2016). Recuperado el 27 de noviembre de 2019, desde https://www.fing.edu.uy/inco/proyectos/butia/

Repenning, A. (2017). Moving Beyond Syntax: Lessons from 20 Years of Blocks Programing in AgentSheets. *Journal of Visual Languages and Sentient Systems*, *3*, 68-91. https://doi.org/10.18293/VLSS2017-010

Repositorio en GitHub de Pilas Bloques. (2019). Recuperado el 10 de diciembre de 2019, desde https://github.com/Program-AR/pilas-bloques/tree/master

Repositorio en GitHub de ScratchJr. (2019). Recuperado el 17 de diciembre de 2019, desde https://github.com/LLK/scratchjr

Repositorio en GitHub de Snap. (2019). Recuperado el 27 de noviembre de 2019, desde https://github.com/jmoenig/Snap

Repositorio en GitHub de Viskell. (2017). Recuperado el 23 de noviembre de 2019, desde https://github.com/viskell/viskell

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. & Ka-

fai, Y. (2009). Scratch: Programming for All. *Commun. ACM*, *52*(11), 60-67. https://doi.org/10.1145/1592761.1592779

Rey, D., Fagian, I. & Rosano, L. (2019). Representación gráfica interactiva de funciones matemáticas, figuras geométricas y animaciones en la Web. Proyecto de Grado [Online: Accedido 31/10/2020]. *Universidad de la República (Uruguay). Facultad de Ingeniería*. https://gitlab.fing.edu.uy/matefun/Frontend/-/wikis/uploads/4a9f5d4a3ee140511353e54f4f27e1c4/Matefun__1_.pdf

SPE Systemhaus GmbH. (2020). Recuperado el 14 de noviembre de 2020, desde https://www.spe-systemhaus.de/

Sugar Labs. (s.f.). Recuperado el 27 de noviembre de 2019, desde https://www.sugarlabs.org

Tablet T8 U800 B. (s.f.). Recuperado el 14 de noviembre de 2020, desde https://www.ceibal.edu.uy/es/articulo/hardware-tablet-t8-u800_b

Tablets de Plan Ceibal. (s.f.). Recuperado el 14 de noviembre de 2020, desde https://www.ceibal.edu.uy/es/dispositivos/tablets

Taller de Introducción a la Computación. (2020). Recuperado el 10 de noviembre de 2020, desde https://eva.fing.edu.uy/course/view.php?id=1398&section=0

Tipos de procesador en Android. (2019). Recuperado el 18 de octubre de 2019, desde https://elandroidefeliz.com/identificar-tipo-de-procesador-en-android/

TortuBots. (2019). Recuperado el 27 de noviembre de 2019, desde https://www.fing.edu.uy/inco/proyectos/butia/mediawiki/index.php/TortuBots

Trello. (2020). https://trello.com/es

Turtle Blocks. (2014). Recuperado el 27 de noviembre de 2019, desde http://wiki.sugarlabs.org/go/Activities/Turtle_Art

Vázquez, N. (2019). Mejoras al intérprete MateFun. Proyecto de grado. *Universidad de la República (Uruguay). Facultad de Ingeniería*.

Web de Pilas Bloques. (2019). Recuperado el 10 de diciembre de 2019, desde http://pilasbloques.program.ar/

Whatsapp. (2020). https://www.whatsapp.com/?lang=es

Yourdon, E. & Constantine, L. L. (1979). *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design* (1st). USA, Prentice-Hall, Inc.

Zoom. (2020). https://zoom.us/

Glosario

API Interfaz de programación de aplicaciones. Conjunto de requerimientos que permite la transferencia de datos entre aplicaciones. 9

API REST API que permite la comunicación con estándares para arquitecutras REST. 21

Android Sistema operativo móvil basado en núcleo Linux y otros software de código abierto. 2, 6, 14, 15, 16, 30, 33, 34, 42, 47, 48, 49, 50, 51, 54, 55, 60, 66, 67

Android Studio Entorno de desarrollo integrado (IDE) oficial para la plataforma Android. 64

Angular *Framework* para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página 19, 50, 51

Animación, matefun Refiere a una acción permitida, que genera una secuencia de gráficas. Para más detalle ver **Manual de Usuario** . 31, 32, 51

