UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

FACULTAD DE INGENIERIA

Interfaz de configuración de Robotito para usuarios no programadores

Santiago Hitta

Tutores: Gonzalo Tejera, Ewelina Bakala, Jorge Visca

Proyecto de grado de Ingeniería en Computación

Instituto de Computación, Facultad de Ingeniería
Universidad de la República

RESUMEN

Robotito es un robot educativo que fue diseñado con el objetivo de propiciar el desarrollo del pensamiento computacional en niños pequeños. El mismo es utilizado por maestras en centros educativos para realizar actividades que fomentan el desarrollo del pensamiento computacional y también es utilizado por investigadores para evaluar el impacto que tienen este tipo de actividades en niños pequeños. El robot cuenta con dos comportamientos predefinidos que permiten llevar a cabo algunas actividades, las cuales deben ser planificadas en base a esos comportamientos.

En este contexto, surge el presente proyecto en donde se aborda el diseño e implementación de un lenguaje de programación visual, el cual permite que usuarios sin conocimientos de programación puedan definir de forma sencilla mediante un dispositivo móvil nuevos comportamientos para Robotito. Esto permite realizar una mayor variedad de actividades con el robot.

Como resultado del análisis de diversos lenguajes de programación visuales, se optó por utilizar Blockly como base para el desarrollo del lenguaje visual. Blockly fue adaptado para generar un conjunto de bloques acorde a las posibilidades de Robotito y para a partir de estos bloques poder generar un código que se le envía al robot y que al ejecutar en el mismo determina su comportamiento.

Por último, a partir de las pruebas realizadas con usuarios, se concluye que el proyecto cumple con los objetivos planteados. Durante el proyecto fue desarrollada una aplicación Android que permite a usuarios no programadores definir de forma gráfica nuevos comportamientos para Robotito, a partir de los cuales se pueden realizar diversos escenarios con fines educativos. Como parte del trabajo a futuro se identificaron algunas mejoras a realizar que ayudarían a los usuarios a aprender más fácilmente el lenguaje.

Palabras clave: robotito, vpl, blockly, programación dirigida por eventos, android, bloques, robótica, educación

Índice de figuras

Índice general

Capítulo 1

Introducción

1.1. Motivación

Robotito [1] es un robot educativo que fue diseñado con el objetivo de propiciar el desarrollo del pensamiento computacional en niños pequeños. El mismo es utilizado por maestras en centros educativos para realizar actividades que fomentan el desarrollo del pensamiento computacional y también es utilizado por investigadores para evaluar el impacto que tienen este tipo de actividades en niños pequeños. El robot cuenta con una aplicación Android que permite configurarlo y seleccionar cuál comportamiento ejecutará, existiendo dos comportamientos definidos.
Hoy en día las actividades realizadas con Robotito están limitadas por estos dos comportamientos predefinidos. Es por esto que surge la necesidad de dar una mayor libertad a los usuarios (maestras e investigadores) permitiéndoles programar de forma sencilla e intuitiva nuevos comportamientos, los cuales les permitirán desarrollar nuevos tipos de actividades y escenarios de prueba. En este contexto surge el proyecto "Interfaz de configuración de Robotito para usuarios no programadores", que propone el desarrollo de un lenguaje de programación visual (VPL por su sigla en inglés) sencillo e intuitivo que permita a usuarios sin conocimientos de programación definir nuevos comportamientos para Robotito.

1.2. Objetivos

El objetivo del presente proyecto es implementar una aplicación móvil para Android que permita a usuarios no programadores definir de forma gráfica nuevos comportamientos para el robot, de forma sencilla e intuitiva. Para ello se realizará un proceso de diseño centrado en el usuario, con el fin de lograr un producto ajustado a las necesidades de los mismos. Esto permitirá ampliar las posibilidades de uso de Robotito tanto por parte de maestras como de investigadores.

1.3. Organización del documento

El presente documento se encuentra dividido en seis capítulos, a su vez, cada capítulo cuenta con 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, haciendo énfasis en lenguajes utilizados para la programación de robots para niños, sobre todo en el ámbito educativo.
En el Capítulo 3, se presenta el marco teórico, introduciendo herramientas y conceptos de gran relevancia para el desarrollo del proyecto.
El Capítulo 4 es el más extenso ya que contiene los detalles de desarrollo de la aplicación. En este capítulo se detalla el análisis realizado y se fundamentan decisiones de diseño y arquitectura de la aplicación, se describe la forma en que se implementaron las distintas funcionalidades, entre otros temas.
En el capítulo 5 se describen las pruebas de usabilidad que fueron llevadas a cabo para detectar posibles mejoras en el diseño de la aplicación así como de la interacción del usuario con la misma, el robot y el entorno.
Finalmente, en el Capítulo 6, se presentan las conclusiones y trabajo futuro.

El presente documento se complementa con los documentos anexos *Manual de Usuario* y *Estado del arte*.

Capítulo 2

Estado del arte

En este capítulo se expone un resumen del relevamiento y análisis de lenguajes de programación visual, que fue realizado en etapas iniciales del proyecto con el objetivo de obtener una perspectiva general de las distintas posibilidades tanto a nivel tecnológico como de diseño que podrían ser utilizadas para la implementación de un VPL para Robotito.

Un VPL o lenguaje de programación visual es cualquier lenguaje de programación que permite a los usuarios crear programas manipulando elementos gráficos en lugar de especificarlos textualmente [21]. Los VPLs 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 íconos, formas y colores, se facilita la comprensión de la semántica de los componentes del lenguaje.

El relevamiento realizado se centró en lenguajes utilizados para la programación de robots para niños, sobre todo en el ámbito educativo, aunque también se estudió otros lenguajes que podrían aportar nuevas ideas de diseño o que podrían ser llevados al ámbito de la robótica. Primero, se identificaron los distintos tipos o categorías de VPLs existentes y sus características, posteriormente, se estudiaron algunos lenguajes representativos de cada una de estas categorías. Para cada lenguaje, los principales puntos analizados fueron: su funcionamiento, su clasificación como VPL, el tipo de licencia mediante la cual se distribuye, las plataformas sobre las que funciona, los lenguajes de programación utilizados, el paradigma de programación utilizado, el público objetivo y la posibilidad de integrarlo o reutilizarlo con Robotito.

El relevamiento completo se puede consultar en el documento *Estado del arte* [33].

2.1. Revisión de VPLs estudiados

Los lenguajes de programación visual pueden clasificarse según su representación visual en tres categorías: lenguajes basados en íconos, lenguajes basados en diagramas o lenguajes basados en formularios [22].

En los lenguajes basados en íconos cada instrucción o conjunto de instrucciones es representada con un ícono descriptivo. Los íconos se van colocando en el espacio de trabajo en cierto orden y de esta forma se va creando un programa. Generalmente no es necesario que los usuarios sepan leer para programar con un lenguaje basado en íconos.

Los lenguajes basados en diagramas utilizan una representación estilo diagrama de flujo que muestra el flujo que va a seguir el programa, teniendo en algunos casos una representación del flujo de control del programa y en otros casos una representación del flujo de datos del programa. El flujo de control determina el órden en que se ejecutan las instrucciones, pudiendo por ejemplo establecer condiciones que determinarán si el programa ejecuta una u otra instrucción o si el programa repite cierto conjunto de instrucciones. El flujo de datos pone el énfasis en los datos y cómo estos fluyen a través del programa, en dónde y cómo se utilizan, cómo se transforman y cuáles son los datos que se obtienen al final del proceso así como los que se requieren al inicio.

Finalmente, los lenguajes basados en formulario le ofrecen al usuario la posibilidad de programar rellenando formularios. Hoy en día estos lenguajes se suelen implementar con bloques, ofreciéndole al usuario un conjunto de bloques predefinidos que contienen algún texto descriptivo y representan bloques de código. Estos bloques le permiten al usuario rellenar ciertos parámetros o seleccionar ciertas opciones como si se tratara de un formulario, de esa forma, mediante la conexión de los distintos bloques el usuario va generando el programa. Los lenguajes basados en formulario requieren que el usuario sepa leer para poder entender el formulario y las opciones que se le ofrecen.

A continuación se describen algunos lenguajes de programación visual representativos para cada una de estas categorías.

2.1.1. Thymio VPL

Thymio II es un robot educativo para niños que cuenta con una gran cantidad de sensores y actuadores [43].

Existen varias alternativas para programar el comportamiento de Thymio, entre ellas Scratch [28], Blockly [11], Aseba Studio (programación textual) [42] y Thymio VPL [44].

Thymio VPL es un lenguaje de programación visual basado en íconos que sigue un paradigma de programación dirigida por eventos y fue diseñado para niños de ensañanza primaria [34]. La programación dirigida por eventos, es un paradigma de programación en

el que tanto la estructura como la ejecución de los programas van determinados por los sucesos (eventos) que ocurran en el sistema [50]. La programación en Thymio VPL implica la creación de pares de eventos-acciones, esto se puede lograr arrastrando íconos hacia los pares de eventos-acciones vacíos que se encuentran en el área de trabajo, de esta forma los mismos se van rellenando con los distintos eventos y sus acciones correspondientes. La interfaz de programación se puede observar en la figura 2.1.

Figura 2.1: IDE de Thymio VPL
https://doi.org/10.3929/ethz-a-010144554

En Thymio VPL se cuenta con un ícono de estado que permite definir y utilizar distintos estados en la configuración. Los estados permiten añadir condiciones adicionales a los eventos, las cuales son determinadas a partir de los eventos que van sucediendo a medida que el robot interactúa con su entorno.

Thymio VPL puede ser accedido a través del entorno de desarrollo integrado Aseba Studio o como un módulo independiente, en ambos casos el código de la implementación es abierto y se distribuye bajo la licencia GNU Lesser General Public License v3.0. Thymio VPL está disponible para Windows, Mac y Linux, también existe un repositorio en github de una versión para Android la cual fue discontinuada. El código existente de esta versión Android es QML en su mayoría y algo de C++, las restantes versiones de este VPL están programadas principalmente en C++ teniendo también código QML y HTML.

Si se quisiera adaptar la implementación de Thymio VPL para otros robots, por ejemplo Robotito, se podría reutilizar parte de la interfaz gráfica (espacio de trabajo, manejo de los pares eventos-acciones, manejo de íconos) ya que la misma posee una licencia de código abierto, sin embargo, deberían reimplementarse casi todos los íconos puesto que los mismos están diseñados para Thymio y sus características. El resto de la implementación es específica para Thymio.

2.1.2. Scratch

Scratch [28] es un lenguaje de programación visual basado en formularios que fue desarrollado por el MIT Media Lab. Scratch fue diseñado como una herramienta introductoria a la programación para niños de entre ocho y dieciséis años y sigue un paradigma de programación dirigida por eventos. A pesar de que este lenguaje no está orientado a la programación de robots, el mismo ha sido adaptado para la programación de diversos robots como Thymio, GoPiGo [7], mBot [26], Sphero [35], etc.

El entorno de programación en Scratch consiste en un área de simulación en donde se muestra el comportamiento de los distintos elementos programados, luego se tiene una paleta de bloques con los distintos bloques de programación y, finalmente, un área de trabajo sobre la cual se van colocando y conectando los bloques (ver Figura 2.2). Estos bloques de programación tienen forma de piezas de puzzle y se pueden conectar unos con otros siempre y cuando la forma del bloque lo permita, esto evita que el usuario forme programas con una sintaxis incorrecta. Los bloques se agrupan por categorías. Los bloques de evento se utilizan para comenzar las actuaciones, luego se añaden bloques de acción de forma secuencial a partir del evento. Varios bloques ofrecen menúes o campos sobre los que se puede escribir, de aquí que el lenguaje sea un VPL de tipo formulario.

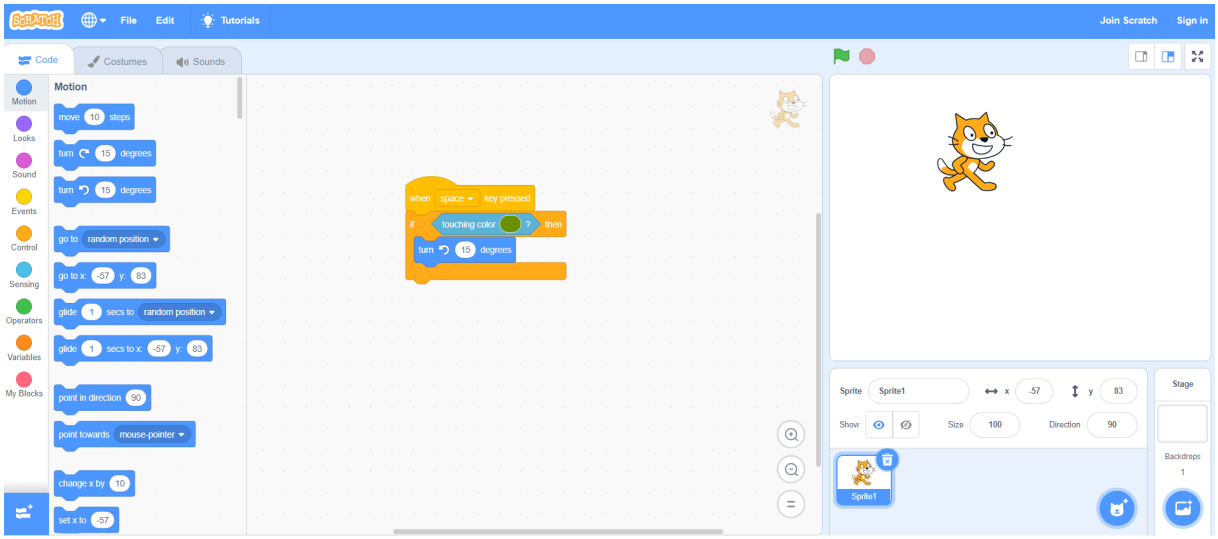


Figura 2.2: IDE de Scratch

La última versión de Scratch es la 3.0 y plantea un rediseño y reimplementación de Scratch basado en HTML5 y javascript. Existen versiones para Windows, macOS, ChromeOS y Android (solo tablets), utilizando ésta última Android Webview [13] que permite ejecutar HTML y javascript en aplicaciones Android. Scratch es de código abierto y se distribuye bajo la licencia BSD 3.
Existen posibilidades de crear extensiones para Scratch mediante ScratchX [15] que es una plataforma diseñada específicamente para esto. Estas extensiones se programan en lenguaje javascript y permiten incorporar nuevos bloques y adaptar este VPL al uso para la robótica. Además de ScratchX se debe utilizar algún software intermediario entre Scratch y el robot, por ejemplo Scratch Link [14] que añade soporte para algunos periféricos como Lego WeDo 2.0 o el kit Lego MindStorms EV3 [24]. El modo de funcionamiento es con el robot conectado al ordenador, siendo el ordenador quien ejecuta las instrucciones y envía comandos al robot.

2.1.3. Scratch Jr

Scratch Jr. [16] es un VPL icónico con algunas similitudes a Scratch aunque más simplificado ya que está orientado a niños más pequeños, de entre cinco y siete años, su interfaz se puede observar en la figura 2.3.

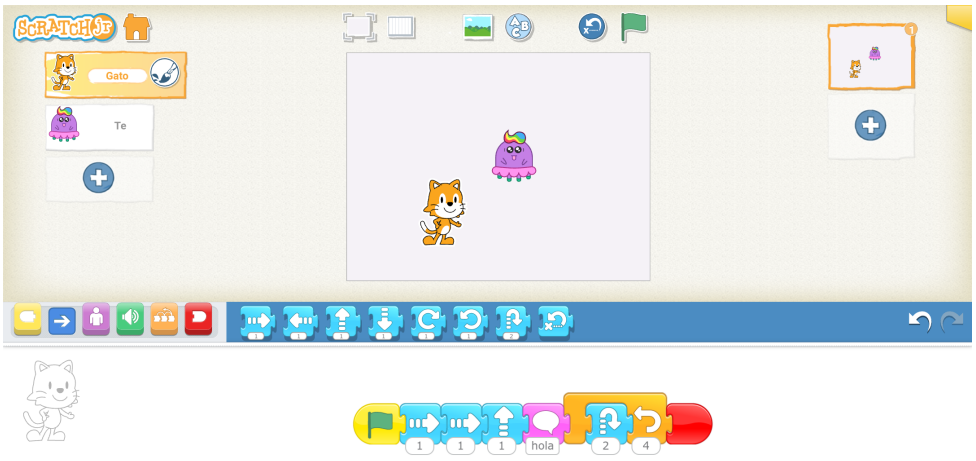


Figura 2.3: IDE de Scratch Jr.

En Scratch Jr se reduce la cantidad y complejidad de los bloques de programación disponibles, los mismos se convierten en íconos sin texto en vez de bloques estilo formulario siendo éste un VPL de tipo icónico.
Scratch Jr se basa en un paradigma de programación dirigida por eventos y ofrece algunos íconos de evento como: presionar bandera verde, pulsar sobre el personaje, colisionar con

el personaje o recibir mensaje.

Scratch Jr es de código abierto, se distribuye bajo la licencia BSD 3 y está desarrollado en javascript, HTML, Java y Objective-C, contando con una versión Android y otra iOS. Lamentablemente, Scratch Jr a diferencia de Scratch no cuenta con posibilidades de agregar extensiones y por lo tanto sería difícil de integrar con Robotito.

2.1.4. Blockly

Blockly [11] es una librería javascript desarrollada por Google que facilita el proceso de creación de un VPL. Blockly permite definir lenguajes que utilizan bloques, ya sea un lenguaje basado en formularios o un lenguaje icónico y tiene un aspecto muy similar a Scratch, contando con una paleta de bloques y un área de trabajo (ver Figura 2.4). Una vez que el usuario genera un programa mediante la unión de los bloques, Blockly permite exportar dicho programa a código en varios lenguajes de programación: javascript, Dart, Lua, Python o PHP.

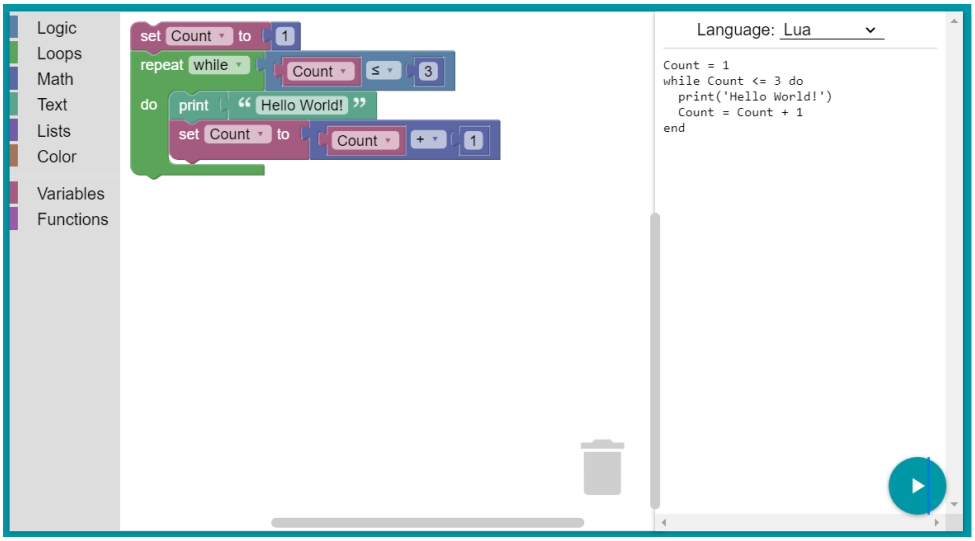


Figura 2.4: IDE estándar de Blockly

Por defecto Blockly cuenta con las siguientes categorías de bloques: lógicos, iterativos, matemáticos, de texto, de listas, de color, de variables y de funciones. Además de ser fácilmente integrable a aplicaciones web y móviles, otra de las fortalezas de Blockly es que es extensible, permitiendo generar nuevos bloques de acuerdo a las necesidades del VPL que se quiera crear. También podemos especificar a qué categoría pertenece el bloque, qué tipo de bloque es, con qué otros bloques puede unirse, qué código generará y en qué lenguaje. La forma de definir estos nuevos bloques es mediante Javascript o JSON o mediante el módulo Blockly Developer Tools que es una aplicación basada en Blockly

para ayudar al desarrollador a crear mediante el uso del VPL nuevos bloques, cajas de herramientas y configuraciones. Si bien en las categorías por defecto no tenemos bloques de eventos, los mismos se pueden crear, por lo tanto Blockly permite crear un VPL que siga un paradigma de programación secuencial o un paradigma de programación dirigida por eventos.

Esta librería está hecha en javascript y HTML con lo cual se puede utilizar en navegadores web o embeber en aplicaciones móviles mediante un WebView. Blockly es de código abierto y se distribuye bajo la licencia Apache License 2.0. Existen varios proyectos de robótica que han utilizado esta librería para ofrecerle al usuario un VPL para la programación del robot, entre ellos Thymio, Yatay [29], Dash [27], Otto [51] y Cubelets [31]. Blockly se puede integrar con Robotito ya que permitiría crear un lenguaje basado en bloques con el cual se podría definir el comportamiento del robot.

2.1.5. Blue-Bot VPL

Blue-Bot [38] es uno de los tantos robots de tipo tortuga desarrollados por la empresa estadounidense Terrapin, desarrolladora del Bee-Bot [37], Blue-Bot, Tuff-Bot [41], Pro-Bot [40] y InO-Bot [39]. Los robots de tipo tortuga [2] son una clase de robots educativos que ofrecen una serie de comandos de movimiento y dirección, los cuales típicamente permiten resolver problemas de trazar una ruta en un mapa o generar figuras geométricas.
Blue-Bot es un robot muy sencillo, para niños de entre cuatro y diez años, que se puede programar de forma gráfica mediante un VPL icónico, el cual se muestra en la figura 2.5.

Figura 2.5: IDE de Blue-Bot VPL

La programación mediante el VPL sigue un paradigma se programación secuencial y se logra arrastrando alguno de los íconos de desplazamiento, rotación o pausa hacia la zona de trabajo ubicada sobre la izquierda, la cual permite la colocación de varios íconos verticalmente comenzando desde arriba. La programación mediante esta interfaz gráfica cuenta con la ventaja de que se puede visualizar en tiempo real los comandos que va ejecutando el robot, siendo los mismos resaltados en la interfaz, además se cuenta con una representación virtual del robot trabajando sobre distintos escenarios.

El VPL para Blue-Bot está disponible para dispositivos iOS, Android, Windows y Mac. Este software se distribuye bajo licencia propietaria y es de código cerrado.

2.1.6. Dash VPL

Dash [27] es un robot educativo diseñado por la empresa norteamericana Wonder Workshop.

Los usuarios tienen varias alternativas para programar el comportamiento de Dash, en particular nos interesa la Wonder app (ver Figura 2.6) que nos ofrece un VPL de tipo diagramático (diagrama de flujo de control), mediante el cual podemos generar máquinas de estado que determinan el comportamiento del robot, siguiendo un paradigma de programación dirigida por eventos.

Figura 2.6: IDE de Dash Wonder App
`https://www.makewonder.com/apps/wonder/`

Este VPL le permite al usuario arrastrar íconos que representan acciones hacia el

espacio de trabajo, luego el usuario puede asociar los íconos estableciendo las trancisiones desde una acción a otra y de esta forma va creando la máquina de estados que define el comportamiento del robot, en donde cada estado representa una actuación del robot y cada trancisión representa algun evento que hace que el robot ejecute otro estado.

Los íconos de estado o actuación se agrupan por categorías, cada ícono dentro de estas categorías representa una actuación del robot y la mayoría de ellos permiten configurar algunas de las características de la actuación mediante una ventana emergente. Muchas de las acciones realizadas por Dash son complejas, por ejemplo la actuación de dormir o despertar implica la utilización sincronizada de varios de los actuadores del robot como por ejemplo los leds, parlantes y ruedas. Otra característica de las actuaciones es que una gran parte de ellas son acompañadas por estados de ánimo, sobre todo las pertenecientes a la categoría de animaciones, en donde se pueden realizar acciones como por ejemplo avanzar hacia adelante en modo feliz, triste o aburrido.
Cuando el robot ejecuta un comportamiento dado podemos obtener retroalimentación visual observando la máquina de estados, en ella los distintos estados por los que va pasando son resaltados, al igual que las trancisiones que se disparan.

La Wonder App, que nos ofrece este VPL, está disponible para dispositivos iOS, Android, ChromeOS y Windows. La app se distribuye mediante una licencia de software propietaria y es de código cerrado. No se pudo encontrar documentación en línea detallando los lenguajes de programación utilizados, con lo cual no existen posibilidades de integrar este producto con Robotito.

