UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Blokino - una plataforma para programar objetos físicos en las escuelas

Autor: Jorge Roman Jair Farfan Coaguila

Directora: Claudia Queiruga

Asesora profesional: Vanessa Aybar Rosales

Carrera: Licenciatura en Sistemas

Resumen

En los últimos años múltiples políticas de estado de nuestro país han incluido a la programación y a la robótica educativa en el sistema de educación obligatorio, como área de conocimiento innovadora vinculada a la experimentación y construcción de conocimiento con tecnologías digitales. La intención de esta tesina de grado de Licenciatura en Sistemas es desarrollar e implementar una plataforma de robótica educativa, llamada Blokino, de acceso libre, basada en software libre, hardware libre y con contenidos abiertos, que colabore con la enseñanza de la programación y la robótica en la educación secundaria. Blokino permite construir y programar objetos electrónicos, adopta el paradigma de programación visual basada en bloques visuales, de uso intuitivo y amigable, favoreciendo el aprendizaje de programación en las aulas de la escuela secundaria.
Blokino se propone como material educativo a ser adoptado por las escuelas de nuestra región a través del proyecto de extensión de la Facultad de Informática “Extensión en vínculo con escuelas secundarias”, con la intención que las y los estudiantes puedan llevar adelante proyectos sencillos de robótica, usando elementos de hardware libre, de bajo costo y frecuentemente disponibles en las escuelas.

Palabras Claves

Escuelas, Enseñanza de Programación, Proyectos de Extensión, Metodología DIY, Programación Visual basada en Bloques, Software Libre, Hardware Libre, Arduino, Sensores, Electron, JavaScript, Javascript Robotics, NodeBots, NodeJS, Robótica educativa, VanillaJS, VueJS.

Conclusiones

El estudio realizado para desarrollar una plataforma educativa que pueda controlar hardware libre de la familia Arduino usando un lenguaje visual basado en bloques como Google Blockly resultó en la implementación de Blokino. El proceso de desarrollo permitió ganar experiencia en la integración de herramientas libres, entre ellas Electron y Johnny-Five, para controlar componentes electrónicas desde la programación con JavaScript.
Blokino se publicó en una página web que contiene todo lo necesario para descargarlo y aprender a usarlo. Se encuentra disponible en: **www.blokino-plataform.com**.
El código fuente de Blokino está disponible en **https://github.com/georgefarfan/blokino** junto con la documentación necesaria. El proyecto es de acceso público alentando de esta manera la conformación de una comunidad de desarrolladores de Blokino que aporten mejoras.
Las pruebas de campo con estudiantes de escuelas secundarias que participan del proyecto “Extensión en vínculo con escuelas secundarias” permitió recolectar información útil sobre la experiencia de usar Blokino, focalizándose en que las y los estudiantes puedan crear programas sencillos que se ejecuten sobre placas de hardware libre.

Trabajos Realizados

- Implementación de la estructura de Blokino, usando VanillaJS.
- Implementación de módulos JavaScript para adaptar las herramientas de Google Blockly, JavaScript Robotics, Electron y controlar componentes electrónicas de hardware libre.
- Integración de JavaScript Robotics con Electron.
- Selección de un conjunto de componentes electrónicos de la familia Arduino que conforman el kit Blokino.
- Implementación de NodeBots.
- Evaluación de la plataforma Blokino con estudiantes de escuelas secundarias.
- Difundir la plataforma Blokino en espacios de profesionales informáticos.

Trabajos Futuros

- Migrar la estructura de la plataforma Blokino a un framework de JavaScript para alcanzar escalabilidad y mejor legibilidad.
- Implementar módulos multiplataforma para el control del hardware libre.
- Implementar módulos para el reconocimientos de dispositivos bluetooth.
- Construir un motor de validación genérico para controlar los desafíos de aprendizaje de Blokino.
- Realizar pruebas de concepto para mejorar el rendimiento y consumo de memoria de la plataforma.
- Continuar con las pruebas de campos con las escuelas de la región.

Fecha de la presentación: Marzo de 2020

Facultad de Informática
Universidad Nacional de La Plata

Blokino: una plataforma para programar objetos físicos en las escuelas

Tesina de grado para la Licenciatura de Sistemas

Alumno: Jorge Roman Jair Farfan Coaguila
Directora: Claudia Queiruga
Asesora profesional: Vanessa Aybar Rosales

Agradecimientos

Quisiera agradecer a toda mi familia, comenzando por mis padres Jorge y Luisa que siempre estuvieron presentes durante todas las etapas de mi vida. Este camino comenzó hace muchos años, que culmina con un título universitario que simboliza un triunfo personal y un regalo para ellos por su sacrificio de todos estos años. Con su apoyo, amor y consejos hoy puedo recibirme de licenciado en sistemas. A mi compañera de vida Cinthia, por estar presente durante mi etapa de estudiante y vivir conmigo cada uno de esos momentos. A mi hermana Milagros por los ánimos y el apoyo. Y a mis abuelos Jorge y Victoria, que a pesar que están lejos siempre me han aconsejado y los tengo presentes en todo lo que hago.

También quiero agradecer a Claudia Queiruga y Vanessa Aybar Rosales, por su orientación y su predisposición para realizar este trabajo.

Finalmente, a la Facultad de informática y a la Universidad Nacional de La Plata , que me dieron todos los recursos para que pueda formarme como profesional.

Índice General **6**

Índice General

Capítulo 1 - Introducción

1.1 Contexto

En los últimos años múltiples políticas de estado han incluido a la robótica educativa en el sistema de educación obligatorio, como área de conocimiento innovadora vinculada a la experimentación y construcción de conocimiento con tecnologías digitales, entendiendo que el dominio de estas tecnologías se vincula con dinámicas de inclusión/exclusión social.
La robótica es una rama de la ingeniería mecatrónica que combina áreas relacionadas con la ingeniería mecánica, electrónica, física e ingeniería informática.
La aplicación de la robótica se centra en el armado de piezas electrónicas y en la programación para la automatización de diversas tareas.
Podríamos afirmar que las tecnología digitales han modificado la vida de las sociedades, hoy la mayoría de nuestras actividades cotidianas se encuentran mediadas por diversas tecnologías digitales: nos comunicamos, estudiamos, trabajamos, compramos, nos entretenemos y vinculamos con otras personas, entre otras tantas actividades. Las competencias digitales, la programación y la robótica educativa, puestas en diálogo con las materias tradicionales de la escuela proponen modelos de enseñanza y aprendizaje para los estudiantes del sistema obligatorio desde un enfoque transversal (Ministerio de Educación de la Nación, 2017) (Benitti F., 2012) (Díaz J. et al, 2015) (Banchoff Tzancoff C. et al, 2019) que favorece la experimentación y por tanto la apropiación de conocimientos. La robótica educativa alienta la realización de actividades creativas en tanto es posible armar artefactos digitales de acuerdo a necesidades e intereses, asimismo favorece el trabajo colaborativo, basado en proyectos, con especial atención en la capacidad de resolución de problemas, de manera critica y analitica, poniendo al estudiante como protagonista en el proceso de enseñanza y aprendizaje.
Asimismo desde la Secretaría de Extensión de la Facultad de Informática, se viene trabajando fuertemente a través del proyecto “Extensión en vínculo con escuelas secundarias”[1] en la difusión de la informática como campo del saber en las escuelas de la región y en la adopción crítica de las tecnologías digitales, con un enfoque que recoge ideas del pensamiento computacional (Wing J., 2006) (Aho A., 2012) (Denning P., 2017) y de la robótica educativa pensada en términos de hardware y software libre (Queiruga C. et al, 2019) (Martínez C. et al, 2015)
Este contexto es propicio para pensar desde el espacio de una tesina de grado en Informática, en el desarrollo de herramientas tecnológicas integradas a proyectos de la facultad que puedan ser adoptadas por escuelas de la región y que contribuyan con las políticas educativas federales.