Blockly Biblioteca del lado del cliente para el lenguaje de programación JavaScript para crear editores y lenguajes de programación visual (VPL) basados en bloques. Utilizada en su version 1.20181219.0. 23, 33, 34, 35, 36, 42, 44, 45, 46, 47, 52, 53, 66, 69

Bloque Unidad mínima en MateFun. Posee una funcionalidad, parametros de entradas y salida. Es la que permite formar los programas. Para más detalle ver **Manual de Usuario**. 23, 24, 25, 28, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 49, 52, 53, 54, 61, 62, 67, 68, 69

Bloque bandera Bloque siempre presente en el área de trabajo. Indica cuáles son los bloques a tener en cuenta para realizar acciones. Para más detalle ver **Manual de Usuario**. 31, 41, 46

Bloque interrogante Permite indicar una entrada y su nombre para un nuevo bloque. Para más detalle ver **Manual de Usuario**. 38, 39, 40

Bootstrap Biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. 19

Botón ejecutar Es el que permite ejecutar el programa generado en el área de trabajo. Para más detalle ver **Manual de Usuario**. 49, 69

Botón guardar, matefun Botón que dispara la acción de guardado de un nuevo bloque. Para más detalle ver **Manual de Usuario** . 32

Bug Error o fallo en el sistema. 59

CSS Hojas de estilo en cascada, lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado 46

Cabal Sistema para crear y empaquetar bibliotecas y programas de Haskell. 47, 48

Caja negra En software, elemento que se estudia desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno. 36

Categoría Refiere a las secciones en que se divide el toolbox. Para más detalle ver **Manual de Usuario**. 11, 12, 13, 22, 24, 26, 32, 36, 44

Framework Estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software. 14, 20, 42, 46, 47, 56

Frontend Parte del sistema que se encarga de la interacción con el usuario. 50

HTML HyperText Markup Language, lenguaje de marcado para la elaboración de páginas web 6, 9

Haskell Lenguaje de programación estandarizado multi-propósito, funcionalmente puro, con evaluación no estricta y memorizada, y fuerte tipificación estática. 1, 7, 16, 47, 48, 49, 68

Imperativo, lenguaje de programación Paradiga en el que un programa consiste en una secuencia claramente definida de instrucciones para un ordenador. 5

Interfaz Conexión funcional entre dos sistemas. **Interfaz de usuario** es el medio que tiene el usuario para comunicarse con el sistema. 22, 28, 29,

30

Intérprete Es un programa capaz de analizar y ejecutar otros programas. En el contexto del informe, el intérprete de MateFun analiza y ejecuta código en este lenguaje. 15, 16, 21, 33, 34, 47, 48, 49, 55, 68

JNI, Java Native Interface *Framework* de programación que permite que un programa escrito en Java ejecutado en la máquina virtual java pueda interactuar con programas escritos en otros lenguajes como C, C++ y ensamblador. 49

JSON Formato de texto sencillo para el intercambio de datos. 21, 33, 48, 51, 52

JavaScript Lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo (*just-in-time*) con funciones de primera clase. Si bien es más conocido como un lenguaje de *scripting* (secuencias de comandos) para páginas web. 6, 19, 34, 42, 43, 51, 53

Layout Diseño de la vista. 33

Makefile, archivo Archivo que contiene intrucciones que debe realizar el comando *make*, utilizado para el lenguaje C. 48

Monolítica, arquitectura Refiere a aquella en el que el software se estructura de forma que todos los aspectos funcionales del mismo quedan acoplados y sujetos en un mismo programa. 33

NDK, Native Development Kit Kit de desarrollo que permite reutilizar código escrito en C/C++ introduciéndolo en las aplicaciones Android a través de JNI. 49

Nativa Refiere a que se ha desarrollado para un sistema operativo o dispositivo en particular 6, 12, 42, 49

Nativo Ver nativa. 49

Paleta de bloques Ver toolbox. 26, 31, 35, 36, 37, 44, 53

Plan Ceibal Plan de Conectividad Educativa de Informática Básica para el Aprendizaje en Linea. 30, 60

Plugin Aplicación que se relaciona con otra para agregarle una función nueva y generalmente más específica. 46, 47