2.1.7. Matefun Infantil

Matefun Infantil [23] es un VPL de tipo diagramático (diagrama de flujo de datos), que sigue un paradigma de programación funcional y fue desarrollado para facilitar la introducción de alumnos de nivel escolar a la programación funcional. Este lenguaje, cuenta con tres paneles de forma similar a Scratch o Scratch Jr, uno donde se almacenan por categorías los bloques que se le ofrecen al usuario para programar, otro panel para organizar los bloques y crear el programa y finalmente un panel para observar el resultado (ver Figura 2.7).

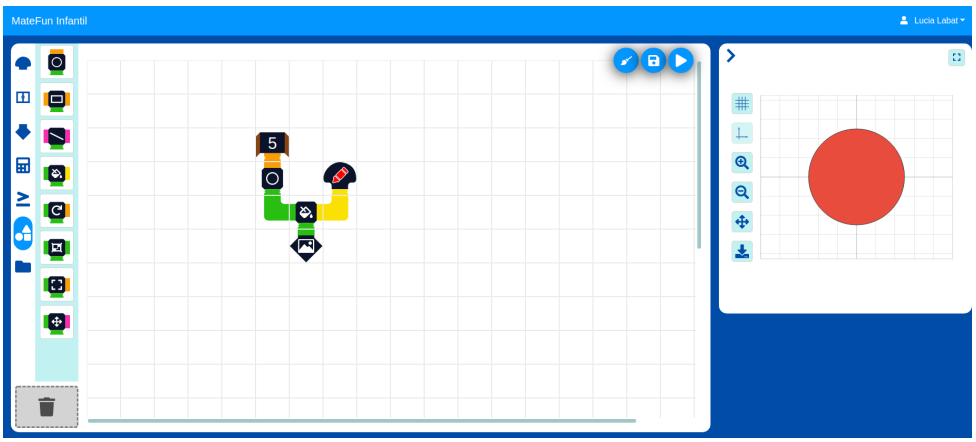


Figura 2.7: IDE de Matefun Infantil
`https://gitlab.fing.edu.uy/matefun/infantil/-/wikis/home`

La programación mediante este VPL consiste en la creación de un sistema de cañerías, las cuales dirigen el flujo de datos del programa a través de distintas funciones de transformación, lo cual resulta en alguna salida de datos. El sistema de cañerías representa una metáfora para indicar que se está dirigiendo el flujo de datos a través del sistema, en donde las uniones de cañerías representan funciones de transformación y los caños representan datos que fluyen a través de las cañerías, los cuales se deberán colocar al inicio de cualquier caño que se desee utilizar. Las funciones pueden recibir hasta tres entradas (por arriba, por la izquierda y por la derecha), y generan una única salida (por debajo), es decir que los datos fluyen desde arriba hacia abajo a través del sistema. Los colores de las cañerías representan el tipo de datos que transportan, teniendo cada una de las funciones una cierta cantidad de caños de entrada (máximo tres), cada uno con un determinado color (es decir, un tipo de datos), y un caño de salida que también tendrá algún color según su tipo de datos.
Matefun Infantil es de código abierto, no se logró encontrar información sobre su licencia. Este VPL plantea un diagrama de flujo de datos, lo cual no parece muy adecuado para programar los comportamientos de Robotito, debido a esto consideraremos que este VPL no es integrable con Robotito.

2.2. Conclusiones

En el Cuadro 2.1 se presenta una comparación de los VPLs relevados según los aspectos definidos en la sección Objetivos.

VPL	Tipo	Licencia	Plataformas	Paradigma	Lenguajes	Integrable	Publico objetivo
Thymio	Icónico	LGPLv3	Windows, Mac, Linux	Eventos	C++/QML	NO	Enseñanza primaria
Scratch	Formulario	BSD 3	Windows, Mac, Android, ChromeOS, Web	Eventos	HTML, js	SI	8 a 16 años
Scratch Jr	Icónico	BSD 3	iOS, Android	Eventos	HTML,js, Java, Obj-C	NO	5 a 7 años
Blockly	Formulario	Apache 2.0	Web	Multi-paradigma	HTML, js	SI	Indefinido
BlueBot	Icónico	Propietario	iOS,Mac, Windows Android,	Secuencial	Sin información	NO	4 a 10 años
Dash	Diagramático	Propietario	Windows, iOS, Android, ChromeOS	Eventos	Sin información	NO	Mayores de 6 años
Matefun Infantil	Diagramático	Sin información	Web	Funcional	js	NO	Enseñanza primaria

Cuadro 2.1: Comparación de VPLs

En cuanto a las alternativas de diseño para Robotito VPL se concluyó que cualquiera de los tres tipos de VPLs identificados serían aptos para los fines de este proyecto, sin embargo los recursos para el desarrollo de este proyecto son limitados.

Optar por la realización de un VPL diagramático sería muy costoso ya que no se podría integrar ninguna de las herramientas y lenguajes estudiados ni tampoco reutilizar parte del código, es decir que se tendría que crear un lenguaje desde cero y además desarrollar una interfaz gráfica compleja (con máquinas de estados o algún otro tipo de diagrama).
Por otra parte, si se realiza un lenguaje icónico o basado en formularios, se podría utilizar Blockly ya que resuelve gran parte del problema permitiendo invertir los recursos en el diseño del lenguaje (los bloques, las categorías y cómo interactúan entre sí), además es open source, de licencia libre y tiene posibilidades de integración con el robot.
Otra alternativa que se consideró en etapas iniciales del proyecto fue la de realizar un lenguaje icónico similar a Thymio VPL, en donde se definen pares de eventos y acciones, este lenguaje parece simple e intuitivo, además de que la interfaz gráfica no parece muy compleja, sin embargo habría que desarrollarlo desde cero o reutilizar y adaptar algún componente.

Luego de evaluar las distintas alternativas se optó por utilizar Blockly para desarrollar un VPL basado en formularios o icónico, esto se debió a varios factores:

- **Trayectoria:** Hay una gran cantidad de proyectos basados en Blockly, entre ellos hay varios aplicados a la robótica como Yatay, Ozoblockly, Thymio, Dash, Otto, Cubelets, Revolution Robotics, Robot Magic.
- **Familiarización de los usuarios:** Los lenguajes basados en bloques se están volviendo cada vez más populares para la enseñanza de la programación. Los mismos se están introduciendo en las aulas de enseñanza primaria y secundaria en Uruguay [3], con lo cual se espera que algunos de los usuarios finales ya estén familiarizados con la programación con bloques.
- **Comunidad:** Existe una gran comunidad online que está activa y ofrece soporte a los desarrolladores. Además hay gran cantidad de documentación y ejemplos.
- **Integración:** Blockly es open source y tiene una licencia libre, además se puede embeber en un WebView, por lo tanto se puede incorporar al proyecto sin problemas.
- **Idoneidad:** Blockly, a diferencia del resto de proyectos estudiados, es una herramienta que fue diseñada para la creación de nuevos lenguajes de programación visual. Además Blockly aporta buenas soluciones a problemas de usabilidad y diseño y permite al desarrollador centrar sus esfuerzos en el diseño de los íconos y categorías del lenguaje.

Capítulo 3

Marco teórico

En los capítulos 1 y 2 se realizó una breve introducción sobre Robotito y Blockly. Para el presente proyecto se optó por adaptar las funcionalidades de Blockly para poder crear un lenguaje de programación que permita definir nuevos comportamientos para Robotito.
En este capítulo se presentan en mayor detalle las características relevantes de Robotito y de Blockly. También se introduce el concepto de diseño centrado en el usuario, el cual será de especial importancia ya que es el proceso de diseño elegido para la realización de este proyecto.

3.1. Robotito

3.1.1. Hardware

Robotito [36] cuenta con un conjunto de sensores y actuadores que le permiten interactuar con el entorno de distintas formas.

El robot tiene tres ruedas omnidireccionales situadas cada una con una separación de 120 grados, esto le permite moverse en todas las direcciones sobre un plano y también rotar sobre sí mismo, pudiendo realizar dichos movimientos a distintas velocidades. También cuenta con un anillo de leds con 24 leds RGB, situado sobre la parte superior de su carcasa en forma circular. Por otra parte incorpora un transductor piezoeléctrico (o buzzer), que le permite convertir señales eléctricas en distintos sonidos.

Para obtener información del entorno el robot dispone de dos tipos de sensores. Por un lado están los seis sensores laser, que son sensores de distancia y permiten que el robot detecte objetos del entorno y a qué distancia aproximada están, los mismos se encuentran distribuidos en el lateral del robot, próximos al suelo pero apuntando horizontalmente

hacia el exterior. Por otro lado existe otro sensor que permite detectar distintos colores así como proximidad, este sensor está debajo del robot en su centro geométrico y apuntando hacia abajo, esto le permite detectar tarjetas de distintos colores sobre el suelo así como detectar si se encuentra parado sobre el mismo. En la figura 3.1 se puede apreciar el robot, en la misma se puede observar tanto el anillo de 24 leds que se encuentra en la parte superior, así como tres de los seis sensores de distancia, que se resaltan con un recuadro rojo. También, dentro del recuadro de color amarillo se observa el interruptor para prender y apagar el robot.

Figura 3.1: Robotito

Finalmente, el microcontrolador del robot es una placa Sparkfun ESP32 Thing, la cual provee la CPU, memoria RAM y memoria no volátil que se necesita para ejecutar aplicaciones, así como los conectores de entrada y salida que permiten la incorportación de los sensores y actuadores mencionados. Además cuenta con una interfaz WiFi que permite la conexión remota al robot.

3.1.2. Software

El sistema operativo instalado en el robot es Lua RTOS [49], un sistema operativo en tiempo real diseñado para ejecutar en sistemas embebidos, con requerimientos mínimos de memoria (tanto memoria RAM como no volátil). El sistema operativo es open source y fue desarrollado por un equipo interdisciplinario de ingenieros, educadores y diseñadores, conocido como Whitecat [48].
Lua RTOS cuenta con un interprete del lenguaje Lua 5.3.4, el cual ofrece al programador todos los recursos del lenguaje de programación Lua, además de módulos especiales para entre otras cosas poder acceder al hardware del robot y utilizar hilos. Lua [25] es un lenguaje de programación de alto nivel, multiparadigma, estructurado, imperativo y bastante ligero, el cual funciona como un lenguaje interpretado, esto último nos permite enviar y ejecutar scripts en el robot de forma sencilla sin necesidad de compilar el código enviado.

Los scripts enviados al robot interactúan con el mismo a través de una librería desarrollada con este propósito (disponible en [46]). En esta librería contamos con módulos para controlar sensores y actuadores del robot, un módulo para manejo de redes, un sistema de memoria flash (no volátil) que sirve para configurar el robot y módulos que permiten definir máquinas de estado jerárquicas para definir el comportamiento del robot.

3.1.3. Aplicación de configuración

Existe una aplicación Android de configuración del robot que fue utilizada por maestras e investigadores para realizar actividades educativas con niños en distintos centros educativos.

Entre otras cosas, esta aplicación permite a los usuarios elegir un comportamiento predefinido para robotito, pudiendo elegir un comportamiento basado en el sensor de color o un comportamiento basado en los sensores de distancia.
En el comportamiento basado en el sensor de color, cuando el robot pasa por encima de una tarjeta de color cambia su dirección de movimiento.
En el comportamiento basado en los sensores de distancia, el robot sensa los objetos en su vecindad y luego avanza hacia el objeto más lejano. Una vez que llega al objetivo, el robot vuelve a sensar sus alrededores para encontrar un nuevo objetivo (el objeto más lejano). Si en algún momento, cuando el robot avanza hacia un objetivo, se pierde de vista dicho objetivo entonces el robot vuelve a sensar el entorno en busca de un nuevo objetivo.

Otra funcionalidad importante de esta aplicación es la calibración del sensor de color, la cual mejora la precisión de dicho sensor y previene problemas de funcionamiento al utilizar

comportamientos basados en color. Pueden haber cambios de iluminación del ambiente que afecten el sensado de colores, así como también el uso de tarjetas de color con distintas tonalidades, es por esta razón que es importante calibrar este sensor antes de utilizar el robot.

3.2. Blockly

A continuación se detallan algunos de los aspectos de Blockly que fueron más relevantes para este proyecto.

Blockly permite el guardado del espacio de trabajo y los bloques en formato XML. Es posible guardar y cargar programas manteniendo las posiciones y conexiones de los bloques, así como los valores de sus campos.

Otra característica de Blockly ofrece la posibilidad de capturar distintos eventos que ocurren en el área de trabajo, ofreciendo gran variedad de eventos como por ejemplo *BLOCK_CREATE*, *BLOCK_DRAG* y *BLOCK_MOVE*, que permiten detectar el momento exacto en que se crea, arrastra o reposiciona un bloque en el área de trabajo. Esta característica es importante ya que permite asociar a los eventos funciones que se ejecutarán cada vez que sucedan dichos eventos y esto habilita un mayor nivel de personalización de la aplicación y el lenguaje.

La definición de bloques en Blockly se realiza especificando principalmente su color, sus conectores y los campos que posee. A partir de los conectores, los campos y el color, Blockly determina la forma que tendrá el bloque.
Los conectores de un bloque pueden ser de tipo *statement*, *input* o *output*. Un bloque puede tener hasta dos conectores de tipo *statement*, el *PreviousStatement* permite conectar bloques encima y el *NextStatement* permite conectar bloques debajo del bloque. Los conectores de tipo *input* pueden ser *ValueInput* o *StatementInput*. Los *ValueInput* permiten conectar bloques o bien externamente hacia la derecha o bien internamente (es decir adentro del bloque). Los *StatementInput* permiten conectar bloques adentro del bloque pero en vertical. Finalmente, un bloque puede tener solamente un conector de tipo *output*, que permite conectar con otros bloques hacia la izquierda. En la figura 3.2 se pueden observar los distintos tipos de conectores existentes.

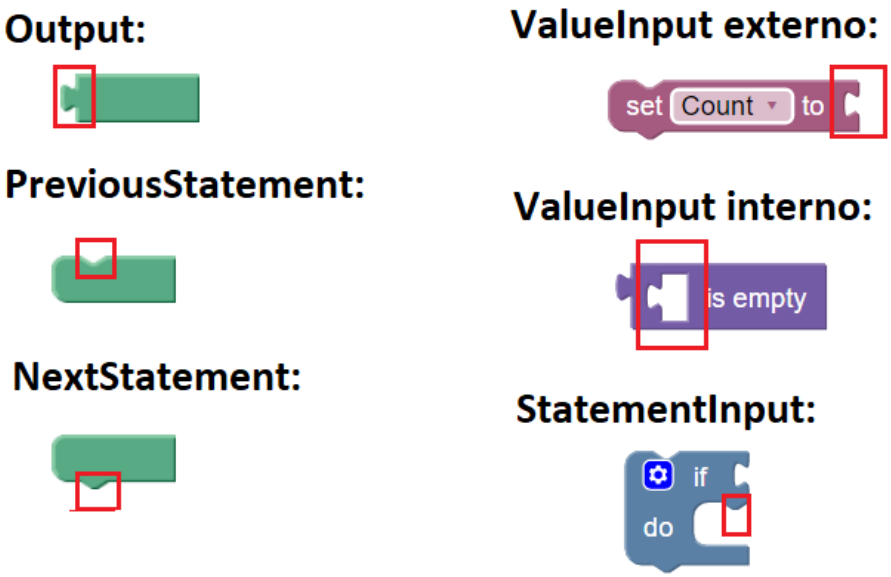


Figura 3.2: Conectores en Blockly

Los campos definen los elementos visuales dentro de un bloque, incluyen etiquetas en formato String, imágenes, entradas para datos literales, listados desplegables y editores enriquecidos como selectores de fecha o ángulos. A modo de ejemplo, en la figura 3.2 se puede apreciar que el bloque con *ValueInput* externo tiene campos de etiqueta en formato String y también un campo de listado desplegable. Blockly permite la creación de nuevos tipos de campos, lo cual resulta útil en caso de que los campos que vienen por defecto sean insuficientes para el lenguaje que se desea crear. Una de las características más interesantes sobre la definición de nuevos campos es la posibilidad de definir un editor para el campo, que puede ser de tipo *DropDownDiv* o *WidgetDiv*, ambos editores flotan encima de la interfaz de usuario de Blockly. El editor se despliega al hacer click sobre el campo, en caso del *DropDownDiv* se despliega un pequeño editor dentro de una caja conectada al campo, en el caso del *WidgetDiv* se debe definir el posicionamiento y la forma del editor manualmente. En ambos casos se debe definir el HTML a mostrar dentro del editor, pudiendo opcionalmente asociar eventos javascript a dichos elementos HTML para que el usuario pueda interactuar con los mismos impactando sobre el valor del campo.

A partir de las estructuras de bloques creadas por el usuario y de los valores de los campos de dichos bloques, Blockly permite generar código en distintos lenguajes, así como definir nuevos generadores de código. Si se opta por definir un nuevo generador de código, se debe determinar qué código retorna cada uno de los bloques del lenguaje.
Para generar código, Blockly recorre el área de trabajo desde arriba hacia abajo en busca de bloques. En el área de trabajo pueden existir varias estructuras conexas de bloques, para cada una de ellas Blockly se queda con el bloque superior de dicha estructura, es

decir aquel que esté más arriba y más a la izquierda. Luego, a partir de dichos bloques superiores se genera el código para cada una de las estructuras de bloques. El código final es de tipo String y concatena los códigos de las distintas estructuras de bloques, quedando primero el código de aquellas que estén más arriba en el área de trabajo. Esto es muy relevante dado que determina el órden en el que se ejecutará el código a posteriori.

Finalmente, una de las características más importantes de Blockly es que es de código abierto, pudiendo adaptar dicho código en caso de ser necesario para que se adecue a las características del lenguaje que se desea crear.

3.3. Diseño centrado en el usuario

En este proyecto es de gran utilidad realizar un proceso de diseño centrado en el usuario, debido a las grandes diferencias existentes entre el modelo mental de un programador y una persona sin conocimientos de programación, sobre todo cuando el producto a diseñar es un lenguaje de programación. Lo que podría parecer sencillo o intuitivo para un programador no tiene porque serlo para una persona sin conocimientos de programación. Por otra parte, también existe una gran brecha entre lo que un programador pueda saber sobre la educación de niños pequeños y los conocimientos de una maestra en esta área.

El diseño centrado en el usuario es un proceso iterativo de diseño que se centra en los usuarios y sus necesidades en cada fase del proceso, involucrando a los usuarios mediante distintas técnicas de investigación y de diseño con el objetivo de crear un producto con una gran usabilidad y accesibilidad para ellos [19]. Según Donald Norman: "Diseño centrado en el humano significa comenzar con un buen conocimiento de las personas y las necesidades que el diseño tiene que satisfacer" [30].

El diseño centrado en el usuario se basa en unos pocos principios fundamentales que se deben aplicar:

- Comenzar conociendo a los usuarios, identificar cuáles son sus necesidades.
- Involucrar a los usuarios en el proceso de diseño.
- Además de considerar al producto y el usuario, se debe tener en cuenta el contexto en el cual el producto se utilizará, cómo ese contexto afecta la experiencia del usuario con el producto y qué implicancias tiene esto para el diseño.
- Realizar un proceso iterativo de diseño, prototipado y testing en donde el producto se va evaluando con los usuarios y mejorando de acuerdo a la retroalimentación obtenida.

A nivel general las fases de un proceso de diseño centrado en el usuario son [45]:

1. Especificar el contexto de uso: Identificar a las personas que van a utilizar el producto, para qué lo van a usar y bajo qué condiciones.

2. Especificar requerimientos: Identificar los requerimientos de negocio y objetivos del usuario que deben ser alcanzados.

3. Diseñar soluciones: Se debe hacer en etapas, prototipando hasta llegar a un diseño completo.

4. Evaluar los diseños: Evaluación, idealmente mediante pruebas de usabilidad con los usuarios finales.

En general, al realizar un proceso de diseño centrado en el usuario, aumentan las probabilidades de lograr un producto que satisfaga las necesidades de los usuarios finales.

Capítulo 4

Desarrollo de la aplicación

4.1. Análisis

En esta sección se describen los requerimientos, casos de uso y las entrevistas con usuarios que fueron realizadas en etapas iniciales del proyecto. Todo esto permitirá tener una clara idea de cuáles son las necesidades de los usuarios finales y qué es lo que se desea del producto.

4.1.1. Requerimientos

A continuación se describen los requerimientos funcionales y no funcionales del proyecto. Los mismos están escritos de forma genérica, sin embargo, puede resultar útil aclarar que el concepto **comportamiento** corresponde a un programa creado con bloques en la aplicación desarrollada. En el contexto de la programación dirigida por eventos, un **comportamiento** puede estar conformado por varios eventos distintos y sus correspondientes acciones asociadas.

4.1.1.1. Requerimientos funcionales

- **Alta de comportamiento:** El usuario podrá definir un nuevo comportamiento para el robot, ésto implica definir cómo el robot reaccionará al entorno.
- **Guardar comportamiento:** El usuario podrá guardar un comportamiento asignándole un nombre o identificador.
- **Visualizar comportamientos:** La aplicación permitirá visualizar en un listado los comportamientos previamente definidos.
- **Editar comportamiento:** El usuario podrá acceder a un comportamiento previamente definido y editarlo.

- **Eliminar comportamiento:** La aplicación permitirá al usuario eliminar un comportamiento previamente definido.
- **Ejecutar comportamiento:** La aplicación permitirá al usuario ejecutar en el robot un comportamiento previamente definido, el robot deberá actuar de acuerdo al comportamiento definido por el usuario. Además se desea que dicho comportamiento quede guardado en el robot y que pueda ejecutar de forma automática (sin utilizar la aplicación) en futuros usos del mismo.
- **Retroalimentación de la actuación del robot:** El usuario podrá obtener información sobre lo que el robot está haciendo en un momento dado durante la ejecución de un comportamiento.

4.1.1.2. Requerimientos no funcionales

- La aplicación debe funcionar en dispositivos Android.
- La aplicación debe contar con un manual de usuario.
- El sistema deberá proveer una interfaz intuitiva y amigable al usuario, que sea fácil de aprender y de utilizar.
- La implementación del sistema debe ser clara y consistente, facilitando el modificar o añadir funcionalidades a futuro.

Si bien no se planteó como un requerimiento del proyecto, durante el desarrollo del mismo siempre se tuvo en cuenta la posibilidad de que la aplicación fuese utilizada de forma experimental para evaluar el uso de la misma por niños pequeños que aún no sepan leer, con lo cual se tuvo preferencia por el uso de elementos gráficos en lugar de elementos textuales siempre que fuera posible.

4.1.2. Entrevistas con usuarios

En las etapas iniciales del proyecto se realizaron entrevistas con usuarios finales, para ello se contactó a los usuarios que habían hecho uso de Robotito, logrando la colaboración de dos maestras y dos investigadores.

4.1.2.1. Objetivos

El objetivo de las entrevistas fue lograr una aproximación a los usuarios finales, conocer sus experiencias con Robotito y la forma en la que lo habían utilizado hasta ahora (en qué contexto, con qué objetivos, cómo resultó la experiencia), aprender sobre sus expectativas y necesidades de cara al futuro, sobre sus conocimientos de programación y robótica,

fundamentalmente lograr obtener escenarios puntuales de uso del robot bajo la hipótesis de que el mismo podrá ejecutar nuevos comportamientos definidos por el usuario.

4.1.2.2. Procedimiento

Las entrevistas se realizaron vía Zoom y consistieron en una serie de preguntas orientadas a cumplir los objetivos detallados anteriormente, la duración de cada una fue de una hora aproximadamente. Las mismas fueron grabadas para poder reproducirlas posteriormente recabando la información más relevante.

4.1.2.3. Resultados

Se logró en mayor medida cumplir con los objetivos definidos en la sección 4.1.2.1, sin embargo hubo dificultad a la hora de lograr que los usuarios definan escenarios concretos y detallados para trabajar con el robot a futuro, con lo cual en muchos casos se obtuvo una descripción genérica del escenario y sus objetivos.

A continuación se detallan los escenarios de uso del robot que se logró recabar:

- **Puntos cardinales:** Trabajo sobre un mapa de Uruguay en donde el robot se mueve y al llegar a cada punto cardinal prende luces de un color determinado.
- **Emociones:** Se utilizan dibujos de caras de distintos colores que representan emociones, cuando el robot pasa encima de las caras reacciona según la emoción.
- **Comportamiento variable:** No se especificó un escenario concreto pero sí el deseo de poder generar distintos comportamientos ante un mismo evento, dependiendo de los eventos anteriores.
- **Identificar elementos sobre mapa:** El robot reacciona ante los distintos elementos dibujados en un mapa de Uruguay, como por ejemplo ríos, fauna, flora.
- **Programación secuencial:** Poder predefinir una secuencia de movimientos para que el robot resuelva un laberinto o haga un recorrido determinado.

Aparte de los escenarios, los cuales son fundamentales para tener una idea de las necesidades de configuración del robot así como los objetivos de los usuarios, se relevaron otros aspectos que también son importantes. Algunos de estos aspectos se detallan a continuación:

- El comportamiento de color fue el más utilizado y valioso para los usuarios, por lo tanto es deseable que dicho comportamiento pueda ser definido mediante la aplicación.