[1] Página de proyectos de extensión de la Facultad de Informática: https://tinyurl.com/tygbc72

1.2 Motivación

La motivación de este trabajo radica en el desarrollo de una plataforma de robótica educativa que favorezca el aprendizaje de conceptos de programación y el uso de componentes electrónicos de Arduino, en el nivel educativo secundario, como así también el armado de proyectos bajo el paradigma DIY (Do It Yourself) usando placas Arduino.

Actualmente es posible encontrar múltiples plataformas de robótica educativa, segmentadas por edades, entre ellas CodeBug («CodeBug», 2015) o Bee-Bot («Bee-Bot», 2015), Scratch («Scratch». 2003) o MBlock («MBlock», 2011). CodeBug es una placa en forma de insecto, compuesta por de leds y sensores. Esta plataforma se puede usar para pequeños proyectos de programación. Bee-Bot, es una placa integrada con componentes electrónicos, tiene forma de una abeja, que a diferencia de CodeBug tiene movimientos mediante sensores. Las plataformas Scratch y MBlock, usan lenguajes de programación visual basados en bloques, facilitando el aprendizaje de programación mediante el uso de la metáforas de bloques que se encastran al estilo de los juegos LEGO. CodeBug y Bee-Bot a diferencia de Scratch y MBlock, tienen un funcionamiento limitado y muy orientado, debido a que el hardware con el que se implementaron tiene un objetivo muy definido, compuesto por un grupo de funcionalidades acotadas.

Actualmente, diferentes políticas educativas en nuestro país han impulsado la enseñanza de la programación en las escuelas, entre ellas la Resolución Nº 263/15 (Res. 263/15) del Consejo Federal de Educación (CFE) que declaró "...la importancia estratégica para el sistema educativo argentino la enseñanza y el aprendizaje de la programación durante la escolaridad obligatoria" y más recientemente la Resolución 343/18 (Res. 343/18) del CFE que aprobó los Núcleos de Aprendizajes Prioritarios (NAP) de Educación Digital, Programación y Robótica, favoreciendo su integración curricular en la educación obligatoria. Bajo este marco normativo se vuelve imprescindible contar con material didáctico apropiado para acercar las ideas de programación al aula de la escuela teniendo en cuenta, además, entre otras cuestiones, la disponibilidad de equipamiento e infraestructura tecnológica de las escuelas. Algunas políticas educativas a lo largo de las últimas décadas han provisto de diferente equipamiento tecnológico a las escuelas, siendo éste particular en las diferentes jurisdicciones. Esto ha permitido que varias escuelas cuenten con kits de robótica educativa e introducir conceptos de programación de computadoras articuladamente con electrónica y así promover el aprendizaje de modelos computacionales en el ámbito escolar.

Como se describió previamente es posible encontrar herramientas y plataformas de robótica educativa, algunas de ellas han sido adoptadas en las escuelas para trabajar con robótica y programación con bloques, entre ellas se encuentran Scratch («Scratch», 2003), Micro bit («Micro bit», 2012) y mBlock(«Micro bit», 2011). La Figura 1.2.1 muestra capturas de pantalla de estas herramientas.

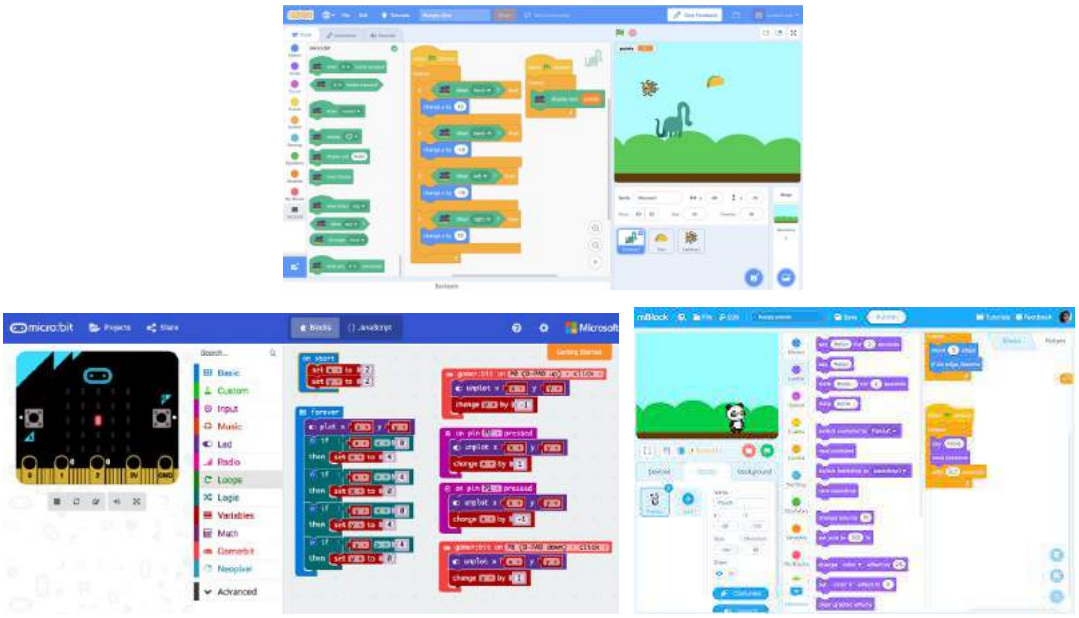


Figura 1.2.1 - Scratch - Micro bit - MBlock

Las herramientas mencionadas aunque son adoptadas en las escuelas, tienen limitantes en relación al acceso a las mismas. Ejemplo de ello son CodeBug, Bee-Bot y Micro bit, que sólo puedan usarse con hardware específicamente diseñado, impidiendo su uso y por consiguiente probar los programas. Para compensar esta restricción con Bee-Bot es posible simular los programas mediante una figura que hace referencia al hardware, pero no se cumple el objetivo principal de crear programas y probarlos en hardware. El limitante fundamental de estas herramientas es el hardware y el software, se trata de proyectos cuyo código es de acceso privativo y de hardware específico del fabricante, en general costoso. Por otro lado están Scratch y MBlock, ambos proyectos son de código fuente abierto, en los que participan una gran comunidad de programadores. Estos proyectos son similares en muchos aspectos visuales y funcionales. MBlock está orientado a realizar proyectos con placas electrónicas Arduino y hardware diseñado específicamente para funcionar con MBlock, que se adquiriere por la web oficial de Mblock. Con Scratch también es posible ejecutar programas en placas Arduino, pero con la diferencia que la configuración es compleja.