Programa En MateFun es el conjunto de bloques, que comienza desde un bloque bandera, y cumple determinada funcionalidad. 28, 29, 30, 31, 32, 35

SPA, Single Web Aplication Aplicación de página única, es una aplicación

web o es un sitio web que cabe en una sola página con el propósito de dar una experiencia más fluida a los usuarios. 19

Sistema de archivos Sistema de ficheros, es el encargado de administras el uso de memoria en un Sistema Operativo. 34, 36, 53

Sombra, bloque Bloque especial que facilita el ingreso de valores en entradas. Para más detalle ver **Manual de Usuario**. 31, 32, 39, 40, 45

Tipo Atributo de los datos que indica al ordenador sobre la clase de datos que se va a manejar, pudiendo ser número, texto, entre otros. 29, 31, 40

Toolbox También denominado paleta de bloques. Sección donde se encuentran todos los bloques que pueden ser utilizados para crear un programa. Para más detalle ver **Manual de Usuario**. 44, 45

Web Services REST Tecnología que utiliza un conjunto de protocolos y estándares REST que sirven para intercambiar datos entre aplicaciones. 21

WebSocket Tecnología que proporciona un canal de comunicación bidireccional y *full-duplex* sobre un único *socket* TCP. 19, 21

WebView Componente del sistema con la tecnología de Chrome que permite a las aplicaciones de Android mostrar contenido web. 33, 42, 51

Workspace Espacio en el que se formará el programa de bloques. Para más detalle ver **Manual de Usuario**. 31, 35, 36, 41, 53

Workspace También denominado Área de Trabajo, espacio en el que se formará el programa de bloques. Para más detalle ver **Manual de Usuario**. 46

XML "Lenguaje de Marcas Extensible", es un metalenguaje que permite definir lenguajes de marcas desarrollado por el *World Wide Web Consortium* (W3C) utilizado para almacenar datos en forma legible. 44, 45, 46, 52, 53

"Shadow", bloque Bloque especial que facilita el ingreso de valores en entradas. Para más detalle ver **Manual de Usuario**. 47

ANEXOS

Resultados de la complejidad ciclomática en los métodos

Métodos	v(G)
com.example.matefun.BlocklyWebViewFragment.onCreateView	1
com.example.matefun.Enums.ErrorEnum.ErrorEnum	1
com.example.matefun.Enums.ErrorEnum.toString	1
com.example.matefun.Enums.MessagesEnum.MessagesEnum	1
com.example.matefun.Enums.MessagesEnum.toString	1
com.example.matefun.FigurasWebViewFragment.onCreateView	1
com.example.matefun.Helpers.GraphHelper.getDate	1
com.example.matefun.Helpers.GraphHelper.getJson2D	1
com.example.matefun.Helpers.GraphHelper.getJson3D	1
com.example.matefun.Helpers.GraphHelper.getJsonAnim2D	1
com.example.matefun.Helpers.GraphHelper.getJsonAnim3D	1
com.example.matefun.Helpers.GraphHelper.menuIconWithText	1
com.example.matefun.Helpers.JsDialogHelper.CancelListener.onCancel	1
com.example.matefun.Helpers.JsDialogHelper.CancelListener.onClick	1
com.example.matefun.Helpers.JsDialogHelper.JsDialogHelper	1

com.example.matefun.Helpers.JsDialogHelper.PositiveListener.PositiveListener	1
com.example.matefun.Helpers.JsDialogHelper.canShowAlertDialog	1
com.example.matefun.Helpers.SaveBlocksHelper.deleteFunction	1
com.example.matefun.MainActivity2.cleanResult	1
com.example.matefun.MainActivity2.convertDipToPercent	1
com.example.matefun.MainActivity2.deleteAllBlocks	1
com.example.matefun.MainActivity2.evalMateFunFromJNI	1
com.example.matefun.MainActivity2.hideWebViewFiguras	1
com.example.matefun.MainActivity2.initHS	1
com.example.matefun.MainActivity2.newIntent	1
com.example.matefun.MainActivity2.onCreateOptionsMenu	1
com.example.matefun.MainActivity2.selectIcon	1
com.example.matefun.MainActivity2.showWebViewFiguras	1
com.example.matefun.WebChromeClient.onJsAlert	1
com.example.matefun.WebChromeClient.onJsConfirm	1
com.example.matefun.WebChromeClient.onJsPrompt	1
com.example.matefun.Helpers.GraphHelper.isInteger	2
com.example.matefun.Helpers.SaveBlocksHelper.createDownloadFile	2
com.example.matefun.MainActivity.onCreate	2
com.example.matefun.MainActivity2.createNotificationChannel	2
com.example.matefun.MainActivity2.onCreate	2
com.example.matefun.MainActivity2.onFocusChange	2
com.example.matefun.MainActivity2.runOnWebView	2