- Las emociones más relevantes para trabajar con los niños son: alegría, tristeza, enojo, miedo.
- Sería útil tener alguna forma de identificar cuál es el frente del robot, ya que esto ha generado confusión en algunos usuarios.
- Para configurar las luces sería ideal o bien configurar todas de un determinado color o bien configurarlas por arcos.
- Sería útil tener opciones predeterminadas de velocidad de movimiento como: lento, medio o rápido.
- No hay una preferencia en cuanto a si las opciones de movimiento deberían ser continuas o acotadas (por ejemplo: por tiempo, distancia o pasos).

Además de los insumos aportados por las entrevistas, se consiguió una planilla excel en donde se recopilan datos sobre las experiencias de las maestras que han utilizado el robot para realizar actividades con niños en distintos centros educativos. En la planilla se registran detalles sobre dichas actividades, en qué consistieron, qué dificultades surgieron, sugerencias, edad de los niños, formato de la actividad, duración, etc.
Una de las dificultades que reportaron las maestras a la hora de utilizar el Robotito fue que habían errores con la detección de los colores. Estos errores podrían indicar una omisión por parte de las maestras a la hora de calibrar el sensor de color previo a la realización de una actividad. Esto deberá ser tenido en cuenta en el proceso de diseño de la aplicación.
Por otra parte a partir de esta planilla se logró obtener una mejor idea del contexto de uso del robot y del tipo de actividades que se realizan. Algo que se dio en varias actividades fue que se hizo una personificación del robot, creando una narrativa en donde al mismo se le dan atributos de una persona o un ser vivo, esto refuerza la idea de que puede ser útil incorporar las emociones de alguna forma en el lenguaje a diseñar, lo cual fue propuesto por uno de los usuarios entrevistados.

4.1.3. 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 actores del sistema son maestras e investigadores, sin embargo el uso del sistema por parte de ambos usuarios es el mismo, con lo cual no se hace ninguna distinción entre ellos en esta especificación de casos de uso. En caso de ser necesario, para contextualizar los casos de uso, puede ser de utilidad consultar el *Manual de Usuario* [32], en ese documento, se presentan capturas de pantalla y se explica el uso de las funcionalidades implementadas.
A continuación, se especifican en alto nivel los casos de uso seleccionados.

4.1.3.1. Crear comportamiento

El caso de uso comienza cuando el usuario quiere crear un nuevo comportamiento para Robotito. Para ello el usuario presiona el botón de crear un nuevo programa. El sistema muestra la pantalla de programación con bloques. El usuario debe arrastrar y unir bloques desde la paleta de bloques al área de trabajo definiendo los distintos eventos y acciones que tendrá el comportamiento a programar. El caso de uso finaliza cuando el usuario no desea agregar más bloques a su programa.

4.1.3.2. Ejecutar comportamiento mediante la aplicación

El caso de uso comienza cuando el usuario quiere ejecutar un comportamiento en el robot mediante la aplicación móvil. Como precondición, el usuario debe estar en la pantalla de programación con bloques y haber creado o cargado algún programa, además el robot debe estar prendido y a la espera de algún comando y el dispositivo móvil del usuario debe estar conectado a la red del robot.
Para ejecutar el programa, el usuario accede al menú y presiona el botón Ejecutar. Si el programa es sintácticamente inválido se muestra un error acorde y se resaltan los bloques conflictivos, finalizando el caso de uso. Si no, si es un programa que utiliza eventos de detección de color, el sistema le muestra al usuario un cartel en donde le recomienda realizar una calibración del sensor de color.
Si el usuario accede a realizar dicha calibración, el sistema muestra en pantalla un mensaje que indica que se debe colocar el robot encima del color especificado y presionar el botón Calibrar, además se muestra una imágen descriptiva. El usuario coloca el robot encima de dicho color y presiona calibrar. Este ciclo se repite hasta que no queden más colores por calibrar, una vez finalizados los mismos el sistema le muestra al usuario un cartel indicando que se ha finalizado la calibración de color y pidiendo una confirmación para comenzar la ejecución. Si el usuario cancela la confirmación finaliza el caso de uso y si confirma, el robot comienza a ejecutar el programa.
Si por el contrario el usuario no accede a realizar la calibración de color o si directamente el programa no utiliza eventos de detección de color entonces el sistema ejecuta el comportamiento en el robot.
Mientras el robot ejecuta el comportamiento, el sistema resalta aquellos bloques de acción que el robot está ejecutando en un momento dado.
El caso de uso finaliza cuando el usuario apaga el robot.

4.1.3.3. Editar comportamiento

El caso de uso comienza cuando el usuario quiere editar un comportamiento previamente definido. Como precondición, el usuario debe estar en la pantalla de inicio y debe existir

algún programa en la lista de programas guardados.
Para editar el comportamiento el usuario presiona sobre el elemento correspondiente de la lista de programas guardados, el mismo estará identificado con el nombre de programa que el usuario le haya asignado. El sistema muestra la pantalla de programación con los bloques del programa que se está editando ya dispuestos sobre el área de trabajo, tal como el usuario lo había guardado. El usuario podrá arrastrar nuevos bloques hacia el área de trabajo, eliminar bloques, conectarlos y modificar sus valores. El caso de uso finaliza cuando el usuario no desea seguir modificando el programa.

4.2. Diseño

En esta sección se describe el diseño de las pantallas de la aplicación, el proceso de diseño del lenguaje visual (las categorías de bloques, los bloques y la forma de utilizarlos en conjunto para crear nuevos comportamientos) y finalmente la arquitectura del sistema.

4.2.1. Pantallas

Cuando el usuario accede a la aplicación se encuentra con la pantalla de inicio en donde se le muestra un listado de programas guardados (si es que tiene alguno) y un botón para crear un nuevo programa. En la figura 4.1 se presenta la pantalla de inicio y a continuación se detallan sus componentes.

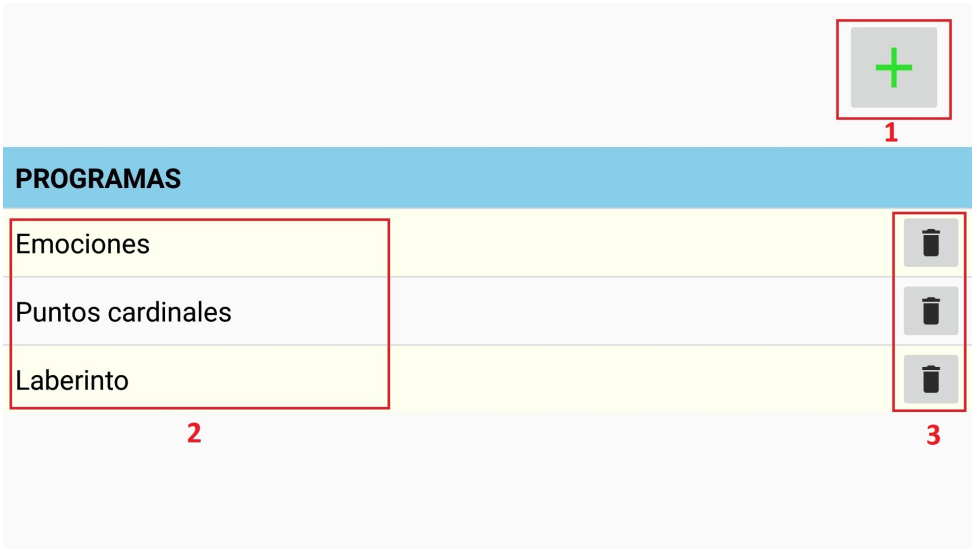


Figura 4.1: Pantalla de inicio

1. **Botón para crear nuevo programa:** Permite acceder a la pantalla de programación para crear un nuevo programa.

2. **Listado de programas:** Lista los nombres de los programas guardados por el usuario y permite acceder a cualquiera de ellos presionando sobre el elemento.

3. **Botones para eliminar programas:** Se encuentran junto a cada uno de los programas guardados por el usuario, permiten eliminar un programa guardado.

En la figura 4.2 se presenta la pantalla de programación y a continuación se detallan sus componentes.

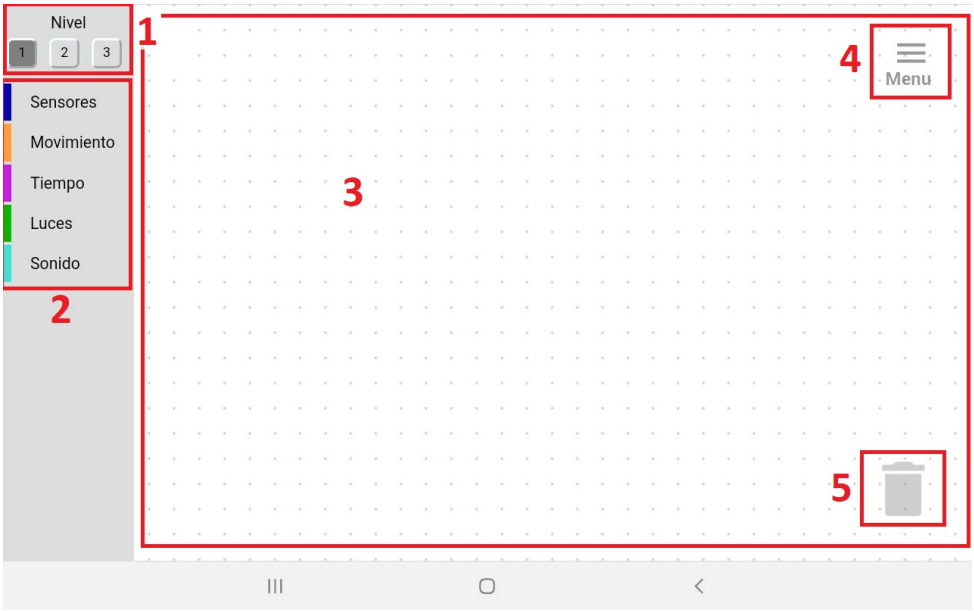


Figura 4.2: Pantalla de programación

1. **Niveles de dificultad:** Se presentan tres niveles de dificultad, el usuario puede alternar entre ellos presionando sobre los botones, lo cual habilitará (o deshabilitará) las categorías de bloques lógicos y de estado que veremos más adelante.

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

3. **Área de trabajo:** Es el lugar sobre el cual se colocan los bloques para poder armar un programa.

4. **Menú:** Permite al usuario acceder a un menú lateral con distintas opciones que permiten ejecutar un programa, guardar un programa o volver a la pantalla de inicio.

5. **Papelera:** Se utiliza para borrar un bloque o conjunto de bloques del área de trabajo. Todos los bloques arrastrados hacia ella serán eliminados y podrán ser recuperados posteriormente accediendo a la papelera.

4.2.2. Lenguaje

Para definir qué posibilidades de configuración del robot ofrecer a los usuarios, se utilizó la información obtenida a partir de las entrevistas y la planilla de reporte de actividades, tanto los datos concretos de configuración del robot como los escenarios planteados por los usuarios. Para algunos escenarios genéricos se definieron posibles formas de resolverlos. De esta forma se determinó el conjunto base de opciones de configuración a ofrecer:

- Movimientos: Desplazamiento hacia adelante, la derecha, atrás, la izquierda. Detener movimiento. Girar sobre sí mismo. Los movimientos podrían ser continuos (se mantienen en el tiempo) o por tiempo, además se les podría asignar una velocidad lenta, media o rápida.
- Luces: Prender todas de un color. Prender arcos de luces (cuatro arcos). Apagar luces.
- Sonidos: Predefinidos, asociados a emociones.
- Sensores: Detectar un color determinado. Detectar objetos con un sensor de distancia determinado. No detectar objetos con ningún sensor de distancia. Para los sensores de distancia se podría además específicar si la distancia es corta, media o larga.

Por otra parte, en base a los comentarios obtenidos en las entrevistas, se decidió señalizar el frente del robot (con un sticker en forma de flecha) así como enumerar los sensores de distancia como se puede observar en la figura 4.3.

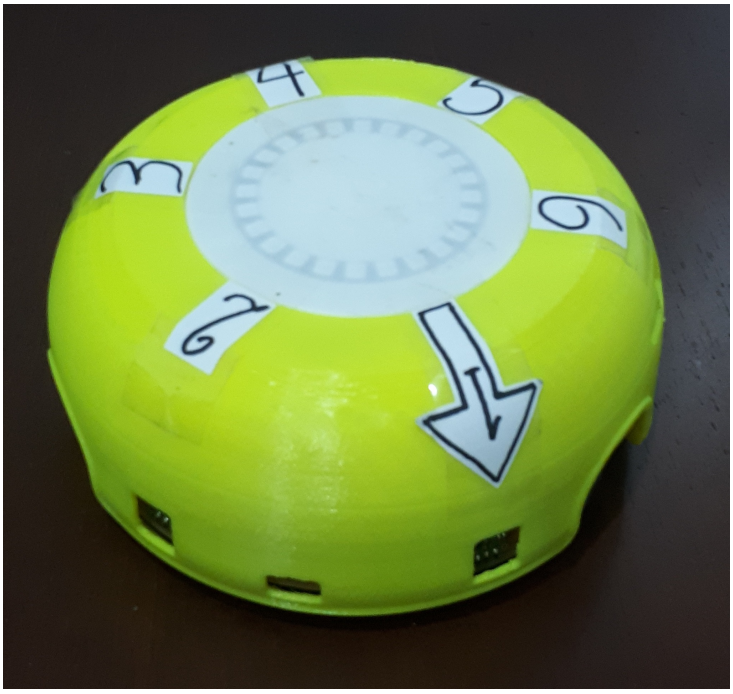


Figura 4.3: Robotito con stickers

Esto mejorará la interacción de los usuarios con el robot y la aplicación, facilitando la programación de los sensores de distancia, los movimientos y evitando la desorientación de los usuarios en cuanto al posicionamiento del robot.

Luego se realizaron bocetos de íconos para el diseño de los bloques, basados en las opciones de configuración determinadas anteriormente. En la figura 4.4 se observan dichos bocetos.

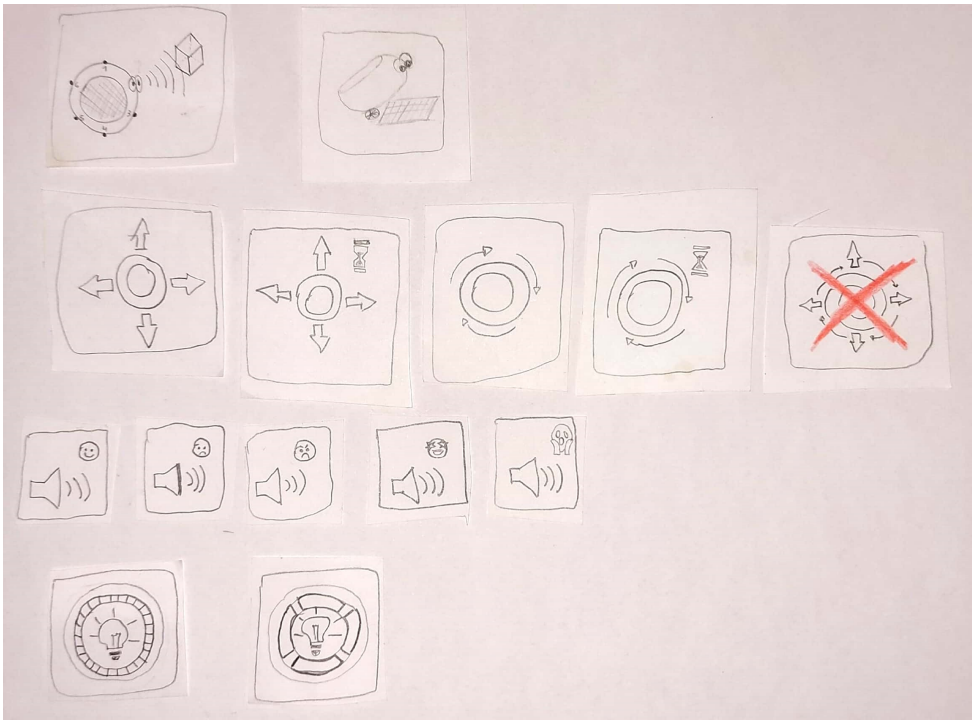


Figura 4.4: Bocetos de íconos para el diseño de los bloques

En etapas iniciales del diseño, se tomó la decisión de desacoplar la velocidad de los movimientos, generando un nuevo bloque para configurar la velocidad, lo cual permite modificar la velocidad de movimiento sin importar cuál sea el movimiento actual. Del mismo modo se desacopló el tiempo de los movimientos, generando una nueva categoría de *Espera* con un bloque que representa el paso del tiempo. Esto permite no solo realizar movimientos por tiempo sino acciones en general separadas por tiempo, en secuencia. Ambas decisiones aumentan la expresividad del lenguaje y simplifican la representación de los bloques de movimiento así como la posterior configuración de dichos bloques.

De esta forma se obtuvo el conjunto de bloques inicial del lenguaje:

1. **Detección de color:** Representa el evento de detectar un determinado color en el suelo.

2. **Detección de objeto:** Representa el evento de detectar un objeto con alguno de

los seis sensores de distancia, o no detectar ningún objeto con ningún sensor.

3. **Ir hacia:** Acción de ir hacia una determinada dirección, que puede ser: adelante, derecha, atrás, izquierda.
4. **Girar:** Acción de girar sobre sí mismo.
5. **Velocidad:** Acción de cambiar la velocidad de movimiento a lenta, media o rápida.
6. **Detenerse:** Acción de detener los movimientos.
7. **Esperar:** Acción que representa el paso del tiempo y en conjunto con otros bloques nos permite crear acciones en función del tiempo. Puede ser espera de uno, dos o cinco segundos.
8. **Prender luces:** Acción de prender todas las luces de un determinado color.
9. **Luces por arcos:** Acción de prender las luces del robot por sectores (o arcos), existiendo cuatro arcos: delantero, derecho, trasero, izquierdo.
10. **Apagar luces:** Acción de apagar todas las luces.
11. **Iniciar sonido:** Acción de iniciar un sonido que puede ser feliz, triste, asustado o enojado.

Los bloques se pueden ver en la figura 4.5 debajo de sus correspondientes categorías y enumerados de acuerdo al listado anterior.

Figura 4.5: Enumeración de bloques iniciales

Como se puede observar, en un principio se optó por un diseño horizontal del lenguaje en donde los bloques se conectan horizontalmente, colocando los eventos primero y luego conectando las acciones hacia la derecha.

Para aquellos bloques que sean configurables el usuario podrá presionar sobre los mismos y se desplegará un menú flotante junto al bloque con las opciones de configuración del mismo (esto lo veremos en detalle en la sección 4.2.2.4). El ícono del bloque se actualizará de acuerdo a las opciones elegidas, en la figura 4.6 se pueden ver algunas variantes de los distintos íconos que muestran los bloques configurables. Las imagenes utilizadas son SVG de licencia libre o autoría propia. Las imágenes SVG no pierden calidad al hacer zoom, además es posible modificar las distintas partes de la imágen de forma programática. Esto fue escencial particularmente para el bloque de *Luces por arcos*, el cual puede representar 9^4 valores distintos.

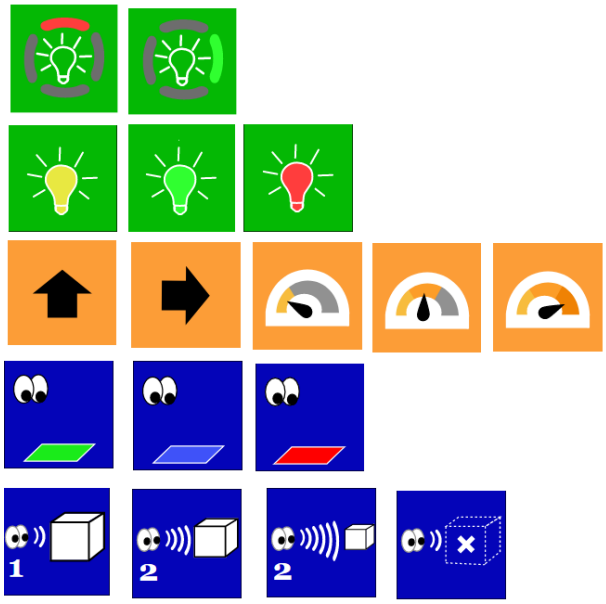


Figura 4.6: Íconos representando distintos valores de los bloques configurables

4.2.2.1. Evaluaciones con usuarios y mejoras al lenguaje

Las categorías y bloques fueron impresos en papel, también se prototiparon en papel los menúes de configuración de cada bloque. Con este prototipo se realizó un proceso iterativo de evaluaciones con usuarios y mejoras al lenguaje.

Las evaluaciones con usuarios fueron presenciales, con un único usuario cada vez, con una duración aproximada de una hora. Las mismas fueron grabadas en video y reproducidas posteriormente para obtener más información. El procedimiento de evaluación estuvo enfocado en el significado y la ubicabilidad de los íconos [18] y en determinar si se utilizan de forma correcta en conjunto. Para ello se realizó el siguiente procedimiento:

1. Presentación del robot: Se presenta el robot, sus distintas partes y sus posibilidades de acción y sensado.

2. Evaluación de íconos: Se muestran los distintos íconos del lenguaje, uno a uno. El objetivo fue que el usuario describa su significado tan pronto como fuera posible.

3. Introducción a la programación dirigida por eventos: Se muestra un video de un minuto que explica cómo se arma un programa dirigido por eventos en Scratch, de esta forma se evita el contacto del usuario con el lenguaje antes del comienzo de la evaluación.

4. Resolver escenarios: Se le planteó a los usuarios escenarios a resolver programando con los bloques en papel. Se realizó una simulación con el prototipo en papel, en donde el usuario pudo interactuar con el mismo.

5. Lectura de programas pre-armados: Se mostró un programa armado en papel, el objetivo fue que el usuario al ver los bloques pueda describir dicho programa.

En los *Anexos* de este documento se describe detalladamente como fueron las instancias de evaluación con los usuarios, qué retroalimentación se obtuvo en cada una de ellas y qué modificaciones se realizaron al lenguaje.

En la figura 4.7 se muestran las categorías y los bloques luego de los cambios realizados a partir de la retroalimentación obtenida durante las pruebas.

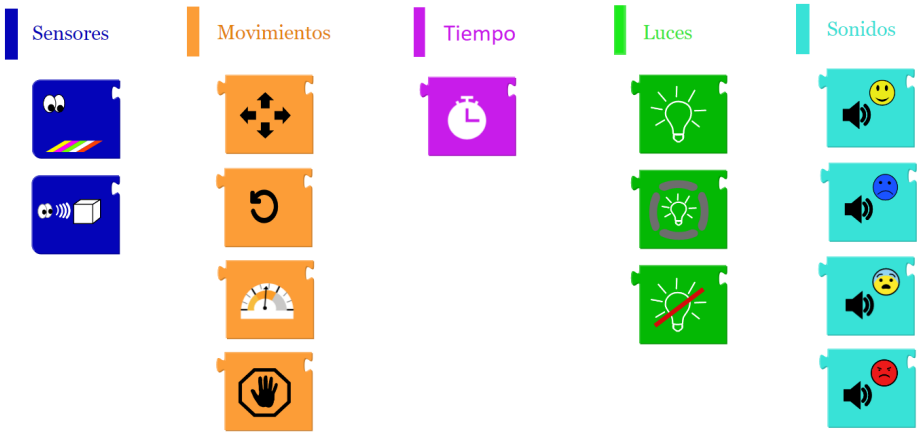


Figura 4.7: Categorías y bloques luego de las evaluaciones con usuarios

Por otra parte, durante las pruebas se tomó la decisión de que al soltar un nuevo bloque en el área de trabajo, si el bloque tiene opciones de configuración entonces se mostrará de forma automática el menú de configuración, incitando al usuario a elegir un valor inicial y también agilizando la programación.

4.2.2.2. Bloques lógicos

En etapas más avanzadas del proyecto, una vez que se tiene implementado en la aplicación móvil el lenguaje con los bloques y categorías descritos anteriormente, se decide ampliar el lenguaje añadiendo una nueva categoría de bloques lógicos con los bloques *Y/O* y *NO*. El bloque *Y/O* al tomar el valor "Y" sirve para definir eventos más complejos formados por más de una condición, por ejemplo el evento de "detectar color azul Y detectar un objeto en el sensor 1 a distancia corta" que se dispara solamente si se cumplen ambas condiciones. El bloque *Y/O* al tomar el valor "O" permite definir varios eventos distintos que desencadenarían la misma secuencia de acciones, por ejemplo el evento de "detectar un objeto en el sensor 1 a distancia corta O detectar un objeto en el sensor 1 a distancia media", dicho evento sucederá si se cumple una condición o la otra, en ambos casos desencadenará la misma secuencia de acciones. Finalmente, el bloque *NO* permite definir el evento de la negación de un evento, por ejemplo el evento de "NO detecta un objeto en el sensor 1 a distancia corta", que se disparará siempre y cuando el robot no detecte ningún objeto en el sensor 1 a distancia corta.