La intención de esta tesis es construir una plataforma de robótica educativa, llamada Blokino, de acceso libre, con contenidos abiertos y basada en software libre y hardware libre, que colabore con la enseñanza de la robótica educativa en la educación secundaria. Blokino adopta el paradigma de programación visual basada en bloques visuales, de uso intuitivo y amigable, permitiendo darle comportamiento a objetos físicos construidos con componentes electrónicos.

Capítulo 2 - Hardware de código abierto

Durante las décadas de los cincuenta y sesenta, el software era concebido como conocimiento libre, grupos de investigadores y académicos producían sistemas operativos y programas, y compartían el código para modificarlo y mejorarlo. A principios de los años ochenta este enfoque cambió y mediante el registro de marca, los derechos de autor y cánones, las empresas que vendían computadoras comenzaron a cobrar por licencias de uso del software, restringiendo así su uso y desarrollo.

Richard Stallman y su proyecto GNU, a partir de 1983, comenzó a cambiar el panorama de la construcción de software. El objetivo de este proyecto es poder escribir un sistema operativo completo, libre de restricciones para el uso, modificación y distribución. Uno de los motivos que impulsaron este proyecto fue la experiencia que atravesó Richard Stallman con el uso de su impresora que no podía ser arreglada, porque el código no estaba público. De esta manera se comenzó a desarrollar el concepto de software libre. Éste fue quizás una de las posibles motivaciones que impulsaron la creación de hardware libre. Al igual que las actuales comunidades de programadores que se reúnen para compartir conocimientos y recursos, también comenzaron a desarrollarse comunidades de desarrollo de hardware libre. Entre las primeras comunidades de hardware libre podemos citar a la Homebrew Computer (Homebrew Computer Club, 2018), donde aficionados de la electrónica como Lee Felsentein, reconocido en 1994 por la Electronic Frontier Foundation (EFF) como el “Pionero de la frontera de la electrónica”, compartían e intercambiaban piezas, circuitos e información referente a los microprocesadores de la época.

La primera reunión de este club tuvo lugar en marzo de 1975 y los encuentros se prolongaron hasta diciembre de 1986. Durante estos 11 años, esta comunidad impulsó la actividad de compartir diseños de hardware, para que otros no sólo pudiesen modificarlos libremente sino además encontrarles nuevas utilidades, con el fin de devolverlos mejorados a la comunidad.

Mientras que con el software es posible compartir las modificaciones, adiciones o correcciones que se van realizando mediante el conocimiento adquirido a través del estudio del código y su funcionamiento, lo mismo sucede con el hardware libre a través de sus esquemas, las listas de materiales, la disposición física de los componentes y cualquier elemento adicional para lograr su funcionamiento.

A pesar que el concepto de hardware libre es menos conocido que el del software libre, lo cierto es que se han realizado proyectos interesantes con el paso de los años. Su objetivo es crear diseños de aparatos informáticos de forma abierta, de manera que todas las personas puedan acceder, como mínimo a los documentos de ensamblaje de los dispositivos. La información sobre la manera de comunicarse con el hardware, el diseño del mismo y las herramientas usadas para crear ese diseño deben ser publicadas para ser usadas libremente. De esta manera se facilita el control, implementación y mejoras en el diseño por la comunidad de desarrolladores.

El movimiento de hardware libre quedó estancado durante los últimos años del siglo XX, sin embargo el uso masivo de Internet ayudó a impulsarlo mediante la metodología DIY, del inglés “Do It Yourself”, que en español significa “Hazlo tu mismo”**.** El movimiento DIY (**«**Introduction to DiY**»**, 2013) surgió como una filosofía, basándose en no comprarlo todo y sustituirlo por el trabajo que se puede hacer por medios propios. Uno de los

precursores de este movimiento fue la cultura “punk”, que al rechazar ideas capitalistas comenzaron a fabricar sus propios instrumentos musicales e incluso su propia ropa. De esta manera lo que surgió como una idea contracultural se fue extendiendo entre la población y se convirtió en una filosofía adoptada por muchas personas. Con el tiempo el movimiento DIY se fue asociando al movimiento Maker, que es otra corriente que corresponde a la misma filosofía, pero que adquiere un nombre propio. Comúnmente conocido como Maker Movement (**«**Maker Movement**»**, 2017) está más cerca al entorno tecnológico hecho por uno mismo. Al igual que el movimiento DIY surgió como una filosofía anticapitalista y basándose en los principios de constructivismo, la cultura *maker* tuvo como primer referente la ética del DIY. La cultura Maker se centra en el aprendizaje activo en un ambiente social; hace hincapié en el aprendizaje informal, en red, en pareja y compartido, motivado por la diversión y la auto-realización; fomenta nuevas aplicaciones de tecnologías, y la exploración de intersecciones entre dominios de conocimiento y formas de trabajo concebidas tradicionalmente separadas.
Con estos movimientos y el uso de Internet los movimientos DIY y Maker han ido proliferando, dado que facilitan y simplifican el acceso público a diagramas, planos y diseños lógicos del hardware.
Estas son algunas de las iniciativas más relevantes de los movimientos DIY y Maker, en torno al hardware libre, que promueven desde la creación de computadoras hasta la fabricación de proyectos didácticos:

- **Arduino**: se basa en una placa base que incorpora un sencillo microcontrolador y un entorno de desarrollo para crear aplicaciones para dicha placa. Los proyectos que se pueden hacer con Arduino pueden ser desde robots educativos hasta sistemas de riego automático. Está disponible con una licencia Creative Commons que otorga libertad de desarrollo, aunque los productos derivados deben cumplir determinadas pautas, sobre todo en lo relativo a la propia denominación de Arduino, lo cual viene a ser una especie de control de marca.
- **Raspberry Pi**: es un ordenador del tamaño de una tarjeta de crédito que consta de una placa base sobre la que se ensambla un procesador, un chip gráfico y memoria RAM. Fue presentado en 2009 por la Fundación Raspberry Pi para estimular la enseñanza de informática en escuelas de todo el mundo. Esta propuesta cuenta con una notable comunidad de desarrolladores dispuestos a compartir paso a paso las nuevas funciones que van encontrando para la placa.