com.example.matefun.Helpers.JsDialogHelper.JsDialogHelper	3
com.example.matefun.Helpers.SaveBlocksHelper.addBlockTooStorageToolbox	3
com.example.matefun.Helpers.SaveBlocksHelper.donIncludeBlock	3
com.example.matefun.MainActivity2.setWeights	3
com.example.matefun.Helpers.GraphHelper.ManegeImportFile	4
com.example.matefun.Helpers.JsDialogHelper.PositiveListener.onClick	4
com.example.matefun.Helpers.SaveBlocksHelper.createNecessaryFiles	4
com.example.matefun.Helpers.SaveBlocksHelper.generateNewBlock	4
com.example.matefun.Helpers.SaveBlocksHelper.writeBlockCode	4
com.example.matefun.MainActivity2.deleteBlock	4
com.example.matefun.MainActivity2.downloadBlocks	4
com.example.matefun.MainActivity2.onActivityResult	4
com.example.matefun.MainActivity2.run	4
com.example.matefun.Helpers.GraphHelper.evaluateResult	5
com.example.matefun.Helpers.GraphHelper.evaluateValidation	5
com.example.matefun.Helpers.JsDialogHelper.invokeCallback	5
com.example.matefun.Helpers.SaveBlocksHelper.deleteFile	5
com.example.matefun.Helpers.SaveBlocksHelper.deleteToolbox	5
com.example.matefun.Helpers.SaveBlocksHelper.getBlocksNames	5
com.example.matefun.MainActivity2.graphFigure	5
com.example.matefun.Helpers.JsDialogHelper.showOk	6
com.example.matefun.Helpers.SaveBlocksHelper.writeBlock	6
com.example.matefun.MainActivity2.saveBlock	6

com.example.matefun.MainActivity2.showResult	6
com.example.matefun.Helpers.SaveBlocksHelper.createToolboxFile	7
com.example.matefun.Helpers.SaveBlocksHelper.deleteFunctionBlockDef	7
com.example.matefun.MainActivity2.onOptionsItemSelected	7
com.example.matefun.Helpers.JsDialogHelper.showDialog	8
com.example.matefun.Helpers.SaveBlocksHelper.deleteFunctionBlock	9
com.example.matefun.Helpers.JsDialogHelper.showError	11
com.example.matefun.Helpers.SaveBlocksHelper.copyFile	11
com.example.matefun.Helpers.JsDialogHelper.showWarning	12
com.example.matefun.Helpers.GraphHelper.updateSelect	13
com.example.matefun.Helpers.SaveBlocksHelper.importBlocks	24
v(G) promedio	**3.62**

Resultados de LCOM en las clases

Clases	LCOM
com.example.matefun.Enums.ErrorEnum	0
com.example.matefun.Enums.MessagesEnum	0
com.example.matefun.BlocklyWebViewFragment	1
com.example.matefun.ExampleInstrumentedTest	1
com.example.matefun.ExampleUnitTest	1
com.example.matefun.FigurasWebViewFragment	1
com.example.matefun.Helpers.JsDialogHelper	1
com.example.matefun.Helpers.JsDialogHelper.CancelListener	1
com.example.matefun.Helpers.JsDialogHelper.PositiveListener	1
com.example.matefun.Helpers.SaveBlocksHelper	1
com.example.matefun.MainActivity	1
com.example.matefun.MainActivity2	3
com.example.matefun.WebChromeClient	3
com.example.matefun.Helpers.GraphHelper	11
LCOM promedio	**1.86**