En esta etapa surgieron dificultades con el diseño de los bloques lógicos en el lenguaje horizontal. Se evaluaron distintas posibilidades de diseño y se encontraron algunas limitaciones de Blockly que dificultaron encontrar una buena solución de diseño en un lenguaje horizontal tal como se pretendía en un principio. La principal limitación fue la imposibilidad en Blockly de definir un bloque con inputs internos y externos, es decir un bloque que permita colocar otros bloques dentro de sí mismo (en forma horizontal) y además pueda conectar con bloques hacia la derecha. Esta solución hubiese sido ideal ya que permitiría definir bloques lógicos que anidan bloques de sensor o otros bloques lógicos dentro de sí mismos y luego permiten conectar las acciones a ejecutar hacia la derecha (tal como los bloques de sensor).

Luego de evaluar distintas alternativas, se decide modificar los conectores de los bloques para pasar a un lenguaje en vertical. Este cambio se realiza de forma muy sencilla modificando los conectores en la definición de los bloques. Las acciones se conectarán debajo de los eventos en lugar de conectarse hacia la derecha, para ello incorporan un conector de tipo *PreviousStatement* y un conector *NextStatement*, además se elimina el conector de tipo *output* y el *ValueInput* externo. En la figura 4.8 se puede observar un programa creado con los bloques modificados para conectar en vertical.

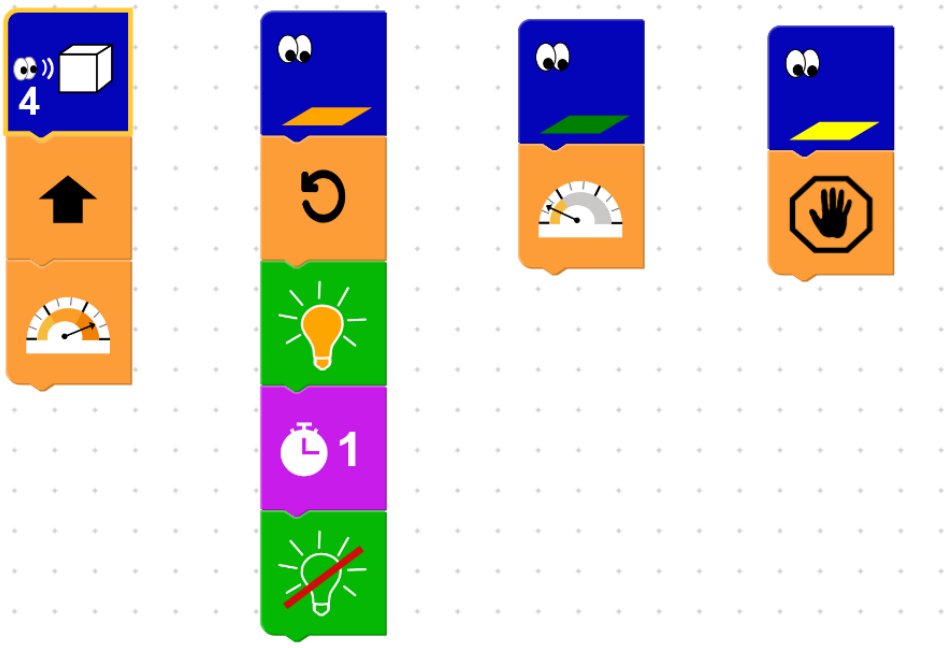


Figura 4.8: Programa hecho en la versión vertical del lenguaje

En esta versión vertical del lenguaje se definen los bloques lógicos que tienen tres conectores distintos. Los conectores internos, de tipo *ValueInput*, permiten colocar bloques de sensor o bloques lógicos dentro de sí mismos. El conector inferior, de tipo *NextStatement*, permite colocar acciones a continuación de un bloque lógico. El conector izquierdo, de tipo *output*, permite conectar un bloque lógico dentro de otro bloque lógico. En la figura 4.9 se pueden observar los bloques lógicos resultantes.

Figura 4.9: Bloques lógicos "Y/O" y "NO"

Para poder conectar bloques de sensor dentro de un bloque lógico los mismos incorporan un conector de tipo *output*, además del conector *NextStatement* que ya tienen para conectar acciones, los bloques resultantes se muestran en la figura 4.10.

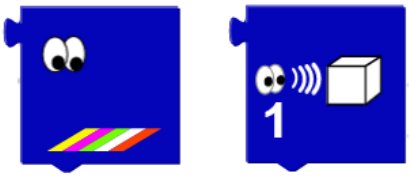


Figura 4.10: Bloques de sensor con *output*

El problema de este diseño es que los bloques de sensor y los bloques lógicos tienen dos roles posibles ya que pueden ir dentro de un bloque lógico o por separado. Además no tendría sentido conectar un bloque de sensor con bloques de acción y luego conectar dicho bloque y sus acciones dentro de un bloque lógico. Para afrontar este problema se modifican los conectores de los bloques de forma programática, si un bloque de lógica o de sensor se coloca dentro de otro bloque de lógica, entonces pierde su conector *NextStatement* y ya no se puede conectar con bloques de acción. En el momento en que dicho bloque se retira de dentro del bloque lógico, el mismo recupera su conector *NextStatement* y puede conectarse con bloques de acción (ver figura 4.11). Asimismo, si a un bloque lógico o de sensor se le conecta algún bloque de acción, entonces dicho bloque pierde su conector *output*, es decir que ya no se puede colocar dentro de un bloque lógico. Al desconectar las acciones el bloque recupera su conector *output*. En la figura 4.11 se pueden observar las distintas configuraciones de los conectores de estos bloques de acuerdo a la situación en la que estén.

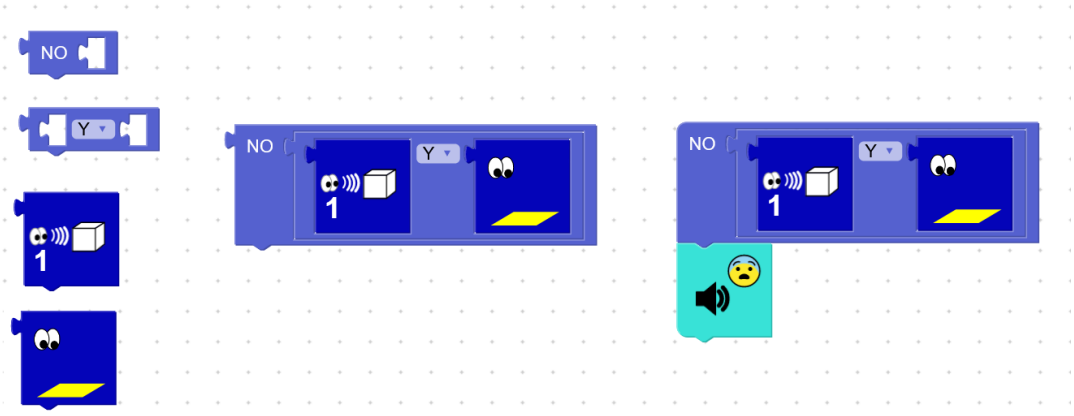


Figura 4.11: Cambios en los conectores de los bloques lógicos y de sensor

La modificación de los conectores de los bloques de forma programática no es una funcionalidad que venga por defecto implementada en Blockly. Tampoco se encontraron otras implementaciones Blockly que recurran a esto. A modo de ejemplo, en la implementación de Thymio VPL con Blockly se tienen bloques de evento que no admiten bloques lógicos, es decir eventos simples, como teníamos en la versión inicial del lenguaje. Por otra parte, esta implementación de Thymio también ofrece bloques lógicos "if", "and/or", "not" y bloques de sensor, sin embargo la forma de utilizarlos en conjunto para crear condiciones lógicas siempre implica la utilización del bloque "if", el cual cuenta con un conector *StatementInput* que permite conectar las acciones a realizar cuando se cumpla la condición lógica. Todos estos bloques se pueden ver en la figura 4.12.

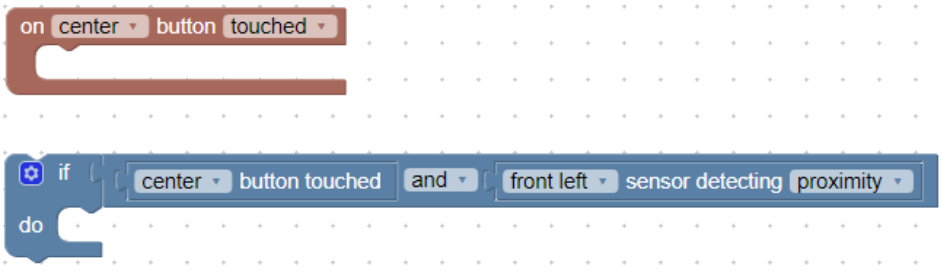


Figura 4.12: Bloques de evento y lógicos en Thymio

Se decide optar por el diseño de bloques lógicos y de sensor con conectores cambiantes para evitar incurrir en la redundancia de utilizar un bloque "if" cada vez que se desea definir un evento. Tampoco se desea contar con distintos tipos de bloques de sensor, como en Thymio VPL, que tiene bloques de sensor con conector *StatementInput* y luego el mismo bloque de sensor pero con conector *output* para conectar dentro de bloques lógicos, esto no solo añade más bloques al lenguaje sino que también puede resultar confuso tener dos bloques con el mismo ícono y distintos conectores.

Esta decisión permite a los usuarios realizar una programación más ágil. Por otra parte los usuarios no tendrán que preocuparse por los conectores de los bloques ya que el sistema ocultará o mostrará los conectores de forma automática cuando corresponda, evitando conexiones erróneas.

4.2.2.3. Bloque de *Estado* y niveles de dificultad

Si bien no se planteó en ningún momento como un requerimiento del proyecto, se decide añadir también una categoría y un bloque de *Estado* que permite al usuario definir máquinas de estado y comportamientos variables frente a un mismo evento. Esta decisión se toma en base a los planteos de algunos usuarios que habían comentado que les sería útil tener esa posibilidad, principalmente los investigadores.

Tanto esta adición como la de los bloques lógicos permiten ampliar mucho la expresividad del lenguaje en torno a la definición de eventos, la cual estaba inicialmente bastante limitada. En contraparte dichas adiciones le añaden una mayor complejidad al lenguaje y es posible que algunos usuarios no precisen esta funcionalidad, sobre todo entre las maestras, que por lo general plantearon escenarios más simples. Es por esta razón que se decide añadir distintos niveles de dificultad al lenguaje, teniendo el nivel 1 que cuenta con los bloques iniciales, el nivel 2 que añade bloques lógicos y el nivel 3 que añade el bloque de *Estado*. El usuario podrá alternar entre los distintos niveles si lo desea, de esta forma a priori se ocultan los bloques más complejos, simplificando el lenguaje pero poniendo esos bloques a disposición de aquellos usuarios que los necesiten.

Para representar los distintos estados se utilizan imágenes de la temática de animales, así como se podría haber utilizado cualquier otra representación abstracta de un estado. Por defecto el bloque de *Estado* tendrá el valor "INICIAL", que se representa visualmente con el ícono de un huevo, a modo de metáfora. El usuario podrá seleccionar cualquier otro estado dentro del menú de configuración del bloque, el cual veremos más adelante. En la figura 4.13 se puede observar el bloque de *Estado*.

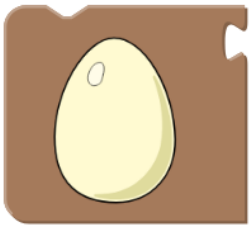


Figura 4.13: Bloque de *Estado* representando el estado inicial

Como se puede observar el bloque de *Estado* cuenta con dos conectores. El conector *PreviousStatement* le permite conectarse a continuación de cualquier bloque de acción, de lógica o de sensor, en este caso el bloque de *Estado* representa la acción de trancisionar a un nuevo estado y será la última acción realizada dentro de dicha secuencia de acciones pues no tiene un conector *NextStatement*. El conector *ValueInput* externo le permite conectarse a cualquier bloque de sensor o lógico (es decir a cualquier evento), en este caso el bloque de *Estado* indica que dicho evento solamente será tenido en cuenta cuando el robot se encuentre en el estado correspondiente.
Para que la conexión del bloque de *Estado* con un bloque lógico o un bloque de sensor sea posible, es necesario que dichos bloques siempre cuenten con el conector *output*, lo cual hasta ahora no siempre sucedía ya que como vimos anteriormente dicho conector se oculta en el momento en que se conectan acciones con el bloque lógico o de sensor. Es por esto que cuando un usuario accede al nivel 3 del lenguaje y habilita el bloque de *Estado*, al mismo tiempo se habilitarán los conectores *output* de los bloques lógicos y de sensor, sin importar si tienen acciones asociadas.

Al igual que se hizo para los bloques de lógica, se opta por un diseño del bloque con conectores cambiantes, en donde se oculta un conector en el momento en que el otro conector está siendo utilizado. Una alternativa a este diseño hubiera sido contar con dos bloques distintos, uno con conector *PreviousStatement* y el otro con el conector *ValueInput* externo. Contar con un único bloque adaptable será más simple para los usuarios.

Para entender mejor cómo se puede definir una máquina de estados con este nuevo bloque

podemos ver la figura 4.14 en donde se tiene un programa con estados y la correspondiente máquina de estados equivalente.

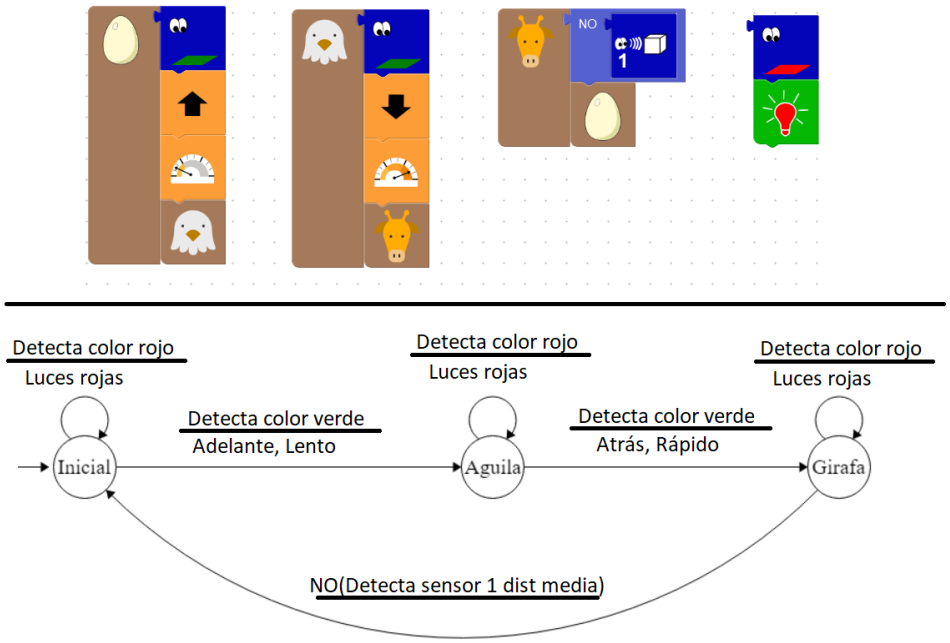


Figura 4.14: Programa con estados y su equivalente representación con máquina de estados

Como se puede observar en la imágen, existe la posibilidad de declarar eventos que no estén asociados a ningún estado, es decir que siempre serán tenidos en cuenta sin importar el estado en el que esté actualmente el robot, tal es el caso del evento de "Detectar color rojo". También podemos apreciar como es posible asociar dos actuaciones distintas al mismo evento de "Detectar color verde" tal como habían solicitado los usuarios investigadores.

4.2.2.4. Menúes de configuración de los bloques

Aquellos bloques que tengan opciones de configuración desplegarán un menú de forma automática cuando el usuario los coloque en el espacio de trabajo por primera vez.

Para el diseño de los menúes de configuración se optó por opciones gráficas, que no requieran que el usuario sepa leer o escribir, ofreciendo botones u otros elementos gráficos con los que puede interactuar y luego observar los cambios en el bloque cuyo valor se ve afectado.

Para algunos de los bloques se ofrece un pequeño menú de configuración flotante, de tipo *DropDownDiv*, que aparece por debajo del bloque como se puede observar en la figura 4.15. Se hicieron dos diseños de menú flotante, uno de ellos permite seleccionar un color, el otro permite aumentar o disminuir el valor del bloque mediante dos botones de "-" y de

“+”.

Figura 4.15: Menúes de configuración

Para los bloques de *Detección de objeto*, *Ir hacia*, *Luces por arcos* y *Estado* se implementaron pantallas flotantes de configuración, de tipo *WidgetDiv*, debido a que no se encontraron opciones eficientes de configuración que pudieran ser incluídas en un menú más pequeño junto al bloque.
Estas pantallas se diseñaron especificamente para cada uno de los bloques y sus opciones particulares de configuración. Cuentan con un texto descriptivo y una imágen SVG descriptiva que se actualiza a medida que el usuario selecciona las distintas opciones de configuración que se ofrecen, ya sea en forma de botones o interactuando directamente con la imágen.

Figura 4.16: Pantallas flotantes de configuración

El resto de los bloques simplemente muestran un texto descriptivo de su significado, podemos ver algunos ejemplos en la figura 4.17.

Figura 4.17: Diálogos descriptivos de los bloques sin opciones de configuración

4.2.3. Arquitectura

En esta sección se detalla la arquitectura de la aplicación móvil, en la sección 4.3.2 se describen los módulos implementados en Robotito y su funcionamiento.

En la Figura 4.18 se observan los principales componentes de la aplicación. A continuación, se describen brevemente los mismos.

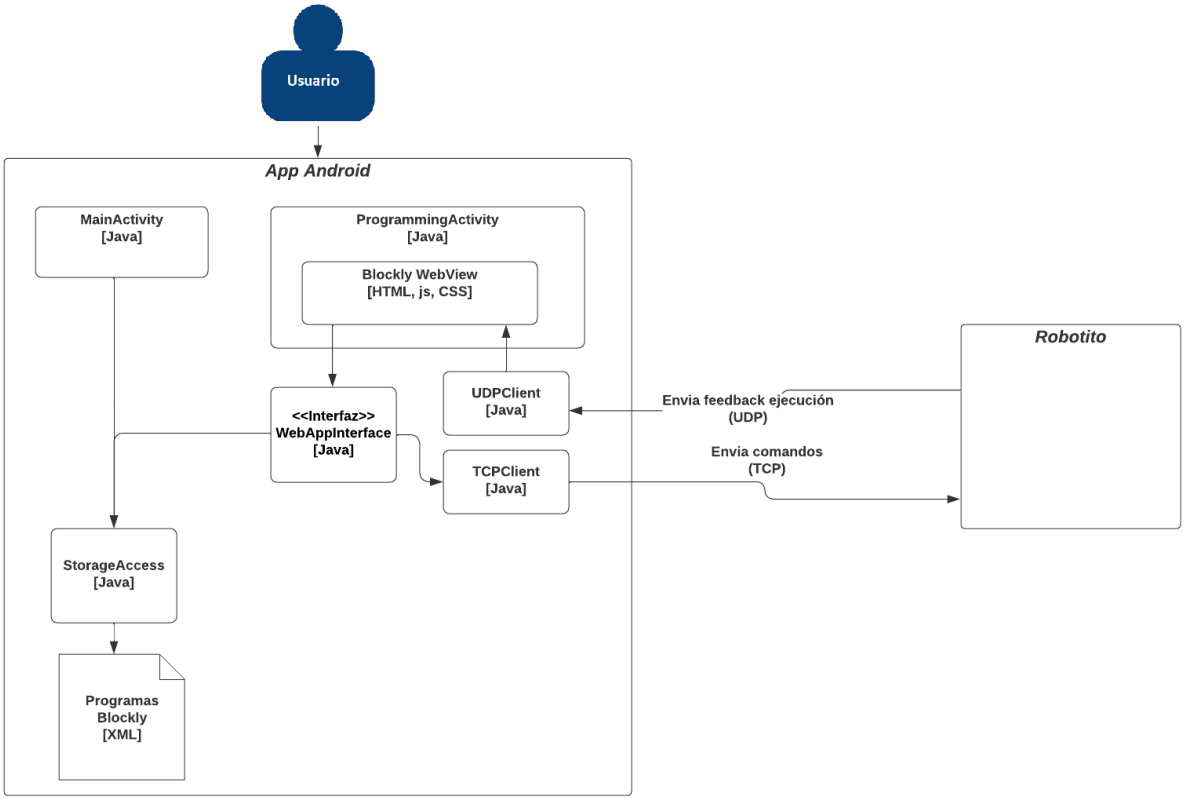


Figura 4.18: Diagrama de arquitectura de la aplicación móvil

- **MainActivity:** Define la pantalla de inicio. Se muestra la lista de programas guardados, los botones de eliminar programa, el botón de nuevo programa. Se asocian eventos a los botones y a los elementos de la lista de programas.
- **ProgrammingActivity:** Contiene el componente WebView utilizado para incluir contenido web en la aplicación. Dentro del WebView está la implementación del lenguaje con Blockly y la definición de los distintos elementos de la pantalla de programación.
- **WebAppInterface:** Interfaz que define las funciones que se exponen para ser invocadas desde el WebView. Dichas funciones permiten guardar programas a través del StorageAccess y enviar distintos comandos al robot mediante un socket TCP.
- **TCPClient:** Provee funcionalidades para establecer comunicación con el robot

mediante un socket TCP. Se utiliza para enviar comandos al robot y recibir una respuesta.

- **UDPClient:** Recibe paquetes UDP enviados por el robot. En dichos paquetes se recibe información que indica los bloques a resaltar y estado actual del robot, esa información se la envía al WebView mediante las funciones definidas en el archivo *androidInterface.js*, que veremos más adelante.
- **StorageAccess:** Provee funcionalidades para listar, guardar, cargar y eliminar programas blockly en el sistema de archivos del dispositivo Android.

Para la comunicación de los comandos al robot se utiliza un socket TCP ya que se desea garantizar la recepción de los comandos por parte del robot, además de que facilita la detección de problemas de comunicación con el mismo. Luego, para la recepción de los datos de ejecución del robot, se utiliza un socket UDP ya que no se desea enlentecer la ejecución del robot y además estos datos no son esenciales, no sería tan grave que algún paquete UDP no llegue al dispositivo móvil, más allá de esto en todas las pruebas que se han realizado durante el desarrollo del proyecto no se detectó ningún caso en el que esto haya sucedido.

La interfaz WebAppInterface define las funciones que se exponen al WebView, luego la implementación de dicha interfaz está en la clase WebAppServices, la misma no se muestra en el diagrama de arquitectura para simplificar.

4.3. Implementación

En esta sección se presentan detalles de implementación de la aplicación móvil y también detalles de la implementación en Robotito.
En la aplicación móvil se puede distinguir entre dos partes, una parte se encuentra implementada en Java utilizando componentes nativos de la API de Android, la otra parte incluye a Blockly y corresponde a una implementación web, es decir que contiene HTML, css y javascript, esta parte fue integrada mediante el uso del componente WebView de Android. La implementación en Robotito se realizó en lenguaje Lua.

Los componentes MainActivity y ProgrammingActivity son actividades, es decir que extienden la clase Activity de Android [5]. Las actividades generalmente permiten definir los elementos de la interfaz gráfica de una única pantalla en la aplicación, tal es el caso de las actividades MainActivity y ProgrammingActivity. La actividad ProgrammingActivity se encarga de inicializar y mostrar en pantalla el WebView, también se encarga de asociar una instancia de WebAppServices a dicho WebView para que el mismo pueda acceder al dispositivo a través de esta interfaz.

Las funciones que se exponen en la WebAppInterface para ser invocadas desde el WebView son:

- **obtenerPrograma**: Retorna el XML del programa cargado en formato String (si es que se cargó un programa).
- **obtenerNombrePrograma**: Retorna el nombre del programa cargado en formato String (si es que se cargó un programa).
- **guardarPrograma**: Permite guardar un programa en el almacenamiento del dispositivo.
- **escribirNvs**: Permite enviar un comando para persistir un dato en el almacenamiento no volátil del robot, esto puede resultar útil para definir aspectos de configuración del robot.
- **calibrar**: Permite enviar el comando de calibración de color al robot, junto con un parámetro que es el color que se desea calibrar.
- **mandarCodigo**: Permite enviar comandos para transferir código al robot, dicho código será grabado en el robot en un archivo denominado *vplProgram.lua*.
- **ejecutar**: Permite enviar un comando para iniciar la ejecución en el robot.