Para el desarrollo de Blokino se optó por elegir la plataforma Arduino, dado que es una plataforma adecuada para la enseñanza y prototipado de proyectos simples orientados a la programación y la aplicación de la metodología DIY & Maker. Por otro lado Raspberry Pi también puede ser usado para el mismo fin, pero a diferencia de Arduino que es un microcontrolador, Raspberry Pi es una computador completamente funcional.

2.1 Arduino: descripción, composición y funcionamiento

Arduino es una plataforma de hardware abierto, basada en una placa electrónica con un microprocesador y un entorno de desarrollo integrado (IDE), que favorece la construcción de proyectos multidisciplinarios. Los diferentes diseños de la plataforma Arduino se distribuyen bajo licencia Creative Commons. Una característica interesante de la plataforma Arduino es que el entorno de desarrollo es multiplataforma, pudiendo

programarse en múltiples sistemas operativos, tales como Windows, Macintosh OS y Linux. Arduino está compuesto por:

- **Un microcontrolador abreviado MCU**: es un circuito integrado programable capaz de ejecutar las órdenes grabadas en su memoria interna. Este tipo de microcontrolador contiene en su interior las 3 principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida.
- **Periféricos de entrada**: son los pines usados para hacer lecturas. Las distintas placas Arduino tienen un número diferente de pines digitales y analógicos. Por ejemplo en las placas Arduino UNO tiene 13 (0-13) pines digitales y 5 (A0-A5) pines analógicos. Los pines de salida se usan para el envío de señales, son los digitales (0-13).
- **Pines**: las placas Arduino disponen de una serie de entradas y salidas digitales y analógicas programables, que son la base para el manejo de Arduino. Los pines principales de la placa son:
    - GND: tensión eléctrica de tierra
    - 5V: tensión de 5 voltios.
    - 3.3V: tensión eléctrica de 3.3 voltios.
    - REF: voltaje uniforme.
    - TX (transmisión) y RX (recepción): canales usados para comunicación serial. Comúnmente se utilizan para conectar a módulos de Bluetooth.
    - RESET: resetea la placa Arduino.
    - VIN: es otro pin para alimentar las placas. Aunque el pin VIN sirve para alimentar la placa, lo más común es hacerlo mediante el *jack* de alimentación usando una tensión de 7-12 voltios; otra opción de alimentación es usar el puerto USB. Está ultma opción es la que se va a usar en Blokino.

El microcontrolador de Arduino es un circuito integrado en el cual se graban las instrucciones, que son enviadas desde el IDE. El lenguaje de programación que usa Arduino, denominado Arduino, es una fusión de las características de los lenguajes de programación Processing[2] y Wiring[3].

El IDE de Arduino está construido con Java y usa el compilador GCC para C/C++ para facilitar las operaciones de entrada y salida. Arduino utiliza comunicación serial para el envío de datos a la placa.

2.2 Arduino hoy

El crecimiento movimiento DIY favoreció la popularización de Arduino, favoreciendo el desarrollo de proyectos en múltiples áreas de aplicación, entre ellas automatización

[2] Processing que es un lenguaje de programación y entorno de desarrollo basado en Java, de código abierto y bajo la licencia GNU GPL. Se inició en 2001 en el MIT Media Lab por Ben Fry y Casey Reas a partir de reflexiones en el "Aesthetics and Computation Group del MIT". Está pensado para no programadores, para diseñadores audiovisuales que quieren crear proyectos multimedia. Es posible realizar gráficos 2D, 3D, texturas y formas geométricas entre otras funcionalidades útiles.

[3] Wiring es una plataforma de software más que un lenguaje de programación. Se inició en 2004 por Hernando Barragán, alumno de Ben Fry y Casey Reas en el instituto IVREA basándose en Processing bajo una licencia de código abierto GNU GPL. Se pensó para diseñadores y artistas que quisieran dar un paso más y adentrarse en el mundo de la electrónica, centrándose desde el principio en los fenómenos físicos interactivos, ideas o conceptos, más que en el código y la electrónica. Wiring permite crear programas y generar prototipos con electrónica, como así también permite la manipulación de dispositivos conectados a un microcontrolador.

industrial, domótica, herramientas de prototipado, plataforma de entrenamiento para aprendizaje de la electrónica, tecnología para artistas, monitorización, adquisición de datos, aprendizaje de habilidades tecnológicas y programación, etc.

La robótica educativa es otra de las áreas de aplicación de Arduino, habiendo conducido a instituciones educativas como escuelas y universidades ha adoptarlo como herramienta de enseñanza.

Por otra parte los usos de Arduino están relacionado con Internet de las Cosas (IoT), Internet de Todo y Máquina a Máquina (M2M). Los conceptos detrás de estos términos hacen referencia a la interconexión digital de objetos de uso cotidiano a través de Internet, a la conformación de múltiples tecnologías, entre ellas sensores que conectan el mundo físico con el digital, computadoras que procesan la información recogida y plataformas web donde se procesan y almacenan los datos. Esta infraestructura de red inteligente permite obtener una perspectiva valiosa de datos para optimizar las automatizaciones.

Por último, un aspecto sumamente valioso de Arduino es su comunidad, dado que promueve la continuidad de su desarrollo, compartiendo el conocimiento generado, mediante la elaboración de librerías para facilitar su uso, publicando sus proyectos para que puedan ser replicados, mejorados o ser usados como punto de partida para otros proyectos relacionados

Capítulo 3 - Blokino

3.1 ¿Qué es Blokino?

Blokino es una plataforma de software de libre, orientada a la enseñanza de la programación de objetos físicos digitales educativos y la construcción de los mismos. Está basado en la plataforma Arduino y en software libre.
Además, Blokino es multiplataforma, funciona sobre los sistemas operativos Linux y Windows; los objetos digitales que se pueden programar con Blokino usan diferentes placas Arduino y sus componentes electrónicos y las funcionalidades usadas dentro de la plataforma están enteramente desarrolladas con JavaScript.

3.2 Kit de componentes electrónicos

Para la selección de los componentes que conforman el kit de Blokino se tuvieron en cuenta lineamientos vinculados a la compatibilidad, facilidad de uso (Perch K., 2015), costo y aspecto estético. Se realizaron distintas pruebas sobre componentes electrónicos que permitieron garantizar estos lineamientos.