Actualmente la función *escribirNvs* se está utilizando únicamente para enviarle al robot los colores a sensar en el programa actual, es decir aquellos que se utilizan en bloques de *Detección de color*. Este dato se envía previo al comienzo de la ejecución y como veremos más adelante, permite que el robot al inicializar el sensor de color solamente cargue estos colores, evitando posibles conflictos con otros colores que no se estén utilizando.

Cada una de las funciones que implican una comunicación con el robot se realizan a través del socket TCP, el protocolo de comunicación con el robot se describe en forma detallada en la sección 4.3.2.5.

En cuanto a la persistencia en la aplicación, la misma se realiza por medio de archivos XML en el sistema de archivos del dispositivo Android donde ejecuta. Los archivos XML son utilizados para guardar programas blockly.

4.3.1. Blockly

El objetivo de esta sección es describir en detalle la forma en la que se integró Blockly y las características de esta librería que fueron utilizadas.

4.3.1.1. Integración

Para la integración de Blockly en la aplicación se siguieron las recomendaciones del equipo de Google Blockly de embeber la versión web de la librería dentro de un WebView [12]. Para ello se tomo como punto de partida un proyecto desarrollado por el equipo de Google Blockly, para Android Studio, que marca los lineamientos a seguir para integrar Blockly de esta forma [8].

Para acceder desde Blockly a funcionalidades nativas de Android como ser el uso de sockets o el acceso al almacenamiento, se utiliza la clase *WebAppServices*.
La *ProgrammingActivity* es quien se encarga de asociar una instancia de *WebAppServices* al WebView, esto se logra mediante la función *WebView.addJavascriptInterface* [6]. Dicha instancia podrá ser referenciada desde dentro del WebView llamando a "WebAppInterface".
A modo de ejemplo y para entender mejor la comunicación entre los distintos elementos del sistema, se presenta un diagrama de secuencia que representa la funcionalidad de guardar un programa (ver Figura 4.19).

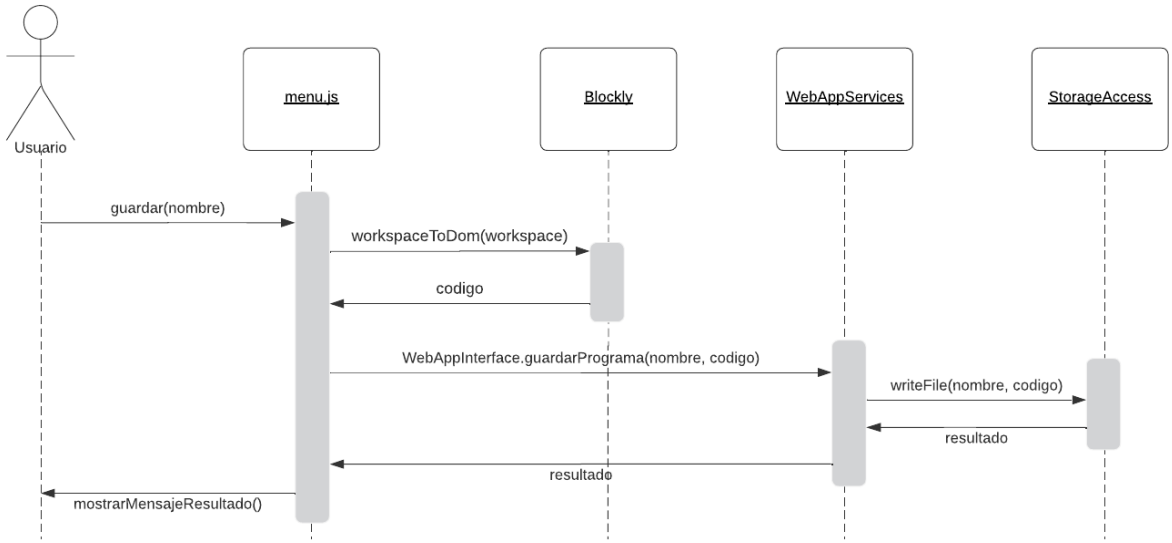


Figura 4.19: Diagrama de secuencia de guardado de programa

En el archivo *menu.js* se encuentra la implementación de las funciones asociadas a los botones del menú de la pantalla de programación.

Para la comunicación desde el UDPClient (que recibe información de la ejecución en el robot) hacia el WebView, se invoca a funciones javascript definidas en el archivo *androidInterface.js* dentro del WebView, en particular la función *notificarEstado* y *notificarResaltado* que sirven para actualizar el estado actual y resaltar los bloques ejecutados en la pantalla de programación. A modo de ejemplo, se presenta un diagrama de secuencia

en donde se puede observar las interacciones que suceden cuando se recibe un paquete en el socket UDP, en particular cuando se recibe un paquete con información del estado actual del robot (ver Figura 4.20).

Figura 4.20: Diagrama de secuencia de actualización de estado actual del robot

4.3.1.2. Definición de bloques

Una de las características más importantes de Blockly es la definición de nuevos bloques. Para la definición de bloques se utilizó código Javascript, a modo de ejemplo en la figura 4.21 se puede observar la definición del bloque *Prender luces*.

```
Blockly.Blocks[BLOQUE_PRENDER_LUCES] = {
  init: function() {
    this.setStyle('loop_blocks');
    this.setOutput(false);
    this.appendDummyInput('DUMMY').setAlign(Blockly.ALIGN_CENTRE)
        .appendField(new CustomFields.FieldPrenderLuces(), BLOQUE_PRENDER_LUCES);
    this.setNextStatement(true);
    this.setPreviousStatement(true);
    this.setColour(COLOR_LUCES);
  }
};
```

Figura 4.21: Definición del bloque *Prender luces*

Las funciones *setOutput*, *setPreviousStatement* y *setNextStatement* permiten definir los conectores del bloque. La función *setColour* permite definir el color del bloque, en este caso se aplicó el color de la categoría de Luces. Para la definición de la mayoría de los bloques se utilizó un *DummyInput*, que no tiene ningún conector de tipo *input*, pero permite añadir el campo. Los campos del lenguaje fueron creados desde cero, en este ejemplo se utiliza el campo *CustomFields.FieldPrenderLuces*. Este campo en particular define los componentes SVG de la imágen de la lamparilla de colores que se puede apreciar en el bloque de *Prender luces*. Además se define un editor de tipo *DropDownDiv* y se

define manualmente dentro de dicho editor un selector de colores, que consiste en ocho cuadrados con los colores del lenguaje. A cada cuadrado se asocian eventos, estos eventos resaltan el cuadrado seleccionado, modifican el valor del campo y también modifican el color del elemento SVG que representa la bombilla dentro de la imágen SVG, esto sirve para reflejar el nuevo valor del campo.
Para la definición de los campos se siguieron los lineamientos planteados por el equipo de Google Blockly [10]. Si bien la definición de nuevos campos basados en imágenes SVG con editores personalizados, requirió un esfuerzo extra, se obtienen varios beneficios. Se aprovecha mejor el espacio dentro del bloque ya que se trasladan los elementos de configuración hacia el menú desplegable, se cuenta con imágenes que se ajustan al zoom del dispositivo sin perder calidad, se puede reconocer un bloque y su valor rápidamente con un vistazo y sin la necesidad de leer, se abre las puertas a la posibilidad de que quienes que no sepan leer igualmente puedan programar, se obtienen soluciones de diseño a bloques con muchos elementos configurables como ser el bloque de *Luces por arcos*.

4.3.1.3. Corrector sintáctico

Se implementa un corrector sintáctico que permite detectar errores de sintaxis cuando un usuario intenta ejecutar un programa en el robot. En ese momento, si existe algún error sintáctico se le muestra un mensaje descriptivo al usuario y se centra y resalta el bloque que está en falta. Los errores de sintaxis corresponden a: bloques sueltos, bloques lógicos sin rellenar, eventos sin acciones, acciones sin eventos.

4.3.1.4. Generación de código

El código generado a partir de los bloques define una estructura de if/elseif en donde las condiciones booleanas a evaluar dentro de cada if/elseif están dadas por los bloques que definen los eventos (bloques de sensor, de lógica y de estado) y el código a ejecutar dentro de cada if/elseif está dado por los bloques de acción (movimientos, luces, sonidos, tiempo, estado) asociados al evento.

Esta estructura de if/elseif será ejecutada en un bucle infinito de detección de eventos en el robot. En cada iteración, el robot evalúa las condiciones dentro de los if/elseif y en caso de que se cumpla alguna de esas condiciones ejecutará el código correspondiente, que define las acciones a realizar.
Para lograr un código desacoplado de la implementación del robot, se propone que cada bloque genere un código de tipo String que haga referencia a una función abstracta, cuya implementación deberá ser realizada en el robot. Por ejemplo, el bloque *Girar* genera el código "GIRAR()", el bloque de *Detección de color* genera el código "SENSOR_COLOR(color)", en donde "color" corresponde al valor que tenga el campo del bloque. En particular la

función “GIRAR()” deberá activar los motores del robot para realizar un giro y la función “SENSOR_COLOR(color)” deberá retornar un booleano dependiendo de si el color actualmente detectado por el robot es igual al color pasado por parámetro.

Para resolver el requerimiento funcional de “Retroalimentación de la actuación del robot”, se incluyen dentro del código a enviarle al robot, llamadas a la función “NOTIFICAR(id)”. El robot deberá implementar dicha función, que enviará a la aplicación móvil mediante un socket UDP el identificador del bloque a resaltar. La notificación no se realizará para todos los bloques ejecutados dado que el robot ejecuta las acciones de forma inmediata, excepto por la acción de *Tiempo*, con lo cual no tendría sentido resaltar cada uno de los bloques de acción durante unas pocas milésimas de segundo. Se decide entonces añadir el llamado a la función “NOTIFICAR(id)” solamente para el primer bloque de una secuencia de acciones y para todo bloque consecutivo a un bloque de *Tiempo*. Al momento de resaltar los bloques, se resaltará el bloque de acción notificado y todas las acciones consecutivas hasta llegar al final de la secuencia de acciones o a un bloque de *Tiempo*.

Para la generación de código se definió un nuevo generador *Blockly.RobotitoVPL* y se definió el código a generar por cada uno de los bloques del lenguaje. Reutilizar el generador de código Lua no hubiese aportado beneficios ya que todos los bloques que se utilizan son bloques nuevos. En la figura 4.22 se puede observar la definición del código a generar por el bloque de *Prender luces*.

```
Blockly.RobotitoVPL['ACCION_PRENDER_LUCES'] = function(block) {
    let rgb = Colors.RGB(block.getFieldValue(block.type));
    let code = "PRENDER_LUCES(" + rgb[0] + ',' + rgb[1] + ',' + rgb[2] + ")";
    return Blockly.RobotitoVPL.addNotificacion(code, block);
};
```

Figura 4.22: Definición de código a retornar para el bloque *Prender luces*

En particular para la generación de código de los bloques de acción se utiliza una función auxiliar *addNotificacion* que fue definida dentro del generador de código. Esta función verifica si el bloque es hijo de un bloque de *Tiempo* o de algún bloque lógico o de *Detección de color* o *Detección de objeto*, en cuyo caso añade una línea de código invocando a la función “NOTIFICAR(idBloque)”.

En la figura 4.23 se muestra un ejemplo de un programa que utiliza todos los bloques del lenguaje y a continuación veremos el código que genera dicho programa.

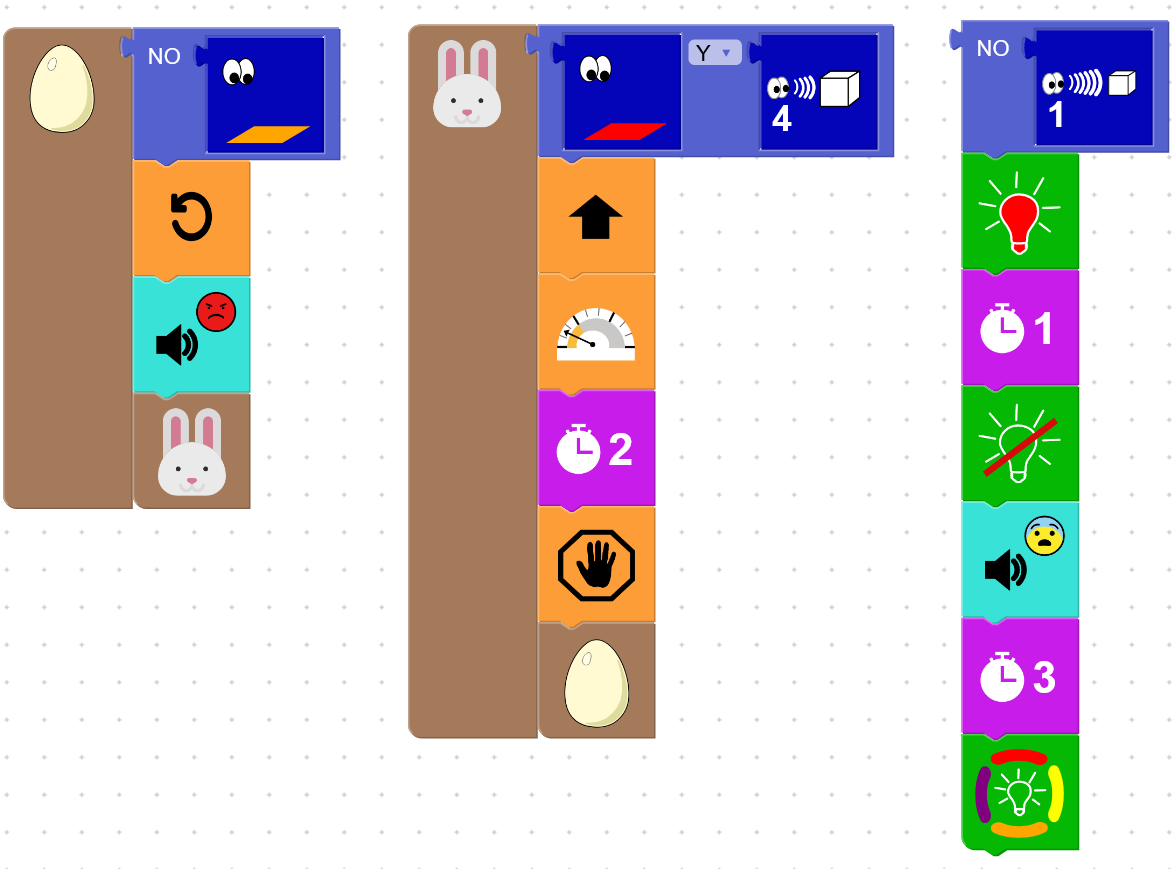


Figura 4.23: Programa de ejemplo para la generación de código

A continuación, en la figura 4.24 se muestra el código resultante del programa de la figura 4.23.

```
if(IS_ESTADO('INICIAL') and not(SENSOR_COLOR('NARANJA'))) then
NOTIFICAR(2)
GIRAR()
SONIDO('ENOJADO')
SWITCH_ESTADO('CONEJO')
elseif(IS_ESTADO('CONEJO') and (SENSOR_COLOR('ROJO') and SENSOR_DISTANCIA(4, 'MEDIA'))) then
NOTIFICAR(3)
DESPLAZAMIENTO('ADELANTE')
VELOCIDAD('LENTA')
TIEMPO(2)
NOTIFICAR(4)
DETENERSE()
SWITCH_ESTADO('INICIAL')
elseif(not(SENSOR_DISTANCIA(1, 'LARGA'))) then
NOTIFICAR(5)
PRENDER_LUCES(255,0,0)
TIEMPO(1)
NOTIFICAR(6)
APAGAR_LUCES()
SONIDO('ASUSTADO')
TIEMPO(3)
NOTIFICAR(7)
LUCES_POR_ARCOS(255,0,0,255,255,0,255,165,0,128,0,128)
end
```

Figura 4.24: Código generado por el programa de ejemplo

A partir de este ejemplo se puede observar que los identificadores de los bloques no son los identificadores reales asignados por Blockly, sino que corresponden a la posición del bloque a notificar dentro de un arreglo. La razón por la que no se utilizan los identificadores de Blockly es que los mismos contienen todo tipo de caracteres, entre ellos caracteres de comillas, lo cual puede afectar el código generado.

4.3.1.5. Conflictos de eventos y prioridades

Cuando el robot ejecuta un programa, es posible que en un determinado momento sucedan dos o más eventos al mismo tiempo. Supongamos que el robot sensa el color verde y al mismo tiempo detecta un objeto con el sensor 1 a distancia corta, en este caso si en el programa se definieron ambos eventos entonces surge un conflicto de eventos.

Como se mencionó anteriormente, por defecto Blockly genera el código recorriendo los bloques desde arriba hacia abajo. En este caso el orden en que se genera el código es muy importante ya que determina qué tan arriba dentro de la estructura de if/elseif va a quedar dicho código, esto determina un orden de prioridad, es decir que un evento que está más arriba en la estructura de if/elseif será evaluado antes que los que estén debajo y por lo tanto se podría considerar que es un evento prioritario respecto a los otros, ya que si se cumplen varios eventos al mismo tiempo se ejecutará el que se evalúe primero. Para el lenguaje actual se optó por cambiar el orden de recorrida de los bloques, yendo desde la izquierda hacia la derecha e ignorando la posición vertical de los bloques por completo. Esto parece más razonable ya que lo más natural es colocar los eventos lado a lado y las acciones hacia abajo, de esta forma se aprovecha mejor el espacio en pantalla y es más fácil visualizar varios eventos y sus correspondientes acciones, tal como se aprecia en la figura 4.23. En los Anexos de este documento se explica en detalle cómo se realizó la modificación para que Blockly genere el código de izquierda a derecha.

Por lo tanto una forma de resolver un conflicto entre eventos es colocar el evento prioritario a la izquierda del evento menos prioritario. Ésta y otra forma de resolver conflictos mediante el uso de bloques de lógica, son explicadas en detalle en el *Manual de Usuario* [32].

4.3.1.6. Eventos de Blockly

Los eventos *BLOCK_CREATE*, *BLOCK_DRAG* y *BLOCK_MOVE* se utilizan para capturar el momento en que se crea un nuevo bloque. Se logró identificar la combinación de eventos que suceden únicamente en estos escenarios y asociar al componente *workspace* (espacio de trabajo) de Blockly funciones para detectar y manejar estos eventos. Esto permite que se desplieguen de forma automática los menúes de configuración de los bloques

configurables, ahorrando el click del usuario y agilizando la programación, para ello dichos bloques deben ser añadidos a un arreglo denominado *camposDesplegables* que se define en el archivo *blocks.js*, en este archivo es donde se definen los bloques del lenguaje.

Por otra parte el evento *BLOCK_DRAG* también se utiliza para realizar las transformaciones sobre los conectores de los bloques en caso de que corresponda, tal como se explicó al introducir los bloques lógicos y el bloque de *Estado*.

Finalmente se utiliza el evento *FINISHED_LOADING* que permite detectar el momento en que se termina de cargar el área de trabajo y todos los bloques al cargar un programa guardado. Este evento se utiliza únicamente para iniciar la función que despliega los menúes de los bloques, esto se hace una vez que se haya terminado de cargar el espacio de trabajo, dado que sino, al cargar un programa se mostrarían los menúes de los bloques.

4.3.1.7. Funciones de Blockly invocadas

Blockly cuenta con una API de funciones que son de utilidad al momento de crear una aplicación utilizando esta librería.
Entre las funciones que fueron utilizadas, algunas de las más relevantes son las siguientes:

- **workspaceToCode**: Permite generar el código del conjunto de bloques que estén en el área de trabajo.
- **workspaceToDom**: Traduce el espacio de trabajo y todos los bloques a XML, se utiliza para el guardado de programas.
- **domToWorkspace**: Restaura el espacio de trabajo y los bloques guardados a partir del XML.
- **addChangeListener**: Permite capturar eventos en el espacio de trabajo y ejecutar funciones cuando suceden esos eventos.
- **highlightBlock**: Permite resaltar bloques y también quitarles el resaltado.
- **setNextStatement** y **setPreviousStatement**: Permiten añadir o quitar los conectores de tipo *statement* de un bloque.
- **setOutput**: Permite añadir o quitar el conector de tipo *output* de un bloque.

4.3.2. Robotito

El objetivo de esta sección es describir la implementación realizada en el robot, así como las dificultades que surgieron durante dicha implementación.

En una primera instancia, se realizó una implementación que consitió en un receptor de comandos, denominado *recibe_ comandos.lua* y un intérprete del código generado por la aplicación móvil, denominado *interprete_ vpl.lua*. El receptor de comandos cuenta con un socket TCP que espera una conexión y responde ante ciertos comandos. Además, al receptor de comandos se le añadió la lógica para ejecutar dichos comandos, contando con una función de calibración del sensor de color, una función encargada de escribir el código recibido desde la aplicación móvil dentro de un archivo denominado *vplProgram.lua*, una función que ejecuta dicho código dentro de un bucle infinito y finalmente una función que permite persistir datos en la memoria no volátil del robot.

Dentro del intérprete del VPL se definieron todas las funciones del lenguaje y sus implementaciones, en la figura 4.24 se puede observar la mayoría de las funciones del lenguaje. También, dentro del intérprete, se definió la función *dump_ rgb*, que se utiliza dentro del módulo del sensor de color *color.lua* para determinar el color sensado a partir del promedio de las cuatro últimas lecturas realizadas. Finalmente, dentro del intérprete se define e inicializa un hilo que será el encargado de emitir sonidos mientras el hilo principal sigue ejecutando. Esto implica que la emisión de sonidos no es bloqueante, pues el robot puede seguir detectando eventos y/o ejecutando acciones mientras emite un sonido, lo cual permitiría definir un comportamiento variable asociado a una emoción.

Con esta implementación inicial, se logró ejecutar los primeros programas creados en la aplicación móvil. Sin embargo, en algunas ocasiones ocurren errores, algunos de estos errores ocasionan un reinicio del robot, otros errores hacen que el hilo de ejecución principal del robot deje de procesar, en este caso si el robot estaba en movimiento se mantendrá en movimiento y si estaba con las luces prendidas se mantendrá con las luces prendidas, sin embargo no detecta nuevos eventos ni ejecuta nuevas acciones.
En las figuras 4.25 y 4.26 se pueden observar los dos errores más habituales.

```
CORRUPT HEAP: multi_heap.c:523 detected at 0x3ffeecac
abort() was called at PC 0x40092d8f on core 0

ELF file SHA256: fcf91730dab2d11c4ed02f28ae48295137efd33649b6653072b7914445a8e4e
4

Backtrace: 0x4008a5e3:0x3ffbc5c0 0x4008a925:0x3ffbc5e0 0x40092d8f:0x3ffbc600 0x4
009327d:0x3ffbc620 0x400857f9:0x3ffbc650 0x40085868:0x3ffbc670 0x40085c49:0x3ffb
c690 0x40086d08:0x3ffbc6b0 0x4000bedd:0x3ffbc6d0 0x40128d07:0x3ffbc6f0 0x401308a
6:0x3ffbc710 0x4012ca39:0x3ffbc730 0x4012caec:0x3ffbc750 0x4012ce37:0x3ffbc770 0
x4013ad09:0x3ffbc7a0 0x4012d03d:0x3ffbc7f0 0x4012d05d:0x3ffbc810 0x40127749:0x3f
fbc830 0x4012ca09:0x3ffbc850 0x4012d1e6:0x3ffbc8d0 0x4012859d:0x3ffbc900 0x4013b
83f:0x3ffbc930 0x4009276f:0x3ffbc970

Rebooting...
ets Jun  8 2016 00:22:57
```

Figura 4.25: Error CORRUPT HEAP

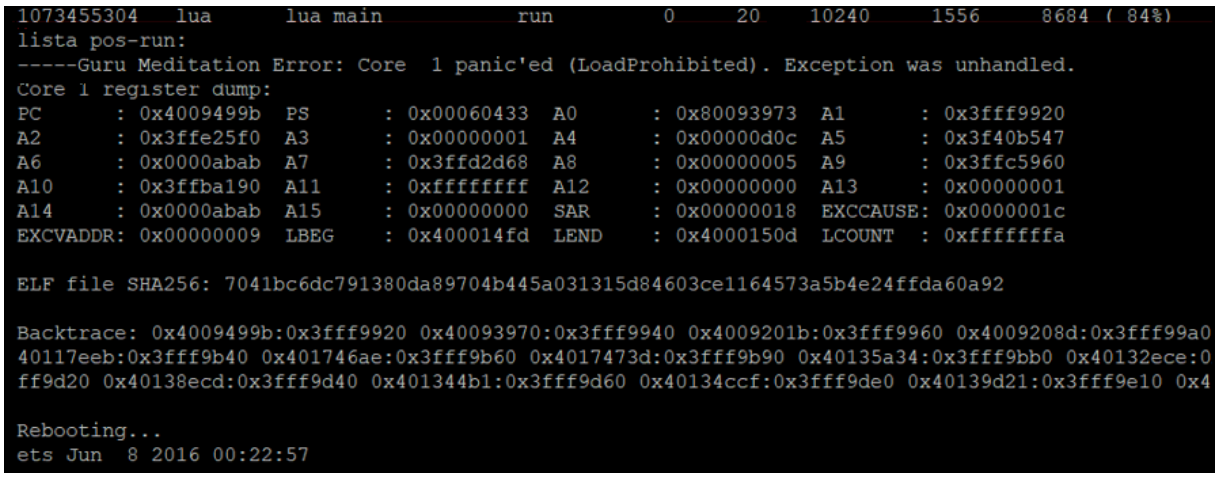


Figura 4.26: Error Guru Meditation

Como se puede observar, ambos errores ocasionan un reinicio del robot. Por otra parte, en los casos en que dejó de funcionar el hilo de ejecución, no se obtiene ningún mensaje de error en la consola. Se intentó sin éxito llegar a un escenario puntual en el que se pueda replicar alguno de estos errores de forma inequívoca, los mismos no siempre suceden y cuando lo hacen no siempre es en un mismo punto de la ejecución.
Si bien estos errores pueden interferir en las actividades llevadas a cabo por los usuarios, los mismos no son tan habituales, con lo cual igualmente se podría llevar a cabo dichas actividades.

Luego de esta primera implementación se decidió explorar una implementación alternativa, utilizando más hilos de ejecución, lo cual permitiría tener un hilo receptor de comandos, otro hilo ejecutor del bucle infinito en donde se ejecuta el programa *vplProgram.lua* y otro encargado de emitir sonidos. Si bien éste no era un requisito del proyecto, igualmente se decidió explorar esta posibilidad ya que permitiría que un usuario envíe comandos al robot mientras el mismo está ejecutando un programa, esto habilita por ejemplo la posibilidad de pausar una ejecución, modificar el programa actual y reenviar el nuevo código para que el robot siga ejecutando el programa modificado, agilizando el proceso de creación y pruebas de nuevos programas.
Para esta implementación se definieron varios módulos lua [17]:

- **cmd_receive.lua**: Contiene el socket TCP que recibe los comandos enviados por la aplicación móvil.
- **cmd_exec.lua**: Exporta funciones para iniciar y pausar una ejecución, calibrar un color y escribir el código recibido desde la aplicación móvil dentro del archivo *vplProgram.lua*.
- **vpl_exec_mgr.lua**: Inicializa y administra el hilo que ejecuta el bucle infinito de

ejecución del programa *vplProgram.lua*, exporta funciones para iniciar y pausar la ejecución.

- **vpl_exec_values.lua**: Contiene algunos valores de la ejecución actual del robot, como ser la velocidad actual, color actual de las luces, dirección actual de movimiento, etc.
- **vpl_interface.lua**: Define las funciones del lenguaje.
- **vpl_implementation.lua**: Implementa las funciones del lenguaje, hace uso de los módulos *omni.lua*, *led_ring.lua*, *laser_ring.lua*, *sound_module.lua*, *color.lua*, que forman parte de la librería mencionada en la sección 3.1, que permite controlar los sensores y actuadores del robot [46].
- **sound_module.lua**: Inicializa el hilo emisor de sonidos y exporta la función *do_sound* que permite emitir un sonido.
- **udp_utils.lua**: Exporta la función *send_message* que permite enviar un mensaje mediante un socket UDP.

Al realizar pruebas en esta versión de la implementación surgió una gran cantidad de errores, los cuales imposibilitaron la ejecución de programas en el robot.
Algunos de estos errores sucedieron en torno al manejo de hilos mediante las funciones del módulo de hilos de LUA RTOS [47], ya sea al iniciar un hilo mediante la función *thread.start* o al detener un hilo mediante la función *thread.stop*. En varias ocasiones, el detener un hilo ocasionó que el hilo principal de ejecución deje de procesar, sin mostrar ningún error en consola.
Por otra parte, uno de los errores más frecuentes sucedía al recibir comandos desde la aplicación móvil en el socket TCP, el mensaje recibido era guardado en una variable y dicha variable se imprimía en la consola, esto permitió observar que a menudo el valor de la variable contenía una referencia a una función o a un hilo de ejecución, en lugar de contener un String con el comando enviado desde la aplicación. En estos casos se hizo debugging en la aplicación móvil para tener la certeza de que el comando enviado fuese correcto. El hecho de que la variable que guarda el comando en algunos casos contenga una referencia a una función o a un hilo de ejecución podría indicar una falla del sistema operativo en el manejo de punteros o memoria, sin embargo no se tiene certeza de que ésta sea la razón del error.
Éstos y otros errores hicieron inviable esta versión de la implementación. Se decidió, en conjunto con los tutores del proyecto, optar por la implementación inicial, poniendo énfasis en la implementación de la aplicación móvil y que la misma esté desacoplada de la implementación del robot. A continuación se profundiza sobre algunos de los aspectos más relevantes de la implementación inicial (y actual) en Robotito.

4.3.2.1. Modificación del módulo de sensor de color

El módulo *color.lua* ofrece funcionalidades que facilitan la configuración y el uso del sensor de color. Al inicializar este módulo, el mismo define una tabla con los colores que interesa detectar, definiendo para cada color una etiqueta que lo representa y los valores correspondientes a dicho color en el modelo de color HSV [4].

Originalmente, los colores definidos en esta tabla eran los que se utilizaban para el comportamiento basado en el sensor de color, de la aplicación de configuración de Robotito. Por otra parte, la tabla de colores se inicializaba por única vez al cargar el módulo del sensor de color.
Se decidió modificar este módulo añadiendo una función *init_color_table*. Esta función permite volver a inicializar la tabla de colores, considerando únicamente los colores utilizados en los bloques de *Detección de color* del último programa enviado desde la aplicación móvil al robot. Dichos colores los obtiene del almacenamiento no volátil, lo cual es posible ya que como se explicó anteriormente, cuando se envía un programa desde la aplicación móvil al robot, también se envían los colores utilizados en dicho programa dentro de bloques de *Detección de color*. La adición de esta función permite obtener los últimos valores sensados durante la calibración del sensor de color, para cada uno de los colores, a pesar de que el módulo del sensor de color ya hubiera sido inicializado antes de realizar dichas calibraciones.
Por otra parte, inicializar el sensor de color considerando únicamente los colores relevantes para el programa a ejecutar, ayuda a prevenir posibles errores en la detección de color. Supongamos que en un programa determinado se utiliza un bloque de *Detección de color* azul, supongamos también que la última vez que se utilizó el robot, el usuario que lo utilizó calibró el color celeste utilizando una tarjeta de color azul o similar, en este caso podría suceder que cuando el robot pasa por encima del color azul, el sensor de color detecte color celeste ya que sus valores HSV son muy similares al color azul que se está utilizando actualmente. Este tipo de errores se evitan inicializando la tabla de colores del sensor de color únicamente con los colores que son relevantes para el programa a ejecutar, en este ejemplo sería solamente el color azul.

4.3.2.2. Modalidades de ejecución

Se definen dos modalidades de funcionamiento del robot. La modalidad Manual requiere conexión con el dispositivo móvil, la misma sirve para enviarle al robot un nuevo programa y que éste lo ejecute y también sirve para calibrar el sensor de color. La modalidad Automática sirve únicamente para que el robot vuelva a ejecutar el último programa ejecutado.

Al prender el robot, mediante el interruptor de encendido, el robot sensa durante un segundo utilizando el sensor de proximidad inferior para determinar en qué modalidad ejecutar. Si no detecta nada en las proximidades del sensor, entonces inicia en la modalidad Manual, de lo contrario inicia en la modalidad Automática.
El robot indica que inició en la modalidad Manual al prender las luces de color blanco, esto significa que está a la espera de recibir un comando desde la aplicación móvil. Por otra parte, el inicio de la ejecución se indica prendiendo las luces de color verde durante un segundo, ya sea que dicha ejecución se inicia desde la aplicación móvil o que dicha ejecución se inicia de forma automática porque el robot está funcionando en la modalidad Automática.

4.3.2.3. Hilo de sonidos

El hilo que emite los sonidos implementa un bucle infinito que emite un sonido durante dos segundos y luego se suspende a sí mismo mediante las funciones *thread.suspend()* y *thread.self()* del módulo de hilos de LUA RTOS. La razón por la cual se realiza esta implementación es que la función *thread.stop()* ocasiona los problemas que se mencionaron anteriormente en esta sección. Al suspender un hilo, el mismo simplemente queda en pausa, el sistema operativo no libera la memoria del hilo. El problema con esta implementación en donde se pausa el hilo de sonidos en lugar de eliminarlo, es que si el robot se encuentra emitiendo un sonido no se puede detener este sonido y comenzar a emitir un nuevo sonido inmediatamente. Ésto solamente sería posible si se pudiera eliminar el hilo e inicializar uno nuevo que ejecute el nuevo sonido.
Como consecuencia, en la implementación actual, el robot ignora toda acción de Sonido mientras se encuentre emitiendo algún sonido anterior.

4.3.2.4. Valores de ejecución

En el intérprete *interprete_vpl.lua*, se utilizan variables para almacenar algunos valores de ejecución que indican el movimiento actual, velocidad actual, color sensado, luces prendidas y otros valores de la ejecución. Los mismos se utilizan, entre otras cosas, para evitar ejecutar repetidas veces una misma acción de forma innecesaria, lo cual sucede cada vez que el robot detecta un mismo evento varias veces, ejecutando las acciones asociadas a dicho evento una y otra vez. A modo de ejemplo, si el robot tiene las luces prendidas de color verde y ejecuta la acción de *Prender luces* de color verde, entonces el robot no llevará a cabo dicha acción, pues es innecesario.
La razón por la cual se utilizan estas variables, es que se detectaron algunos efectos adversos al ejecutar varias veces con los mismos valores las funciones de prender luces y de movimientos, definidas en los módulos *led_ring.lua* y *omni.lua*. A modo de ejemplo, al

ejecutar las funciones de encendido de luces múltiples veces, se puede observar que algunos de los leds, de forma aleatoria, toman otros colores.

4.3.2.5. Protocolo de comunicación

El protocolo de comunicación con el robot mediante el socket TCP consiste en cinco comandos, algunos de ellos se envían con parámetros que se separan con el caracter "*". A continuación se listan y describen los comandos:

- **CMD_WRITE_NVS**: Permite persistir algún valor en el almacenamiento no volátil del robot. Se envía con tres parámetros: espacio de nombres, clave y valor.
  Por ejemplo, al enviar "CMD_WRITE_NVS*color*vpl_colors*ROJO|AZUL", el robot persisitrá el valor "ROJO|AZUL" dentro del espacio de nombres "color", con clave "vpl_colors".

- **CMD_GET_CODE**: Comando que le indica al robot que a continuación va a recibir el código del programa.

- **CMD_END_CODE**: Comando que le indica al robot que ya se finalizó el envío del código del programa, al recibir este comando el robot finaliza la escritura del archivo *vplProgram.lua* que contiene el código recibido.

- **CMD_CALIBRATE_COLOR**: Comando que le indica al robot que debe realizar la calibración de color, este comando se envía con un parámetro que contiene el nombre del color a calibrar.
  Por ejemplo, si se envía el comando "CMD_CALIBRATE_COLOR*AZUL", entonces el robot inicia la calibración del sensor de color y obtiene los valores HSV sensados, los cuales serán persistidos en el almacenamiento no volátil, asociados al color "AZUL".

- **CMD_RUN_PROGRAM**: Le indica al robot que debe comenzar la ejecución del bucle infinito en donde se ejecuta el programa *vplProgram.lua*.

A continuación, en la figura 4.27 se puede observar cómo la aplicación móvil le envía al robot el código del programa creado con bloques.

```
TCPClient clienteComandos = new TCPClient();
clienteComandos.startConnection();
clienteComandos.sendMessage( msg: "CMD_GET_CODE");
for (String linea : codigo) {
    clienteComandos.sendMessage(linea);
}
clienteComandos.sendMessage( msg: "CMD_END_CODE");
resultado = clienteComandos.getMessage();
clienteComandos.stopConnection();
```

Figura 4.27: Envío del código mediante socket TCP

Esta constituye la parte más compleja del protocolo ya que implica el envío de varios mensajes por el socket TCP, involucrando más de un comando.

Si el robot ejecuta correctamente un comando entonces responde "CMD_EXECUTED".
En caso de que el dispositivo móvil no se encuentre conectado al robot, se informa al usuario que debe conectarse a la red WiFi del robot.
En caso de que el robot reciba correctamente un comando pero al cabo de tres segundos no haya respondido "CMD_EXECUTED" ocurrirá un timeout en el socket, en este caso se asume que ocurrió algún error en el robot y se informa al usuario que lo reinicie y vuelva a intentarlo.
En caso de que haya otro usuario que le haya enviado comandos al robot, el mismo responderá con mensaje "ERR_MULTIPLE_CLIENTS", en este caso la aplicación le informa al usuario que el robot está siendo utilizado por otro usuario. Esto último puede resultar de utilidad en algún contexto en donde haya varios usuarios trabajando con distintos robots.

Por otra parte, el socket UDP del robot envía mensajes a la dirección IP del dispositivo móvil que inició la ejecución. Como se mencionó anteriormente estos mensajes sirven para indicarle al dispositivo móvil, durante la ejecución de algún programa, qué bloques debe resaltar y cuál es el estado actual del robot (en referencia al bloque de *Estado*).
El mensaje "ESTADO*estado" le indica al dispositivo móvil que el estado actual es "estado", por ejemplo si recibe "ESTADO*MORSA" entonces actualizará la imagen del estado actual para mostrar el estado MORSA.
El mensaje "IDBLOQUE*id" le indica al dispositivo móvil que debe resaltar el bloque cuyo identificador es "id".

Capítulo 5

Pruebas de usabilidad

En este capítulo se describen las pruebas de usabilidad llevadas a cabo, los resultados de las mismas, ajustes realizados a partir de la retroalimentación obtenida y posibles mejoras a futuro.

5.1. Introducción

Se puede definir la usabilidad como la medida en la cual un producto puede ser utilizado por usuarios específicos para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso dado.
La usabilidad se puede evaluar en términos de los siguientes elementos:

- **Utilidad**: Grado en el que un producto habilita a un usuario a alcanzar sus objetivos.
- **Eficiencia**: Refiere a la velocidad con la cual el objetivo del usuario puede ser alcanzado.
- **Efectividad**: Refiere a la medida en la que el producto se comporta de la forma que los usuarios esperan y la facilidad con la cual los mismos pueden utilizarlo para hacer lo que pretenden.
- **Satisfactoriedad**: Refiere a la percepción, sentimientos y opiniones del usuario acerca del producto.
- **Aprendibilidad**: Es parte de la efectividad, refiere a la habilidad del usuario para utilizar el producto con cierto nivel de competencia luego de algún período de entrenamiento.
- **Accessibilidad**: Refiere al grado en que los usuarios tienen la posibilidad de utilizar el producto para cumplir sus objetivos.

El objetivo al realizar pruebas de usabilidad es detectar deficiencias o aspectos a mejorar del producto. Esto permite crear productos utiles y valiosos para los usuarios, que son sencillos de aprender y le permiten al usuario realizar las actividades deseadas de forma efectiva, eficiente y satisfactoria.

Lo recomendable es realizar pruebas de usabilidad con cinco usuarios cada vez. En la primera ronda de pruebas con cinco usuarios, se puede detectar alrededor del 85 % de los problemas de usabilidad [20].

5.2. Protocolo

Se realizaron pruebas con cinco usuarios representativos, dos de ellos son los investigadores que fueron entrevistados a inicios del proyecto, los tres usuarios restantes son maestras.
Las pruebas fueron individuales, se realizó la grabación de la pantalla del dispositivo móvil así como el audio de la sesión, además se tomaron notas en una libreta a medida que los usuarios hacían sugerencias o se detectaba alguna inconsistencia. Las evaluaciones tuvieron una duración aproximada de una hora. Las grabaciones fueron reproducidas a posteriori para recabar la mayor cantidad posible de información relevante.

A continuación se detalla el protocolo seguido durante las pruebas:

1. **Presentación del robot**: Se le muestra el robot al usuario, el mismo incorpora stickers que enumeran los sensores de distancia e indican cuál es el frente del robot, tal como fue indicado en la figura 4.3.

2. **Indicaciones para la prueba**: Se le explica al usuario en qué consistirá la prueba, se le solicita que piense en voz alta durante la evaluación transmitiendo lo que hace y lo que piensa en su interacción con el dispositivo móvil y el robot.

3. **Introducción a las funcionalidades básicas de la aplicación y el robot**: Se guía al usuario para que interactúe con el robot y el dispositivo móvil, permitiéndole crear y ejecutar un programa simple, que consiste en dos eventos (*Detección de color* y *Detección de objeto*) y cuatro acciones en total (*Ir hacia*, *Prender luces*, *Detenerse*, *Sonido enojado*). Además se le enseña cómo realizar la calibración del sensor de color previo a la ejecución. También se le explica al usario las dos modalidades de funcionamiento del robot y finalmente se le pide que guarde el programa y luego lo elimine.

4. **Problema a resolver**: Se le plantea al usuario un comportamiento a programar, para lo cual deberá interactuar con la mayoría de los bloques de nivel 1.

5. **Problema planteado por el usuario**: Se invita al usuario a plantear un escenario que le gustaría probar en la práctica, con los niños en el centro educativo. Si al usuario se le ocurre algún escenario se le pide que intente resolverlo.

6. **Presentación de los bloques faltantes**: Se le presentan y se le explican al usuario los bloques lógicos y el bloque de Estado.

7. **Sugerencias**: Se le pregunta al usuario si tiene alguna sugerencia o propuesta de modificación, para ello se le muestran nuevamente todos los menúes de configuración de los bloques, las distintas pantallas, los diseños de los bloques. Se aprovecha para validar el ícono del bloque de Sonido asustado, que no estaba validado y también se valida la elección de colores realizada tanto para el bloque de *Detección de color* como para los bloques de luces. Finalmente se le consulta por la interacción con el robot en general.

8. **Encuesta**: Se le presenta al usuario una encuesta, que permitirá evaluar distintos aspectos de usabilidad.

Este protocolo apunta a que el usuario interactúe lo máximo posible con la aplicación y el robot, logrando que ejecute los distintos casos de uso del sistema y que interactúe con la gran mayoría de bloques del lenguaje. De esta forma se pueden detectar fallas en el diseño o posibles mejoras a realizar, además se pueden evaluar aspectos de la usabilidad del sistema.

El problema a resolver que se le plantea al usuario, requiere el uso de los bloques: *Detección de color*, *Detección de objeto*, *Ir hacia*, *Velocidad*, *Girar*, *Luces por arcos*, *Apagar luces* y *Sonido asustado*.

5.3. Evaluaciones

Uno de los usuarios que participó de la evaluación fue una maestra que enseña a niños de primer año escolar. Este usuario nunca había interactuado con ningún robot, tampoco cuenta con conocimientos de programación.

Para evaluar el problema a resolver, se dispusieron tarjetas de color verde, amarillo y celeste en el suelo. El objetivo planteado a los usuarios fue que el robot hiciera un recorrido por dichos colores y que luego evite el choque contra la pared, ya que el último movimiento del recorrido es “Ir hacia adelante”, que sucede cuando el robot pasa por encima de la tarjeta de color celeste.

En la figura 5.1, se puede observar la solución planteada por este usuario al problema, que es el mismo problema que se le planteó a todos los usuarios evaluados.

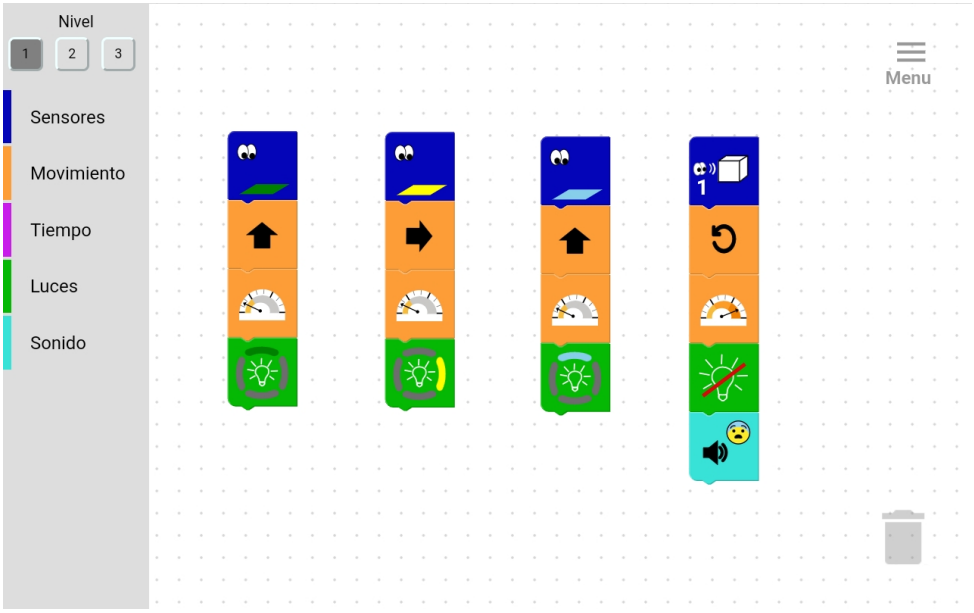


Figura 5.1: Solución de un usuario al problema planteado en las pruebas

Tal vez, un aspecto negativo de este problema que se le planteó a los usuarios, es que se presenta un escenario de ejecución predefinido, es decir que las tarjetas se encuentran dispuestas en el suelo y el usuario sabe de antemano cuál es el orden en que sucederán los eventos, lo cual en un paradigma de programación dirigida por eventos no tiene porqué ser así.

Durante la programación del escenario, al definir el evento correspondiente al bloque de *Detección de color* amarillo (el segundo evento que definió el usuario), el usuario intentó conectar dicho bloque a continuación del bloque de *Luces por arcos* con arco frontal verde, como si fuera una secuencia de acciones y eventos que suceden. Gracias a que el diseño de los bloques no permite dicha conexión, el usuario dudó pero finalmente colocó el bloque de evento separado de los otros bloques, como corresponde.

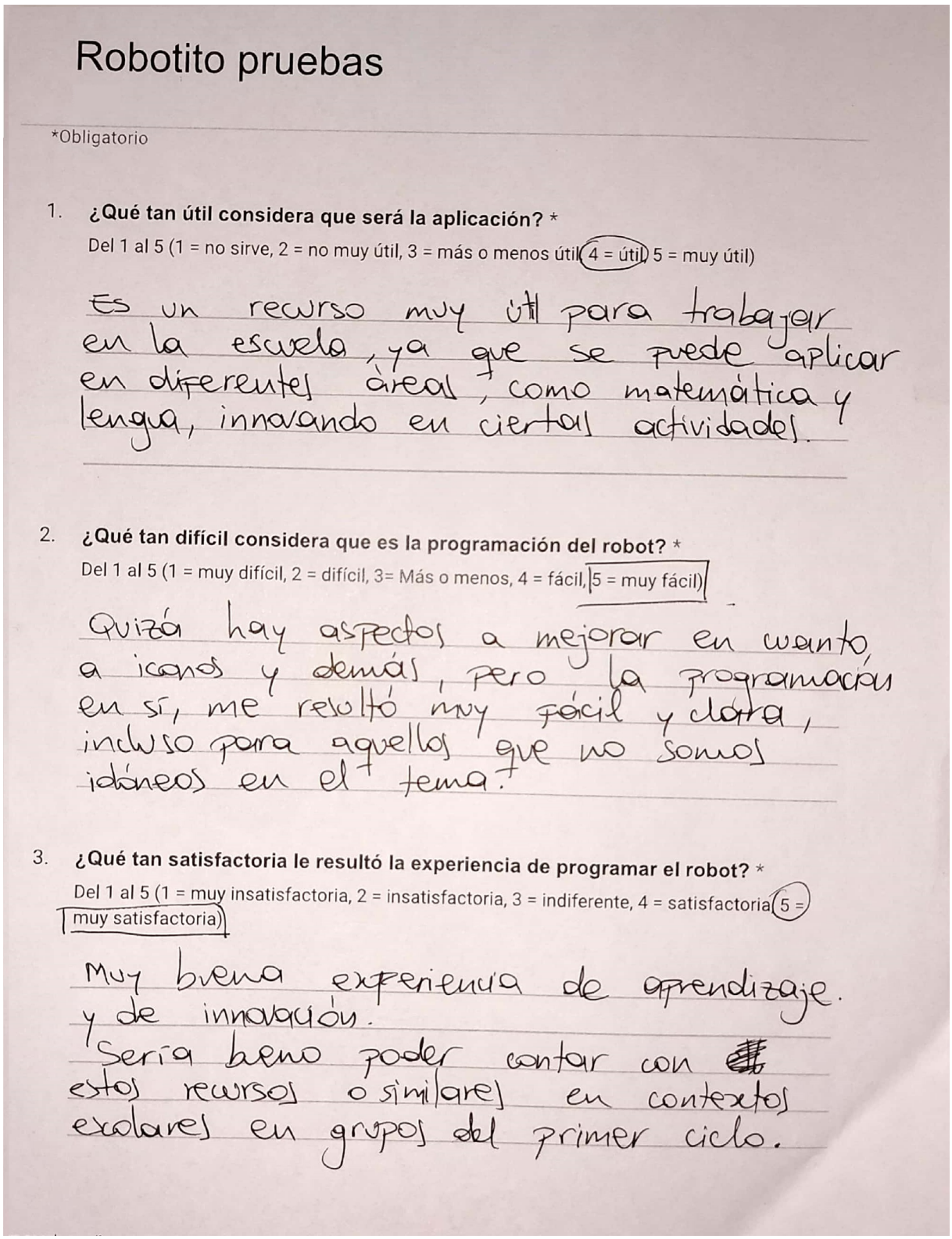
Si bien el usuario logró programar correctamente el comportamiento adecuado en el primer intento, se observa que utilizó tres veces el bloque de *Velocidad* lenta, cuando en este escenario en particular no era necesario definir una velocidad lenta para los eventos de *Detección de color* amarillo y *Detección de color* celeste, dado que el robot ya venía con una velocidad lenta que se le había asignado en el evento de *Detección de color* verde. Al explicarle al usuario, luego de la resolución del problema, que el robot mantiene las acciones de los eventos anteriores, el usuario no planteó objeciones y comprendió la redundancia de las dos acciones de *Velocidad* lenta definidas innecesariamente.