- **Compatibilidad con JavaScript**: Blokino no ejecuta scripts en Processing ni C/C++ como se hace en Arduino. Para lograr ejecutar los programas JavaScript en las placas Arduino fue necesario buscar una librería que permitiera hacerlo.
  La librería Johnny-Five («Johnny-Five», 2012) es la opción elegida dada su estabilidad. Así mismo se evaluaron otras opciones como Cylon.js y Node-Serialport y Node-Red.
  Al incluir los componentes electrónicos a Blokino se encontraron inconsistencias en la librería para el desarrollo de la GUI (Graphic User Interface) y librerías internas que componen Johnny-Five, que debieron ser atendidas. Esto llevó a hacer pruebas con distintos componentes electrónicos, que son tenidos como segunda opción en varios proyecto de IoT.
- **Facilidad de uso**: durante el armado de los diferentes circuitos, se comenzó a evidenciar el aumento del cableado necesario para hacer las conexiones de los cables dupont de los componentes que se conectan a la placa Arduino. Esto resultó en circuitos complejos de usar y manipular. Por ello, se buscó opciones para reemplazar componentes que necesiten muchos conexiones. Para esto se optó por componentes electrónicos con circuitos integrados, para disminuir la cantidad de cables dupont.
- **Costo**: los kits de Arduino tienen una gran variedad de componentes electrónicos, como así también gran variedad de costos dependiendo de la marca. Se buscó variedad para lograr un precio accesible.
- **Aspecto estético**: para el armado de los circuitos electrónicos se buscó componentes electrónicos que sean amigables con los intereses de las y los jóvenes. Estos componentes electrónicos se clasificaron según la importancia y el uso dentro del armado de los circuitos.

3.3 Placas Arduino soportadas

Las placas Arduino que se eligieron son Arduino UNO, NANO y MEGAs dado que son las más comúnmente usadas en las prácticas de IoT en educación. No es necesario que sean placas Arduino originales (aunque sería ideal que lo fueran) dado que el soporte para las placas depende de una configuración que se realiza desde la plataforma Blokino con Firmata[4]**.** La Figura 3.3.1 muestra una imagen de los 3 modelos de placas Arduino utilizadas en Blokino.

Figura 3.3.1 - Arduino Nano, MEGA, UNO

Cada una de estas placas está pensada para un uso en particular, tienen en común el modelo de procesador interno, pero son distintas en cuanto al diseño, función y el fin para el que se puede usar:

- La placa Arduino UNO es la placa estándar que es usada para cualquier proyecto de Arduino, ideal para iniciarse en los proyectos de IoT.
- La placa Arduino MEGA se usa para proyectos donde se necesiten más entradas digitales y analógicas, dado que tiene en total 54 pines digitales de Entrada/Salida (15 pines tienen salida PWM). Es útil para proyectos donde los circuitos son más grandes y complejos.
- La placa Arduino NANO es una versión minificada de la placa Arduino UNO. Es adecuada para proyectos donde se tiene poco espacio para acomodar la placa electrónica y el circuito.

3.4 Componentes electrónicos

Cuando se seleccionaron los componentes electrónicos se tuvo en cuenta la compatibilidad y se los clasificó según la importancia para la construcción de los circuitos:

- Componentes principales: placa Arduino, cable usb para la transmisión serial de datos, protoboard, cables dupont de los distintos tipos.
- Componentes de uso: leds, leds-rgb, potenciómetro, buzzer, botones, pulsadores, interruptores, servomotores, servomotores continuos, motores, sensor de movimiento, sensor de proximidad.
- Componentes con módulo integrado: pantallas LCD, joystick, matriz-leds, teclado capacitivo MPR121.

[4] Firmata es un protocolo genérico para la comunicación con microcontroladores desde cualquier software instalado en un ordenador. El objetivo de este protocolo es hacer que el microcontrolador sea una extensión del entorno de desarrollo donde se ejecute.

- Componentes de soporte: resistencias, transistores y diodos.

3.4.1 Componentes principales

Protoboard: prácticamente son tableros con puntos que se encuentran conectados eléctricamente entre sí y que permiten construir circuitos. Las conexiones en un protoboard se hacen con sólo insertar los componentes, permitiendo armar y modificar circuitos con mayor velocidad. La Figura 3.4.1.1 muestra imágenes de diferentes protoboard.

Figura 3.4.1.1 - Protoboard

Las protoboards están formadas por 2 bandas esenciales:

- Canal central: está ubicado en la parte central del tablero y es justo donde van conectados los circuitos integrados, para lograr el aislamiento de los pines de los dos lados.
- Buses: localizados a los lados de la protoboard, identificados por franjas de color negro o azul que indican el bus de tierra; pero también, por franjas rojas que denotan el bus de voltaje positivo. Es justamente donde se encuentran los orificios del protoboard y como ya se ha hecho referencia, están separadas en filas conectadas entre sí. Las filas se identifican por números, mientras que las columnas por letras.

Estos tableros tienen distintas medidas y propósitos, basadas en puntos de conexión:

- Formato de 832 puntos: es el tamaño más usado, permite montajes complejos o montajes simples con los componentes organizados y espaciados. Tiene 2 pares de líneas de alimentación, igual que el formato de 400 puntos. Son usadas para proyectos con circuitos complejos.
- Formato de 400 puntos: tienen 2 pares de líneas de alimentación y un total de 400 puntos de conexión para montajes. Denominadas “media protoboard” por ser la mitad de tamaño que el formato grande. Son usadas para circuitos de complejidad media, dado que se acota el número de puntos.
- Formato de 170 puntos: son las protoboards pequeñas que carecen de líneas de las bandas de alimentación y simplemente tienen 2 mitades con líneas de 5 conexiones, haciendo un total de 34 líneas. Sirve para montajes muy básicos y permiten ahorrar espacio. Son usadas para circuitos pequeños.

La corriente con la que puede operar una protoboard varía entre 3 A y 5 A, y esto depende del fabricante. Suelen operar a bajas frecuencias, entre 10 MHZ – 20 MHz.

Blokino tiene soporte para las distintas protoboards dado que son una extensión para el armado de circuitos y no están sujetas a la configuración vital que se hace sobre las placas Arduino.

Cables dupont: se usan para interconectar los componentes electrónicos con las protoboards. A través de los cables dupont se transfieren señales eléctricas que envían los datos o impulsos generados a partir del código de los programas de Blokino. La Figura 3.4.1.2 muestra figuras de cables dupont.

Figura 3.4.1.2 - Cables dupont

Los tipos de cables dupont que se incluyeron en los kits de Arduino son: Cables Macho - Macho, Cables Macho - Hembra y Cables Hembra - Hembra.
Para el armado de circuitos con Blokino, se seleccionaron estos 3 tipos de cables dupont para la conexión o adaptación de los componentes electrónicos.

3.4.2 Componentes de uso

Leds: son diodos emisores de luz que se iluminan cuando la electricidad pasa por ellos. El ánodo, que es la clavija más larga, se conecta a la corriente, y el cátodo, que es la más corta, se conecta al pin de datos. La Figura 3.4.2.1 muestra imágenes de diferentes Leds.

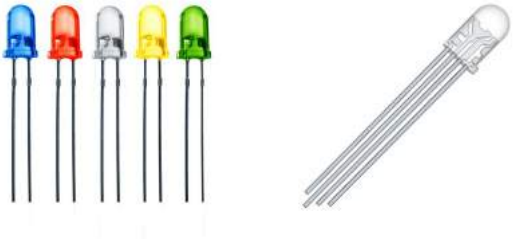


Figura 3.4.2.1 - Leds

Por tratarse de dispositivos electrónicos semiconductores, los leds funcionan con corriente continua, tienen polaridad y es imprescindible para su funcionamiento que sean conectados en el sentido correcto.
Potenciómetros: son resistencias eléctricas cuyo valor de resistencia es variable. Permiten controlar la intensidad de corriente eléctrica que fluye por un circuito si se conecta en paralelo, o la diferencia de potencial al conectarlo en serie. La Figura 3.4.2.2 muestra imágenes de diferentes modelos de potenciómetros.

Figura 3.4.2.2 - Potenciómetros

Buzzers: son dispositivos que permiten convertir una señal eléctrica en una onda de sonido. Estos dispositivos no disponen de electricidad interna, por lo que tenemos que proporcionar una señal eléctrica para conseguir el sonido deseado. La Figura 3.4.2.3 muestra imágenes de buzzers.

Figura 3.4.2.3 - Buzzer pasivos

Técnicamente tanto buzzers como altavoces son transductores electroacústicos, es decir, dispositivos que convierten señales eléctricas en sonido. Estos componentes electrónicos sólo son capaces de generar un tono a la vez, por lo que con Blokino se pueden generar una secuencia de tonos y pausas asimilando en tonadas a melodías o canciones que se quieran replicar en base de tonos.

Pulsadores: son botones que permiten realizar acciones mientras se presiona sobre el componente electrónico, en ese momento se cierra el circuito y se deja pasar la corriente. La Figura 3.4.2.4 muestra diferentes modelos de botones.

Figura 3.4.2.4 - Botones estándar, Switch y Pulsadores.

Servomotores: son un tipo de motor de engranajes que pueden girar 180° o 360°, además tienen una capacidad de ubicarse en cualquier posición dentro de su rango de operación. Se controla enviando impulsos eléctricos desde el microcontrolador Arduino. Estos impulsos le dicen al motor en qué posición debe colocarse. La Figura 3.4.2.5 muestra imágenes de diferentes modelos de servomotores.

Figura 3.4.2.5 - Servomotores clásicos y continuos

Los modelos usados con Blokino son SG90 Tower Pro, de gran calidad, diminutas dimensiones y económicos. Funcionan con la mayoría de tarjetas electrónicas de control con microcontroladores y además con la mayoría de los sistemas de radio control comerciales. Este tipo de motores es ideal para las primeras experiencias de aprendizaje y prácticas con servomotores, dado que sus requerimientos de energía son bastante bajos y es posible alimentarlos con la misma fuente de alimentación que el circuito de control.
Otro tipo de servomotor probado con Blokino es el de rotación continúa. La característica de este tipo de servomotor es que la rotación es de 360°, además de tener una resistencia de giro más consistente que los modelos SG90 Tower Pro.
Motores: son motores de corriente continua que convierten la energía eléctrica en mecánica. Se compone de dos partes:

- El estator: la parte mecánica del motor donde están los polos del imán.
- El rotor: la parte móvil del motor con devanado y un núcleo al que llega la corriente a través de las escobillas.

Cuando la corriente eléctrica circula por el devanado del rotor se crea un campo electromagnético. Éste interactúa con el campo magnético del imán del estator que deriva en un rechazo entre los polos del imán del estator y del rotor, creando un par de fuerza donde el rotor gira en un sentido de forma permanente. La Figura 3.4.2.6 muestra imágenes de motores de corriente contínua.

Figura 3.4.2.6 - Motores

Sensores de movimiento: son sensores infrarrojos para la detección de movimiento. En general son pequeños, de baja potencia, económicos y fáciles de usar. Por esta razón son utilizados frecuentemente en juguetes electrónicos, aplicaciones de domóticas o sistemas de seguridad.
Estos sensores se basan en la medición de la radiación infrarroja. Todos los cuerpos (vivos o no) emiten una cierta cantidad de energía infrarroja, mayor cuanto mayor es su temperatura. Los sensores infrarrojos disponen de un sensor piezoeléctrico capaz de captar esta radiación y convertirla en una señal eléctrica. La Figura 3.4.2.7 muestra una imagen de un sensor infrarrojo.

Figura 3.4.2.7 - Sensor de movimiento

Sensores de proximidad: son sensores infrarrojos de proximidad que funcionan utilizando un sensor de luz específico para detectar una longitud de onda de luz seleccionada en el espectro infrarrojo. Cuando un objeto está cerca del sensor, la luz del LED rebota en el objeto y entra en el sensor de luz. Esto da como resultado un gran salto en la intensidad, respecto de un umbral. Dado que el sensor funciona buscando luz reflejada, es posible tener un sensor que pueda devolver el valor de la luz reflejada. Este tipo de sensor se puede usar para medir qué tan "brillante" es el objeto. Esto es útil para tareas como seguimiento de línea. La Figura 3.4.2.8 muestra una imagen de un sensor de proximidad.

Figura 3.4.2.8 - Sensor de proximidad

El modelo usado en Blokino es el sensor IR Sharp, una opción popular para muchos proyectos que requieren mediciones de distancia precisas. La interconexión con la mayoría de los microcontroladores es sencilla: la salida analógica individual se puede conectar a un convertidor analógico a digital para tomar medidas de distancia, o la salida se puede conectar a un comparador para la detección de umbral.

3.4.3 Componentes con módulos integrados

Pantalla LCD: es un tipo de pantalla alfanumérica gráfica basada en cristal líquido. La pantalla LCD tiene tamaño, forma y estilo variado; además contiene un módulo adaptador llamado I2C. Este módulo adaptador está basado en el controlador I2C PCF8574, el cual es un expansor de entradas y salidas digitales controlado por I2C. Con este módulo se reduce la cantidad de pines. La Figura 3.4.3.1 muestra imágenes de pantallas LCD.

Figura 3.4.3.1 - Pantallas LCD

Blokino usa este tipo de pantalla LCD porque con el módulo integrado se reduce la complejidad al armar los circuitos.

Joystick: es un módulo compuesto de 5 pines que indican 3-5 voltios, energía de tierra, dirección X, dirección Y y pulsación. Los pines de dirección X e Y son analógicos, con éstos se puede presionar el joystick hacia abajo o arriba. La Figura 3.4.3.2 muestra una imagen de un joystick similar a los utilizados en Blokino.

Figura 3.4.3.2 - Joystick

Matriz-Led: es una pantalla formada por múltiples LEDs en distribución rectangular. Existen distintos tamaños, siendo la pantalla más común la de forma cuadrada de 8×8 Leds. Es posible combinar varios módulos para formar una pantalla de mayor tamaño. En estas pantallas se pueden mostrar textos, dibujos o incluso animaciones, así como desplazar un texto (scroll).