Luego de la resolución de este problema, el usuario hizo algunos comentarios y suge-

rencias.
En primer lugar sugiere modificar el ícono del bloque de *Detección de objeto*, sustituyendo las ondas que representan la distancia al objeto por flechas horizontales de distintos largos. Esto se debe a que inicialmente le costó distinguir a qué distancia se encuentra el objeto, específicamente cuando el bloque se encuentra dispuesto en el área de trabajo, ya que dentro de la pantalla de configuración del bloque, le resulta más sencillo identificar las diferencias. Más allá de esta dificultad, el usuario logró identificar la distancia al presionar sobre el bloque, ya que dentro de la pantalla de configuración además de verse mejor el ícono, se puede leer el significado del bloque.
Luego, el usuario tiene dificultades para distinguir dentro del bloque de *Luces por arcos*, el color verde que le asignó a uno de los arcos, sugiere utilizar un tono de verde que tenga mayor contraste con el fondo verde de los bloques de luces. Además, dentro de la pantalla de configuración de este bloque, el usuario tiene dificultades para seleccionar el color, con lo cual sugiere hacer los cuadrados de colores de dicha pantalla más grandes.
Finalmente, durante la observación de la ejecución del programa por parte del robot, el usuario se sorprendió por la lentitud del giro del robot a velocidad rápida, con lo cual sugiere ajustar la velocidad de los giros para que gire más rápido en general.

Continuando con la evaluación, el problema planteado por el usuario para trabajar con los niños en clase, consistió en un recorrido en donde se guía al robot utilizando tarjetas de colores, luego en cada tarjeta se escribe una letra, que puede ser vocal o consonante, finalmente el robot recorre una ruta que pasa por dichas tarjetas y en aquellas que sean vocales prende luces y emite un sonido feliz, pero en aquellas que sean consonantes apaga las luces y no emite sonidos. El usuario no tuvo problemas para resolver rápidamente el escenario propuesto, el mismo es de índole similar al ejercicio que había resuelto anteriormente, aunque utilizando mayor cantidad de eventos y acciones.

Luego de la resolución de este escenario, se le mostraron al usuario el resto de bloques de nivel 2 y 3 y las posibilidades que ofrecen dichos bloques. El usuario no aportó más sugerencias o comentarios aparte de los que ya había aportado hasta el momento.
Finalmente se le presentó al usuario una breve encuesta, cuyas respuestas pueden observarse en la figura 5.2.



Robotito pruebas

*Obligatorio

1. **¿Qué tan útil considera que será la aplicación?** *

Del 1 al 5 (1 = no sirve, 2 = no muy útil, 3 = más o menos útil, 4 = útil, 5 = muy útil)

Es un recurso muy útil para trabajar en la escuela, ya que se puede aplicar en diferentes áreas, como matemática y lengua, innovando en ciertas actividades.

2. **¿Qué tan difícil considera que es la programación del robot?** *

Del 1 al 5 (1 = muy difícil, 2 = difícil, 3= Más o menos, 4 = fácil, 5 = muy fácil)

Quizá hay aspectos a mejorar en cuanto a iconos y demás, pero la programación en sí, me resultó muy fácil y clara, incluso para aquellos que no somos idóneos en el tema.

3. **¿Qué tan satisfactoria le resultó la experiencia de programar el robot?** *

Del 1 al 5 (1 = muy insatisfactoria, 2 = insatisfactoria, 3 = indiferente, 4 = satisfactoria, 5 = muy satisfactoria)

Muy buena experiencia de aprendizaje y de innovación.
Sería bueno poder contar con estos recursos o similares en contextos escolares en grupos del primer ciclo.

Figura 5.2: Respuestas del usuario a la encuesta realizada

Otro usuario que realizó la prueba fue un investigador, que ha realizado actividades y evaluaciones con Robotito en centros educativos. Este investigador es ingeniero y fue uno de los usuarios que en etapas iniciales del proyecto comentaron que les sería útil contar

con la posibilidad de definir comportamientos distintos ante un mismo evento.
Durante la resolución del problema a resolver, el usuario cometió un error ya que definió un desplazamiento hacia la derecha al detectar color verde y un desplazamiento hacia adelante al detectar color amarillo. Al observar el fallo, el usuario corrigió rápidamente el programa y alegó que fue un error de distracción.
El escenario que plantea el usuario para trabajar con los niños es un escenario en donde el comportamiento del robot cambia dependiendo de los eventos anteriores. Puntualmente, mediante el uso de el bloque de *Estado* define un comportamiento en el cual cuando el robot detecta color amarillo ejecuta una actuación si anteriormente había detectado color verde y otra actuación distinta si anteriormente había detectado color azul. El usuario logró por sí solo resolver el escenario de forma correcta en un primer intento. Algo a considerar es que este usuario ya conocía de antemano el concepto de máquinas de estado, esto permite evaluar únicamente la interfaz y el diseño del lenguaje, dejando fuera la barrera conceptual de entender qué son y para qué sirven los estados.
El usuario identifica el bloque de *Sonido asustado* como un "sonido de sorpresa", pero dice que también podría representar un sonido asustado. Al consultarle por distintos aspectos del lenguaje, la aplicación y el robot, el usuario opina que todo está bien así. Por otra parte, este usuario es daltónico y planteó que le resulta muy valioso que dentro de los menúes de configuración de los bloques se nombre textualmente los colores que se están configurando, este es un aspecto que mejora la accesibilidad del sistema. Finalmente el usuario destaca como aspecto positivo el contar con un diseño gráfico de los bloques y el uso de colores para distinguir las distintas categorías de bloques, ya que considera que sería posible que los niños pequeños que aún no sepan leer puedan programar comportamientos sencillos.
En la encuesta el usuarió puntuó con un cinco la utilidad del sistema, con un cuatro la facilidad de programación y con un cinco la satisfactoriedad de la experiencia. Al consultarle cómo se podría hacer que la programación sea más fácil, el usuario recomienda incorporar un corrector semántico del lenguaje, así como contenido audiovisual que le enseñe al usuario a programar. La razón por la cual se sugiere el corrector semántico es porque en un momento dado de la evaluación, el usuario decidió probar a generar una secuencia de movimientos a partir de un único evento y al hacerlo utilizó únicamente los bloques de *Ir hacia* y no utilizó bloques de *Tiempo* para asignarle un tiempo a cada movimiento, esto constituye un error semántico en el programa, el mismo es explicado en mayor detalle y con ejemplos en la sección 4 del documento *Manual de usuario* [32].

Otro de los usuarios con los que se realizó pruebas fue una investigadora que también había planteado en etapas iniciales del proyecto que le sería útil contar con la posibilidad de definir comportamientos distintos ante un mismo evento, en base a lo que hubiera

sucedido anteriormente.
El problema a resolver fue implementado de forma correcta en el primer intento, aunque también se utilizan bloques de *Velocidad* lenta de forma redundante. Luego el usuario experimenta con los bloques de lógica y ejecuta un evento que utiliza el bloque lógico *Y/O*. El usuario sugiere añadir alguna opción que permita copiar un bloque determinado y también bajar la velocidad de movimientos rápida y media. Por otra parte sugiere mejorar el resaltado de bloques, ya que para los bloques de movimiento es muy leve la diferencia de un bloque normal y uno resaltado, también propone añadir un indicador sonoro (además de las luces blancas) cuando el robot inicia en la modalidad Manual. Finalmente menciona que los sensores de distancia 2, 3, 5 y 6 no sabe cómo utilizarlos ya que no coinciden con ninguna dirección de desplazamiento, en este aspecto el usuario opina que sería útil contar con un bloque de movimientos en dirección de los sensores para darle más utilidad a los mismos.
En la propuesta de escenario para trabajar con los niños, el usuario propone un escenario de eventos condicionados utilizando estados. Al intentar programar dicho escenario el usuario tiene varias dificultades, este usuario no está familiarizado con el concepto de máquinas de estados. Uno de los problemas que se pueden observar es que intenta utilizar estados en todos los eventos que define, a pesar de que no es necesario. Por otra parte, en un momento determinado, el usuario intenta conectar dos bloques de Estado para indicar que el robot debe ejecutar primero un estado y luego el otro, como si fueran acciones en secuencia, al no poder realizar dicha conexión debido a que los conectores del bloque no lo permiten el usuario no sabe como proceder.
En la encuesta el usuarió puntuó con un cinco la utilidad del sistema, con un cuatro la facilidad de programación y con un cinco la satisfactoriedad de la experiencia, sugiriendo añadir tutoriales en formato audiovisual para mejorar la facilidad de programación.
Queda claro que la programación de comportamientos con bloques de *Estado* requiere de ensayo y error, además de algún tutorial con explicaciones y ejemplos, especialmente para aquellos usuarios que no conozcan el concepto de máquina de estados.

Luego se realizaron pruebas con una maestra que trabaja con niños de cinco años.
El problema a resolver fue implementado correctamente en el primer intento, sin embargo se observa que intenta colocar los eventos en una única secuencia en vertical, como ya había sucedido con otro usuario, en este caso el también se dio cuenta por sí solo que no tiene forma de conectar el evento a continuación de una acción, con lo cual pudo corregir el error. Luego, el usuario decide experimentar con la definición de secuencias de movimientos y luces, para ello define una secuencia de acciones sin utilizar el bloque de *Tiempo*. Al explicarle que debe definir los tiempos de ejecución el usuario ajusta el programa y define una actuación con un bloque de *Ir hacia* adelante, un bloque de *Prender luces* rojas y un

bloque de *Tiempo* 2 segundos y plantea que el robot irá hacia adelante con las luces rojas durante dos segundos y luego frenará y apagará las luces. En la implementación actual de bloque de *Tiempo* esto es incorrecto, ya que para que el robot se detenga y apague las luces se deberían conectar los bloques correspondientes a dichas acciones a continuación del bloque de *Tiempo*. Al plantearle esto al usuario, el mismo expresa que le resulta más intuitiva su interpretación, en donde el bloque de *Tiempo* determina el tiempo durante el cual se ejecutarán las acciones anteriores y además implica que luego de ese tiempo las mismas cesarán. Esta es una interpretación válida, sin embargo podría dar pie a que un usuario utilice un bloque de sonido y luego el bloque de *Tiempo*, interpretando que el robot cortará el sonido luego de transcurrido un tiempo. En la sección 6.2 se profundiza sobre la viabilidad de esta sugerencia.
Por otro lado, al seguir con la evaluación, el usuario sugiere hacer más llamativo el resaltado de bloques ya que opina que no es tan notorio. También sugiere aumentar la velocidad de los giros en general y aumentar el tamaño de los cuadrados de colores en la pantalla de configuración del bloque de *Luces por arcos*.
El usuario no planteo ningún escenario para trabajar con los niños.
En la encuesta el usuarió puntuó con un cuatro la utilidad del sistema, con un cuatro la facilidad de programación y con un cinco la satisfactoriedad de la experiencia. En cuanto a la utilidad del sistema, este usuario opina que sería una herramienta útil para enseñarle conceptos de programación a niños de alrededor de siete años, sin embargo no tiene tan claro qué actividades puede programar una maestra para que los niños interactúen con el robot. En cuanto a la facilidad de programación opina que el lenguaje es intuitivo en general, sin embargo esto no fue así con el bloque de *Tiempo*, aunque luego de explicarle el funcionamiento el usuario lo utilizó correctamente.

Finalmente, se contó con la participación de una maestra que trabaja con niños de primero, segundo y tercero de enseñanza primaria.
El problema a resolver fue implementado de forma correcta en el primer intento, aunque también intentó conectar los eventos en secuencia y pudo corregir por sí misma, además utilizó redundantemente bloques de *Velocidad lenta*. Este usuario en particular se tomó más tiempo que el resto de usuarios en la evaluación en general, observando detenidamente cada uno de los bloques y con un gran nivel de concentración al programar, debido a esto no alcanzó el tiempo para plantear un escenario para trabajar con los niños en el centro educativo.
La única sugerencia del usuario luego de evaluar el producto fue la de utilizar letras mayúsculas en todos los textos, ya que los niños que recién aprenden a leer conocen únicamente las mayúsculas, esto pensando en la posibilidad de que los niños programen a futuro.

El usuario experimentó con los bloques lógicos y definió un programa con un evento lógico de forma exitosa, estos bloques le llamaron la atención y opina que podrían ser de utilidad para añadir nuevos comportamientos. También destaca el resaltado de bloques que le pareció muy útil.
En la encuesta el usuarió puntuó con un cinco en todas las preguntas.

5.3.1. Resultados

Como resultado de las pruebas se mejoraron algunos aspectos del sistema:

- Resaltado de bloques: Se resaltan más los bloques.
- Bloque de Luces por arcos: Se agrandan los bloques de colores en la pantalla de programación del bloque.
- Bloque de Luces por arcos: Se aumenta el contraste entre el arco de color verde y el fondo verde del bloque.
- Velocidad de giros: Se aumenta la velocidad de los giros en general.
- Velocidad de desplazamiento: Se disminuye la velocidad de desplazamiento en general.
- Letras mayúsculas: Se utilizan letras mayúsculas en todo el sistema.

También se obtiene retroalimentación valiosa, que puede considerarse para realizar ajustes a futuro, en caso de que hayan más usuarios que sugieran cambios, por ejemplo: modificar el funcionamiento del bloque de *Tiempo*, añadir un bloque de movimiento en dirección de los sensores, utilizar líneas horizontales en vez de ondas en el bloque de *Detección de objeto*, añadir una opción de copiar bloques, que el robot emita un breve sonido al iniciar en modalidad Manual, etc.

Otro aspecto positivo de las pruebas de usabilidad fue poder observar cuáles áreas de la programación generan dificultades o dudas a los usuarios. En el documento de *Manual de usuario* se cuenta con ejemplos y explicaciones para resolver las dudas más habituales de los usuarios, principalmente en la sección 4 de dicho documento [32].

En cuanto a la utilidad del sistema, todos los usuarios evaluados consideran que les será útil o muy útil contar con el producto, además los tres usuarios que se plantearon programar un escenario que les sería útil llevar a cabo en el centro educativo, lograron encontrar implementaciones viables de dichos escenarios.
Para evaluar la eficiencia del sistema de forma correcta, es importante contar con algún punto de comparación, por ejemplo teniendo varias implementaciones distintas de la

interfaz, lo cual permitiría medir la cantidad de tiempo que lleva resolver ciertos escenarios en las distintas interfaces. En este proyecto no se implementaron varias interfaces. Sin embargo, desde inicios del proyecto se consideraron distintos diseños del lenguaje y de cada uno de los bloques y sus menúes de configuración, varios de ellos fueron descartados o mejorados con el objetivo de minimizar la cantidad de clicks necesarios para configurar los bloques. Un ejemplo de esto es la decisión de utilizar bloques que cambian sus conectores con el objetivo de evitar el uso redundante del bloque "if". También, con el objetivo aumentar la eficiencia de la programación con bloques, desde etapas tempranas de diseño del sistema se exploraron formas de lograr que los bloques configurables desplieguen su menú de configuración de forma automática sin necesidad de hacer click, esto se logró llevar a cabo, permitiendo configurar bloques que ofrecen varias opciones de configuración con un solo click, como por ejemplo los bloques de *Ir hacia*, *Detección de color* y *Prender luces*.

La efectividad del sistema fue buena en general, cuatro de los cinco usuarios lograron resolver el problema que se les planteó en el primer intento. El usuario que falló la prueba pudo corregir inmediatamente por sí solo el problema y resolver exitosamente en el segundo intento. En cuanto al uso del bloque de *Tiempo*, no se cuenta con suficiente información, sin embargo el usuario que lo utilizó comentó que no se comporta de la forma que esperaba, con lo cual, a menos que se trate de un caso aislado, este podría ser un aspecto de la efectividad a mejorar, pudiendo aplicar en un futuro la modificación sugerida por el usuario en caso de que se considere necesario.

La satisfactoriedad del sistema es muy buena, todos los usuarios puntuaron con cinco en el cuestionario, mostraron entusiasmo en sus interacciones e hicieron comentarios muy positivos.

La aprendibilidad del sistema es buena, luego de una introducción básica al funcionamiento general del mismo los usuarios pudieron resolver un problema de mayor complejidad que el que se les mostró inicialmente como ejemplo, logrando definir varios eventos distintos y logrando utilizar exitosamente la mayoría de los bloques de nivel 1.

En cuanto a la accesibilidad del sistema, se considera que la misma es buena para el contexto de uso que se le dará al producto. Un usuario que es daltónico logró utilizar el producto correctamente e hizo comentarios positivos sobre el diseño de los menúes de configuración de los bloques que ofrecen una representación textual del valor del bloque. Varios de los usuarios comentaron que con esta interfaz los niños podrían programar gracias al uso de elementos gráficos y colores. Finalmente, aquellos usuarios que cuenten con dispositivos Android podrán utilizar el sistema, lo cual en un contexto de educación pública, en donde se cuenta con tablets del plan Ceibal debería ser posible. Además, se realizó un diseño responsive tanto de la pantalla de inicio como de la pantalla de programación, lo cual permite hacer uso de la aplicación en dispositivos con distintas resoluciones.

Finalmente, las pruebas de usabilidad permitieron validar no solamente la aplicación móvil, sino también la interacción del usuario con la aplicación y el robot en general. En particular, se logró validar la utilidad de los stickers del robot, que le permitieron a los usuarios orientarse de forma correcta. Por otra parte, se pudo validar la utilidad del uso de luces en el robot para indicar el estado en el que se encuentra el mismo, ya sea que está esperando un comando (prende luces blancas) o que está iniciando la ejecución (prende luces verdes durante un segundo).

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

Se realizó un proceso de diseño centrado en el usuario, en el que a través de entrevistas, evaluaciones de prototipos y pruebas de usabilidad, se logró desarrollar una aplicación Android que permite a usuarios no programadores definir el comportamiento de Robotito. Para definir el comportamiento del robot, los usuarios programan en un lenguaje de programación visual que es icónico y basado en bloques. A través de este proceso de diseño se logró generar un producto accesible y útil para los usuarios, que se puede aprender a utilizar fácilmente en líneas generales y que además ofrece una experiencia que resulta muy satisfactoria.

Se obtuvieron muy buenos resultados en las pruebas de usabilidad. Todos los usuarios se mostraron entusiasmados al interactuar con el producto, lo cual se ve reflejado en los resultados de las encuestas. La mayoría de los usuarios expresaron que el uso del sistema les resultó sencillo e intuitivo.

Además, se consideró desde inicios del proyecto la posibilidad de que en algún momento esta interfaz pueda ser utilizada por niños o adultos que no sepan leer, esto impactó en varias decisiones de diseño, logrando una interfaz con bloques gráficos, distintos colores y textos en mayúsculas.

Se logró implementar las funcionalidades propuestas a inicios del proyecto, incluso fue posible implementar el bloque de *Estado* que será de gran utilidad para algunos usuarios.

Se desarrolló un documento de *Manual de usuario* en donde se explican las distintas funcionalidades de la aplicación, el funcionamiento de los bloques y el lenguaje en general.

6.2. Trabajo futuro

En primer lugar es de gran importancia crear sonidos asociados a emociones, actualmente el robot emite una frecuencia aleatoria, que fue utilizada con fines de testing, con lo cual los sonidos no son muy útiles.

Luego, sería conveniente añadir un corrector semántico del lenguaje. Esto permitiría detectar aquellos casos en los que un usuario define una secuencia de acciones con más de un bloque de movimiento o de luces, sin utilizar el bloque de *Tiempo* para separar dichas acciones. También permitiría detectar casos en donde se define un mismo evento más de una vez. Sería positivo que al detectar un error semántico, se le presente al usuario con algún tipo de ayuda que le permita entender qué es lo que está haciendo mal.
También sería conveniente producir contenido audiovisual como tutoriales o ejemplos, sobre todo para atacar los conceptos que puedan generar una mayor confusión, como la programación dirigida por eventos, el uso del bloque de *Tiempo* y el bloque de *Estado*. Habría que evaluar si es conveniente embeber este contenido dentro de la aplicación o si es mejor generar contenido externo.

Por otra parte, sería conveniente investigar el origen de los errores que suceden de vez en cuando en el robot y lograr solucionarlos, ya que los mismos podrían afectar las actividades realizadas por los usuarios.

En cuanto al uso del bloque de *Tiempo* y su intuitividad, resta por evaluar con una mayor cantidad de usuarios si realmente es necesario realizar un cambio. El cambio sugerido por uno de los usuarios, en donde el bloque de *Tiempo* implica el cese de las acciones anteriores, se podría realizar haciendo modificaciones al generador de código del bloque de *Tiempo* y de los bloques de sonido, la duración de los sonidos podría cambiar a un segundo y finalmente modificar la función de "SONIDO(emocion)" dentro de la interfaz del lenguaje en el robot, para que reciba también como parámetro la duración del sonido.

El proceso de diseño centrado usuario nunca termina, es decir que siempre se puede seguir evaluando el uso del sistema y detectando aspectos a mejorar. Para ello sería conveniente evaluar el uso del sistema en el contexto de las actividades educativas realizadas por maestras e investigadores en los centros educativos.

Bibliografía

[1] Ewelina Bakala, Jorge Visca, Gonzalo Tejera, Andres Sere, Guillermo Amorin y Leonel Gómez Sena. "Designing child-robot interaction with Robotito". En: oct. de 2019. DOI: 10.1109/RO-MAN46459.2019.8956448.

[2] Dave Catlin y Dr. John Woollard. "Educational Robots and Computational Thinking". En: jul. de 2014.

[3] Ceibal. *Ceibal - ¡Llega el Scratch Day 2022!* https://www.ceibal.edu.uy/es/articulo/llega-el-scratch-day-2022. [Última consulta: 14/09/2022].

[4] Jun-Dong Chang, Shyr-Shen Yu, Hong-Hao Chen y Chwei-Shyong Tsai. "HSV-based Color Texture Image Classification using Wavelet Transform and Motif Patterns". En: *Journal of Computers* 20 (ene. de 2010).

[5] developer.android.com. *Activity | Android Developers*. https://developer.android.com/reference/android/app/Activity. [Última consulta: 15/8/2022].

[6] developer.android.com. *Cómo crear aplicaciones web en WebView | Desarrolladores de Android | Android Developers*. https://developer.android.com/guide/webapps/webview#BindingJavaScript. [Última consulta: 11/8/2022].

[7] Dexter Industries. *GoPiGo3 is a Raspberry Pi Robot Car for Learning Coding*. https://www.dexterindustries.com/gopigo3/. [Última consulta: 27/10/2020].

[8] Google. *blockly/demos/mobile/android at develop · google/blockly*. https://github.com/google/blockly/tree/develop/demos/mobile/android. [Última consulta: 13/8/2022].

[9] Google. *Class: Block | Blockly | Google Developers*. https://developers.google.com/blockly/reference/js/Blockly.Block#getMatchingConnection. [Última consulta: 10/8/2022].

[10] Google. *Creating a new field type | Blockly | Google Developers*. https://developers.google.com/blockly/guides/create-custom-blocks/fields/customizing-fields/creating. [Última consulta: 27/10/2020].

[11] Google. *GitHub - google/blockly: The web-based visual programming editor*. https://github.com/google/blockly. [Última consulta: 27/10/2020].

[12] Google. *google/blockly-android: Blockly for Android*. https://github.com/google/blockly-android. [Última consulta: 13/8/2022].

[13] Google. *WebView | Desarrolladores de Android | Android Developers*. https://developer.android.com/reference/android/webkit/WebView. [Última consulta: 27/10/2020].

[14] MIT Media Lab Lifelong Kindergarten Group. *GitHub - LLK/scratch-link: Device interoperability layer for Windows and MacOS*. https://github.com/LLK/scratch-link. [Última consulta: 27/10/2020].

[15] MIT Media Lab Lifelong Kindergarten Group. *Home · LLK/scratchx Wiki · GitHub*. https://github.com/LLK/scratchx/wiki. [Última consulta: 27/10/2020].

[16] MIT Media Lab Lifelong Kindergarten Group. *Scratch Jr - Home*. https://www.scratchjr.org/. [Última consulta: 27/10/2020].

[17] http://lua-users.org/. *lua-users wiki: Modules Tutorial*. http://lua-users.org/wiki/ModulesTutorial. [Última consulta: 10/8/2022].

[18] Shih-Miao Huang, Kong-King Shieh y Chai-Fen Chi. *Factors affecting the design of computer icons*. Nov. de 2001. URL: https://www.sciencedirect.com/science/article/abs/pii/S0169814101000646.

[19] Interaction Design Foundation. *What is User Centered Design? | Interaction Design Foundation (IxDF)*. https://www.interaction-design.org/literature/topics/user-centered-design. [Última consulta: 16/7/2022].

[20] Jakob Nielsen. *Why You Only Need to Test with 5 Users*. https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/. [Última consulta: 30/08/2022].

[21] Beate Jost, Markus Ketterl, Reinhard Budde y Thorsten Leimbach. "Graphical Programming Environments for Educational Robots: Open Roberta - Yet Another One?" En: dic. de 2014. DOI: 10.1109/ISM.2014.24.

[22] James D. Kiper, Elizabeth Howard y Chuck Ames. "Criteria for Evaluation of Visual Programming Languages". En: *Journal of Visual Languages & Computing* 8.2 (abr. de 1997), págs. 175-192. DOI: https://doi.org/10.1006/jvlc.1996.0034.

[23] Lucía Labat, Felipe Parodi, Gonzalo Tejera y Marcos Viera. *Home · Wiki · matefun / Infantil*. https://gitlab.fing.edu.uy/matefun/infantil/-/wikis/home. [Última consulta: 27/10/2020].

[24] Lego. *MINDSTORMS EV3 Support | Everything You Need | LEGO® Education*. https://education.lego.com/en-us/support/mindstorms-ev3. [Última consulta: 27/10/2020].

[25] lua.org. *Lua: about*. https://www.lua.org/about.html. [Última consulta: 02/07/2022].

[26] Makeblock. *mBot - Bluetooth - Robot Educativo 90054- Makeblock*. https://www.makeblock.es/productos/robot_educativo_mbot/. [Última consulta: 27/10/2020].

[27] MakeWonder. *Dash - Wonder Workshop - US*. https://www.makewonder.com/robots/dash/. [Última consulta: 27/10/2020].

[28] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman y Evelyn Eastmond. "The Scratch Programming Language and Environment". En: *ACM Trans. Comput. Educ.* 10.4 (nov. de 2010). DOI: 10.1145/1868358.1868363.

[29] Andrés Nebel y Renzo Rozza. "Yatay - IDE Android para la Programación de Comportamientos Robóticos". Tesis de grado. Facultad de Ingenieria, 2014.

[30] Don Norman. *The Design of Everyday Things*. Basic Books, Inc., 2002.

[31] Modular Robotics. *Modular Robotics | Cubelets robot blocks*. https://www.modrobotics.com/#whatr-cubelets. [Última consulta: 27/10/2020].

[32] Santiago Hitta. *Manual de usuario*. https://gitlab.fing.edu.uy/santiago.hitta/robotito_vpl/-/wikis/uploads/943d6cd74215d70d4496e03020f15bd5/Manual_de_usuario.pdf. [Última consulta: 13/09/2022].

[33] Santiago Hitta. *Robotito VPL - Estado del Arte*. https://gitlab.fing.edu.uy/santiago.hitta/robotito_vpl/-/wikis/uploads/fc204ff7547a603d8987cf5718fc563f/Estado_del_arte_1.1.pdf. [Última consulta: 13/09/2022].

[34] Jiwon Shin, R. Siegwart y S. Magnenat. "Visual Programming Language for Thymio II Robot". En: 2014. DOI: https://doi.org/10.3929/ethz-a-010144554.

[35] Sphero, Inc. *Sphero Education | STEM & STEAM EdTech Company: Sphero*. https://sphero.com/pages/educators. [Última consulta: 27/10/2020].

[36] Gonzalo Tejera, Guillermo Amorin, Andrés Sere, Nicolás Capricho, Pablo Margenat y Jorge Visca. "Robotito: programming robots from preschool to undergraduate school level". En: *2019 19th International Conference on Advanced Robotics (ICAR)*. 2019, págs. 296-301. DOI: 10.1109/ICAR46387.2019.8981608.

[37] Terrapin. *Bee-Bot*. https://www.terrapinlogo.com/products/robots/bee/bee-bot-family.html. [Última consulta: 27/10/2020].

[38] Terrapin. *Blue-Bot*. https://www.terrapinlogo.com/products/robots/blue/blue-bot-family.html. [Última consulta: 27/10/2020].

[39] Terrapin. *InO-Bot*. https://www.terrapinlogo.com/products/robots/ino/ino-bot.html. [Última consulta: 27/10/2020].

[40] Terrapin. *Pro-Bot*. https://www.terrapinlogo.com/products/robots/pro/probot.html. [Última consulta: 27/10/2020].

[41] Terrapin. *Tuff-Bot*. https://www.terrapinlogo.com/products/robots/tuff/tuff-bot.html. [Última consulta: 27/10/2020].

[42] Thymio.org. *Aseba Studio - Thymio & Aseba*. http://wiki.thymio.org/en:asebausermanual. [Última consulta: 27/10/2020].

[43] Thymio.org. *Specifications Thymio and Aseba*. http://wiki.thymio.org/en:thymiospecifications. [Última consulta: 27/10/2020].

[44] Thymio.org. *Visual programming environment - Thymio and Aseba*. http://wiki.thymio.org/en:thymiovpl. [Última consulta: 27/10/2020]. URL: http://wiki.thymio.org/en:thymiovpl.

[45] usability.gob. *User-Centered Design Basics | Usability.gov*. https://www.usability.gov/what-and-why/user-centered-design.html#:~:text=User%2DCentered%20Design%20Process&text=Design%20is%20based%20upon%20an,process%20and%20it%20is%20iterative.. [Última consulta: 16/7/2022].

[46] J. Visca. *robotito / firmware*. https://gitlab.fing.edu.uy/robotito/firmware. [Última consulta: 02/07/2022].

[47] Whitecat. *THREAD Module · whitecatboard/Lua-RTOS-ESP32 Wiki*. https://github.com/whitecatboard/Lua-RTOS-ESP32/wiki/Thread-Module. [Última consulta: 27/08/2022].

[48] whitecatboard.org. *Home - Whitecatboard.org - The Internet of Things for Humans*. whitecatboard.org. [Última consulta: 02/07/2022].

[49] whitecatboard.org. *What's Lua RTOS? · whitecatboard/Lua-RTOS-ESP32 Wiki*. https://github.com/whitecatboard/Lua-RTOS-ESP32/wiki/What's-Lua-RTOS%3F. [Última consulta: 02/07/2022].

[50] Wikipedia. *Programación dirigida por eventos - Wikipedia, la enciclopedia libre*. https://es.wikipedia.org/wiki/Programación_dirigida_por_eventos. [Última consulta: 13/8/2022].

[51] Otto DIY Workshop. *Otto DIY*. https://www.ottodiy.com/. [Última consulta: 27/10/2020].

Glosario

Android Sistema operativo para dispositivos móviles basado en Linux y otros software de código abierto. 3, 8, 12, 17, 19, 50, 76

Android Studio Entorno de desarrollo integrado (IDE) oficial para la plataforma Android. 52

API Interfaz de programación de aplicaciones. Subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizada por otro software. 50, 58

Componente nativo Componente de software que se ha desarrollado para un sistema operativo en particular. 50

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. 50

Github Plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. 12

Hilo Secuencia de tareas encadenadas muy pequeña que puede ser ejecutada por un sistema operativo. 24, 59, 60

HSV Modelo de color en términos de sus componentes (matiz, saturación y valor). 62, 64

HTML HyperText Markup Language es un lenguaje de marcado para la elaboración de páginas web. 12, 14, 15, 19, 50

IDE Entorno de desarrollo integrado, es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software. 12–16, 18

Interfaz Conexión funcional entre dos sistemas que permite el intercambio de información. 14, 17, 20

Interfaz de usuario Medio con que el usuario puede comunicarse con el sistema. 26

Javascript Lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo con funciones de primera clase. Si bien es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web. 14, 15, 50, 52

JSON JavaScript Object Notation, es un formato de texto sencillo para el intercambio de datos. 15

LUA Lenguaje de programación multiparadigma, imperativo, estructurado y bastante ligero, que fue diseñado como un lenguaje interpretado con una semántica extendible. 24, 55, 61

Omnidireccional Que se puede utilizar en todas las direcciones o sentidos. 22

Open source Código abierto, es un modelo de desarrollo de software basado en la colaboración abierta. 21

Pensamiento computacional Proceso por el cual un individuo, a través de habilidades propias de la computación y del pensamiento crítico, del pensamiento lateral y otros más, logra hacerle frente a problemas de distinta índole. 8

QML Lenguaje basado en JavaScript creado para diseñar aplicaciones enfocadas a la interfaz de usuario. 12

RGB Modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores de luz primarios. 22

Sistema de archivos Sistema de ficheros, controla cómo se almacenan y recuperan los datos. 50, 51

Sistema operativo en tiempo real Sistema operativo que ha sido desarrollado para aplicaciones de tiempo real. Como tal, se le exige corrección en sus respuestas bajo ciertas restricciones de tiempo. 24

SVG Gráficos vectoriales escalables, es un formato de gráficos vectoriales bidimensionales, tanto estáticos como animados, en formato de lenguaje de marcado extensible XML. 39, 47, 53

TCP El protocolo de control de transmisión (en inglés: TCP) es, un protocolo de Internet cuyo objetivo es crear conexiones dentro de una red de datos compuesta por redes de computadoras para intercambiar datos sin errores y en el mismo orden en que se transmitieron. 50, 61, 64

UDP El protocolo de datagramas de usuario (en inglés: UDP) es un protocolo de Internet basado en la transmisión sin conexión de datagramas. No tiene confirmación ni control de flujo, por lo que los paquetes pueden adelantarse unos a otros y tampoco se sabe si ha llegado correctamente. 50, 53, 61

VPL Lenguaje de programación que permite a los usuarios crear programas manipulando elementos gráficos en lugar de especificarlos textualmente. 8, 10–16, 18–20, 43, 44, 59

WebView Componente de software con la tecnología de Chrome que permite a las aplicaciones de Android mostrar contenido web. 14, 16, 49, 50, 52

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. 25, 51, 58

ANEXOS

Modificaciones realizadas a Blockly

En el archivo *workspace.js* del core de Blockly se modificó la función *getTopBlocks* que se utiliza para definir el orden de la generación de código.
El código de la función *getTopBlocks* luego de la modificación se puede ver en la figura 6.1, en donde se comenta el código original que define el órden de los bloques y se añade un nuevo ordenamiento en función de la posición horizontal de los bloques.

```
Blockly.Workspace.prototype.getTopBlocks = function(ordered) {
  // Copy the topBlocks_ list.
  var blocks = [].concat(this.topBlocks_);
  if (ordered && blocks.length > 1) {
    /*this.sortObjects_.offset =
        Math.sin(Blockly.utils.math.toRadians(Blockly.Workspace.SCAN_ANGLE));
    if (this.RTL) {
      this.sortObjects_.offset *= -1;
    }
    blocks.sort(this.sortObjects_);*/
    var posicionesX = [];
    var bloque = blocks[0];
    for(bloque of blocks){
        posicionesX.push([bloque.getRelativeToSurfaceXY().x, bloque]);
    }
    posicionesX.sort((a,b)=>a[0]-b[0]);
    var bloquesOrdenados = [];
    var posi = posicionesX[0];
    for(posi of posicionesX){
        bloquesOrdenados.push(posi[1]);
    }
    blocks = bloquesOrdenados;
  }
  return blocks;
};
```

Figura 6.1: Código modificado de la función *getTopBlocks*

En el archivo *blocks.js* del core de Blockly se modificó la función *getMatchingConnection*. Esta función se utiliza para matchear las conexiones de un bloque respecto a su marcador de inserción [9]. El marcador de inserción de un bloque se puede ver al aproximar algún conector de un bloque hacia otro conector de otro bloque con el que se pueda conectar, en esos casos podemos ver como Blockly muestra una sombra que simula ser el bloque conectado. En la figura 6.2 se puede observar como se ve el marcador de inserción del bloque de *Apagar luces* al aproximar dicho bloque hacia el bloque *Detenerse*.

Figura 6.2: Marcador de inserción del bloque de *Apagar luces*

El marcador de inserción es en realidad un bloque, a bajo nivel, que se crea a partir de la definición del bloque que estamos arrastrando. La función *getMatchingConnection* se utiliza para identificar cuál de los conectores del marcador de inserción es el que corresponde conectar y resaltar.
La razón por la cual se modifica la función *getMatchingConnection* es que la misma chequea que la cantidad de conectores del bloque y de su marcador de inserción coincidan, lo cual no siempre sucede dado que se optó por quitar conectores a los bloques en algunos escenarios, esto implica que el marcador de inserción tendrá todos los conectores originales del bloque pero en algunos escenarios puntuales el bloque original no tendrá todos sus conectores. Dicho escenario se da cuando se arrastra un bloque que no tiene todos sus conectores originales. Esto sucede cuando se desconecta un bloque de sensor o un bloque lógico de adentro de un bloque lógico, como se puede apreciar en la figura 6.3, o cuando se desconecta un bloque de *Estado* de un bloque de acción.

Figura 6.3: Diferencias entre los conectores del bloque y su marcador de inserción

El chequeo dentro de la función *getMatchingConnection* se comentó y se resolvió el error, se realizó la verificación para los tres escenarios en los que sucedía el error. El código

resultante se muestra en la figura 6.4.

```
Blockly.Block.prototype.getMatchingConnection = function(otherBlock, conn) {
  var connections = this.getConnections_(true);
  var otherConnections = otherBlock.getConnections_(true);
  /*if (connections.length != otherConnections.length) {
    throw Error("Connection lists did not match in length.");
  }*/
  for (var i = 0; i < otherConnections.length; i++) {
    if (otherConnections[i] == conn) {
      return connections[i];
    }
  }
  return null;
};
```

Figura 6.4: Código comentado en la función getMatchingConnection

Finalmente, para aumentar el resaltado de los bloques, se modifica el archivo "/core/renderers/common/constants.js" de Blockly, aumentando el valor de la constante *specularConstant* de 0.5 a 0.75. Esta constante define qué tan blanco se vuelve el fondo de los bloques resaltados, a mayor valor el fondo se volverá más blanco y el contraste entre un bloque resaltado y uno no resaltado será más evidente. Este cambio se realiza luego de las pruebas de usabilidad realizadas con usuarios, debido a que dos usuarios comentaron que el resaltado de bloques era apenas visible.

Evaluaciones de prototipos

Una dificultad encontrada al realizar este proceso de prototipado, evaluaciones y mejoras, fue la escasez de usuarios representativos (maestras e investigadores que han utilizado el robot) que se pudieron contactar. Debido a esto hubo que recurrir a voluntarios no tan representativos, aunque siempre se mantuvo el requerimiento de que no tuviesen conocimientos de programación, ya que éste es uno de los factores en donde existe una mayor brecha entre el diseñador y los usuarios finales. Se aprovechó la participación de estos voluntarios para realizar un proceso iterativo de prototipado, evaluación y mejoras del lenguaje, que permitió presentarle a los usuarios representativos un prototipo más refinado.

En la primera iteración participaron cuatro voluntarios.

Figura 6.5: Evaluación con un voluntario

A partir de la retroalimentación obtenida en las pruebas surgieron algunos cambios.
El ícono de *Detección de color*, al tener una tarjeta de color verde por defecto, llevó a algunos usuarios a asumir que solamente servía para sensar color verde, sugirieron añadir franjas de varios colores a la tarjeta para comunicar que detecta colores en general.
Todos los usuarios demoraron en reconocer el significado del ícono de *Detección de objeto*, a algunos no les quedó claro el significado del número. Se decide simplificar el ícono que se muestra por defecto, eliminando el número de sensor, es decir que el bloque dentro de la categoría de Sensores se mostrará sin el número de sensor, el mismo aparecerá una vez que se arrastra el bloque al área de trabajo. Además resultó muy problemático y confuso el concepto de configurar el bloque de *Detección de objeto* para indicar el evento en donde el robot no detecta ningún objeto, esto llevó a realizar algunas modificaciones en el menú de configuración del bloque.
Algunos usuarios demoraron en reconocer el ícono de *Apagar luces*, sugirieron añadir los rayos de luz para que se reconozca más fácilmente la lamparilla. También sugirieron eliminar el fondo gris y dejarlo transparente.

El ícono de *Sonido asustado* fue difícil de reconocer para los usuarios. Esto se dejó para modificar más adelante debido a la dificultad de crear el emoticono en formato SVG.
La categoría y el bloque de *Esperar* fueron malinterpretados por varios usuarios, que al parecer tienen un concepto de espera pasiva, es decir que el término "esperar" les llevó a pensar que el robot se detendría por completo, apagaría las luces y los sonidos y no haría absolutamente nada. Se decide cambiar el nombre de la categoría y del bloque a *Tiempo* y se ajusta el texto que se muestra en el menú de configuración.
El ícono de *Prender luces*, al tener un fondo amarillo por defecto, llevó a algunos usuarios a asumir que solamente servía para prender las luces de color amarillo, algunos usuarios sugirieron cambiar el color de la lamparilla a transparente, otros sugirieron hacer una lamparilla multicolor. Se resuelve cambiar el color de la lamparilla a transparente y evaluar los resultados en las próximas iteraciones, esto además permite mantener una coherencia con los otros dos bloques de luces que tampoco tienen colores.

En la figura 6.6 se muestran las categorías y los bloques luego de los cambios.

Figura 6.6: Categorías y bloques luego de la primera iteración

En la segunda iteración se logró testear el prototipo con tres nuevos voluntarios.
Hubo una mejoría en la interpretación de los íconos por parte de los usuarios, tanto el bloque de *Apagar luces* como el de *Detección de objeto*, el de *Detección de color* y el de *Prender luces* fueron más rápidamente reconocibles por los usuarios, que acertaron en todos los casos al describir el significado.
Hubo algunos desaciertos aislados al describir el significado de otros bloques. Un usuario no entendió el bloque de *Velocidad* y sugirió añadir una flecha y líneas al velocímetro para que sea más similar al de un automóvil. Otro usuario no entendió el bloque de *Luces por arcos* ya que pensó que era para bloquear las luces. En ambos casos se decide tomar nota pero no realizar cambios de momento dado que el resto de usuarios no tuvieron estos problemas.

Los usuarios siguieron malinterpretando en el bloque de *Detección de objeto* el concepto de indicar que no detecta ningún objeto, incluso les cuesta encontrar dicha opción de configuración dentro del menú del bloque. Se decide eliminar dicha opción de configuración por el momento.
Se eliminó el problema de interpretación del bloque de *Tiempo* como una espera pasiva del robot, sin embargo al pedirle a los usuarios que el robot avance durante tres segundos los mismos no se dieron cuenta de que podrían utilizar un bloque de dos segundos y otro bloque de un segundo. Los usuarios recomiendan tener un único bloque configurable, se realiza dicho ajuste.

En la figura 6.7 se muestran las categorías y los bloques luego de los cambios realizados a partir de la retroalimentación obtenida en la segunda iteración.

Figura 6.7: Categorías y bloques luego de la segunda iteración

En la tercera iteración se probó el prototipo con tres voluntarios más.
En esta etapa un usuario identificó el *Sonido triste* como un sonido de disgusto, se tomó nota pero no se hacen cambios al emoticono debido a que es la primera vez que sucede. Más allá de esto no surgió ningún problema relevante, los usuarios lograron reconocer y entender de forma correcta los íconos y crear programas utilizándolos.

Luego de esta instancia, dado que no surgieron cambios a realizarle al prototipo, se realiza la siguiente iteración con tres maestras, dos de las cuales ya habían utilizado al robot anteriormente.
En esta iteración dos de los usuarios tuvieron dificultades para identificar el bloque de *Velocidad*, uno de ellos demoró y el otro directamente lo malinterpretó. En ambos casos la recomendación fue añadir una flecha y líneas simulando las medidas de velocidad en el velocímetro de un automóvil. A estos errores se suma el de la segunda iteración, que también hizo la misma sugerencia, por lo tanto se decide modificar el ícono. Sin embargo,

dado que en esta instancia estamos modificando un ícono que fue previamente validado por diez usuarios, antes de realizar la modificación se envió a dichos usuarios ambos íconos para validar si el nuevo ícono transmite la idea correcta y además determinar cuál de los dos es mejor. Los diez usuarios coincidieron en que ambos íconos transmiten la idea de velocidad, la opinión sobre cuál ícono es mejor estuvo dividida. En base a esto se decide cambiar el bloque de velocidad.
Con el bloque de *Luces por arcos* también hubo problemas ya que dos usuarios tuvieron dificultades en reconocerlo, por lo que recomendaron asignar distintos colores a los arcos. Algo a tener en cuenta para este bloque en particular es que existen 9^4 combinaciones de valores posibles, por lo tanto si definimos cuatro colores por defecto estaremos obligando a todos los usuarios a configurar siempre cada uno de los arcos del bloque. Es por este motivo principalmente que se decide no modificarlo. Si bien puede ser que algunos usuarios principiantes no logren entender fácilmente el significado del ícono en primera instancia, es preferible no someter a todos los usuarios a realizar una configuración de los cuatro arcos de forma innecesaria, ya que es posible que en muchos casos se desee dejar varios de los arcos con las luces apagadas, tal como están por defecto en el bloque actual. A pesar de la dificultad inicial que tuvieron estos usuarios para reconocer el significado del ícono, una vez que se les explicó dicho significado, los mismos no tuvieron problemas en utilizar el bloque para la resolución de escenarios.

En la etapa de resolución de escenarios se le pidió a los usuarios que piensen algún escenario que les gustaría trabajar en el centro educativo con los niños, luego se les pidió que creen el programa y posteriormente se hizo una simulación manual con el robot. Una de las maestras planteó un escenario de preguntas y respuestas, en este escenario surgió la idea de tener movimientos relativos a los sensores, es decir tener algún bloque similar al bloque de *Ir hacia* pero que permita ir en la dirección de alguno de los seis sensores de distancia, esta idea ya se había considerado en etapas iniciales del proyecto pero quedó fuera del alcance inicial del lenguaje, se toma nota para posibles mejoras a futuro. A pesar de no contar con dicho bloque la maestra logró una solución del escenario con la cual estuvo satisfecha. Otra maestra trabajó sobre el escenario de las emociones, ya que fue ella quien lo sugirió en primera instancia. En un momento dado la maestra quiso definir dos comportamientos distintos asociados al mismo evento, lo cual era imposible ya que no se contaba con bloques que permitan definir estados, se toma nota de esta necesidad para futuros cambios al lenguaje. Más allá de esto se logró una implementación del escenario que resultó satisfactoria para el usuario. En ambos casos las maestras se mostraron entusiasmadas al interactuar con el prototipo y al finalizar comunicaron que les gustó. Finalmente, la tercera maestra que se ofreció para probar el prototipo es maestra de informática de quinto y sexto grado, con lo cual no es un usuario realmente representativo

ya que trabaja con niños de mayor edad y además sabe programar. Para esta maestra el prototipo se quedó muy corto, ella quería enseñarle a sus alumnos a programar y tenía expectativas de poder definir eventos más complejos, condicionales y estados. Más allá de esto se pudo sacar provecho de la identificación de bloques del lenguaje por parte de la maestra.

Luego de esta iteración con usuarios finales, dado que hubieron muy pocas dificultades con la identificación de los bloques, que los usuarios lograron resolver los escenarios que propusieron y que se mostraron satisfechos con el prototipo, se decide realizar la implementación del prototipo en la aplicación móvil.

En la figura 4.7 de la sección 4.2.2.1 se muestran las categorías y los bloques luego de los cambios realizados a partir de la retroalimentación obtenida en la cuarta iteración. También se añade una nueva versión del bloque de *Sonido asustado*, el cual había resultado confuso para los usuarios. El mismo deberá ser validado durante las pruebas de usabilidad que se realizarán (ver sección 5).