Encender una matriz de LED directamente con Arduino requiere emplear una gran cantidad de pines, lo cual supondría un gran desperdicio de recursos. Por este motivo en Blokino se eligió una Matriz-LEDS con el módulo integrado MAX7219. Como se mencionó anteriormente, al incorporar módulos se reduce la cantidad de cables duponts en los circuitos. En este caso se redujo de 16 pines a 4 pines. La Figura 3.4.3.3 muestra una matriz-leds similar a la utilizada en Blokino.

Figura 3.4.3.3 - Matriz de Leds

Teclado MPR121: es un teclado que usa pads sensibles al tacto MPR121/12, para brindar un 'teclado' simple con una salida I2C. Este modelo tiene en los extremos agujeros de montaje que se pueden acoplar a una base sólida para poder tener una manipulación más cómoda del componente electrónico. La Figura 3.4.3.4 muestra una imagen de un teclado MPR121 similar al usado en Blokino.

Figura 3.4.3.4 - Teclado MPR121

3.4.4 Componentes de soporte

Diodos: son polarizados, esto significa que la dirección en la que se colocan en el circuito indica la dirección que tomará la electricidad. Se utilizan en aplicaciones que requieren que la electricidad fluya en una sola dirección. Al colocarlo en una determinada dirección permiten que la corriente los atraviese, por el contrario, colocados en la dirección contraria, la bloquean. Los diodos poseen dos lados, el ánodo y el cátodo. El lado del ánodo, por lo general, conecta con el punto de mayor energía del circuito. El lado del cátodo habitualmente se conecta al punto de energía más bajo, o a la toma de tierra. El cátodo suele marcarse con una banda en un lado del cuerpo del componente. La Figura 3.4.4.1 muestra una imagen de diodos estándares en la que es posible identificar en ánodo y el cátodo.

Figura 3.4.4.1 - Diodos estándar

Resistencias: son dispositivos que resisten el flujo de energía eléctrica en un circuito, cambiando el voltaje y la corriente. Los valores de la resistencia se miden en ohmios, representados por Ω y las bandas de colores nos indican su valor, aunque siempre podemos usar un polímetro para averiguarlo. La Figura 3.4.4.2 muestra imágenes de resistencias similares a las utilizadas en Blokino.

Figura 3.4.4.2 - Resistencias

Estos dispositivos son usados para evitar cortocircuito o sobrecarga dependiendo de la resistencia que se use en los circuitos.

Transistores: son dispositivos usados para controlar componentes de alta corriente o alto voltaje como los motores. Los transistores poseen tres clavijas, la primera se conecta a la toma de tierra, la segunda al componente que está siendo controlado y la tercera se conecta al Arduino. Cuando el componente recibe voltaje en la clavija conectada al Arduino, cierra el circuito entre la toma de tierra y el otro componente. La Figura 3.4.4.3 muestra una imagen de un transistor.

Figura 3.4.4.3 - Transistor

Se agregó este dispositivo para entregar una señal de salida en respuesta a una señal de entrada, dependiendo qué componentes están conectados entre los transistores.

Capítulo 4 - Desarrollo de la plataforma Blokino

La plataforma Blokino está compuesta por dos aplicaciones: una plataforma web y una aplicación de escritorio multiplataforma, ambas basadas en código fuente abierto y tecnologías JavaScript.

4.1 Estructura del Proyecto

La estructura del proyecto Blokino fue pensada de la siguiente manera:

- **Una aplicación web** hosteada en una máquina virtual de Digital Ocean («Digital Ocean», 2019). En esta web se encuentra la documentación y la sección para la descarga de Blokino (software).
- **Una aplicación de escritorio multiplataforma para Windows y Linux.** Con esta aplicación se pueden ejecutar programas de Blokino sobre placas Arduino.

Para poder albergar aplicaciones web en Digital Ocean se usan droplets. Los droplets son máquinas virtuales basadas en Linux que se ejecutan sobre hardware virtualizado. Cada droplet creado es un nuevo servidor que se puede usar, ya sea de forma independiente o como parte de una infraestructura más grande basada en la nube.
El droplet elegido para Blokino es una versión con Linux Mint con un entorno NodeJS con una configuración predeterminada. En la Figura 4.1.1 se muestra la estructura del droplet de la aplicación web de Blokino.

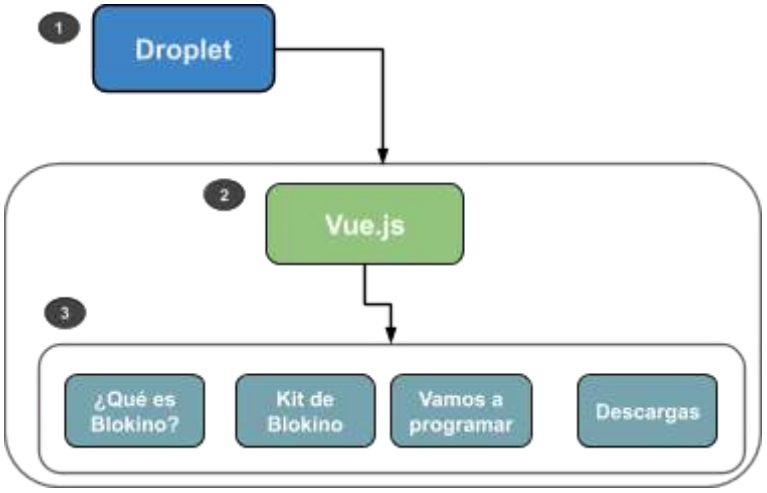


Figura 4.1.1 - Arquitectura de la aplicación web Blokino

El frontend de la aplicación web de Blokino está soportado por el framework Vue.js («Vue.js», 2014). Éste es un framework de JavaScript, de código fuente abierto, que permite construir interfaces de usuarios usando la metodología SPA (Single Page Application).
Las SPA son aplicaciones web alojadas en una sola página, con el propósito de dar una experiencia más fluida a los usuarios, similar a una aplicación de escritorio. En una SPA todos los códigos de HTML, JavaScript y CSS (Cascade Style Sheet) se cargan la primera vez que el usuario de la aplicación ingresa a la página. El resto de los recursos necesarios se cargan dinámicamente a medida que lo requiera la página y se van agregando, normalmente como respuesta de las acciones del usuario.

El creador de VueJS es Evan You, ex empleado de Google, y anteriormente desarrollador de Angular. Publicó el framework en el año 2014. Inicialmente fue pensado para ser una biblioteca personal, pero la comunidad hizo que el proyecto creciera a un ritmo impresionante, posicionándose actualmente como uno de los frameworks web más populares, junto a Angular y React.
Una de las características más importantes de VueJS es el concepto de componentes. Los componentes son elementos que encapsulan código reutilizable. Dentro de cada componente hay archivos html, estilos css y código JavaScript. Los componentes permiten desarrollar proyectos modularizados y simples de escalar.
El cliente web de Blokino está compuesta por 4 secciones principales:

- **¿Qué es Blokino?:** contiene una descripción de la herramienta y acceso al repositorio donde está el proyecto almacenado de manera pública.
- **Kit Blokino:** contiene una descripción de todos los componentes electrónicos con los cuales es posible usar Blokino.
- **Vamos a programar:** contiene la documentación de Blokino y comprende el siguiente contenido:
  - Armado de circuitos.
  - Configuración de las placas Arduino.
  - Descripción y modo de uso de los bloques funcionales.
  - Novedades de Blokino.
- **Descarga:** contiene las opciones de descarga de las distintas versiones de Blokino. Las versiones soportadas actualmente son Windows y Linux (versión Debian).

La Figura 4.1.2 muestra la arquitectura de la aplicación de escritorio de Blokino, en la que se identificaron los elementos fundamentales y sus funcionalidades. Más adelante, en este informe, se detallarán cada uno de estos elementos.

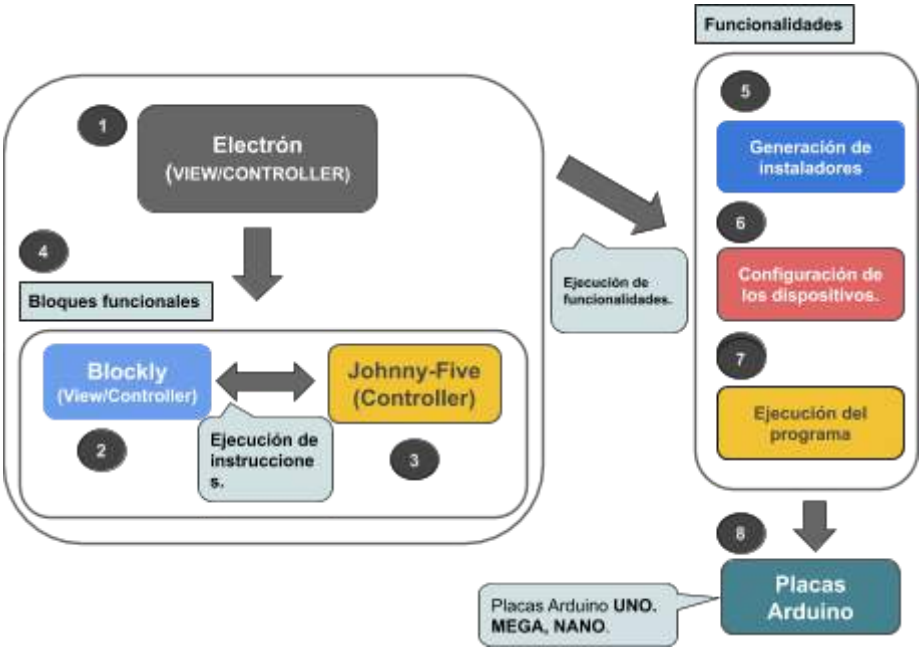


Figura 4.1.2 - Arquitectura de la aplicación de escritorio de Blokino

A continuación se describen brevemente cada uno de los elementos que conforman la aplicación de escritorio de Blokino:

- **Electron**: es el framework pri[5]ncipal para crear la plataforma de escritorio. («Electron - Build cross platform desktop apps», 2013).
- **Blockly**: es un lenguaje de programación visual compuesto por un sencillo conjunto de comandos. Contiene una estructura visual que se encarga de la construcción de los bloques visuales que contienen funcionalidad. («Blockly - A JavaScript library», 2012).
- **Johnny-Five**: es un framework open-source usado para la programación robótica usando JavaScript. («Johnny-Five», 2012).
- **Bloques funcionales**: son los bloques visuales usados al estilo “rompecabezas” para crear los programas Blokino. Es la fusión de la vista de bloques que provee Blockly y la funcionalidad sobre las placas de Johnny-Five.
- **Generación de instaladores**: es la funcionalidad que permite construir instaladores de Blokino para las plataformas Windows y Linux.
- **Configuración de dispositivos**: es la funcionalidad que permite configurar los dispositivos Arduino.
- **Ejecución del programa**: es la funcionalidad que permite ejecutar el código programado con Blokino, para ello se aplicó el concepto de hilos de ejecución.
- **Placas Arduino**: son los dispositivos físicos sobre los cuales Blokino puede operar.

4.2 JavaScript - ECMAScript 6

Como se mencionó previamente Blokino está desarrollado en su totalidad con JavaScript. Se trata de un lenguaje actualmente muy popular y usado mundialmente para desarrollar aplicaciones en distintos entornos, desde el desarrollo de servicios web hasta aplicaciones móviles y servidores. A continuación, los hitos más relevantes de su historia:

- En 1995, Brendan Eich crea un lenguaje llamado Mocha cuando pertenecía al grupo de desarrollo del navegador Netscape.
- En septiembre de ese mismo año lo renombra como LiveScript. Posteriormente pasa a llamarse JavaScript.
- Cuando Netscape es adquirida por Sun Microsystems, propietaria del lenguaje Java en esos años, es renombrado como JavaScript debido a una estrategia de marketing, aún cuando los lenguajes no están relacionados.
- En 1997 se crea un comité, TC39, para estandarizar JavaScript por la European Computer Manufacturers Association (ECMA). Se diseña el estándar del DOM[5] para evitar incompatibilidades entre navegadores. A partir de entonces los estándares de JavaScript se rigen por ECMAScript.
- En 1999 aparece la tercer versión del estándar ECMAScript
- En 2011 se estandarizó la versión 5 (ES5).
- En junio de 2013 se lanzó la primera versión de ECMAScript 6.
- En 2014 JavaScript fue premiado como el lenguaje con mayor crecimiento, actualmente es uno de los lenguajes más populares, sólo por debajo de los ancestrales C y Java. Sin duda por sus características se apodera cada día más de la web.

[5] DOM (Objeto de Documento) es una API para documentos HTML y XML, que define la estructura lógica de los documentos, el modo en que se accede y manipula estos documentos.

- A partir de junio del 2015 se estandarizó ECMAScript 6, y con esta estandarización se lograron más alcances a distintos entornos de desarrollo.
- Durante los últimos años han salido más versiones. Pero a diferencia de la versión ES6 del 2015, son cambios menores. La versión ES6 trajo cambios radicales con respecto a la sintaxis y semántica usada para programar con JavaScript.

Para el desarrollo de Blokino se eligió ECMAScript 6 o ES6. Los cambios y aportes que se usan de ES6 en la plataforma son los siguientes:

- **Arrow Function:** tiene una sintaxis más corta que la clásica "function". Este tipo de funciones siempre son anónimas. Estas funciones no están relacionadas con métodos y no pueden ser usadas como constructores.
- **Clases:** en esta versión se pueden crear clases, como se hace en los lenguajes basado en objetos.
- **This:** en ES6 cambió la semántica de la palabra clave **this,** hace referencia a todo el contexto de la aplicación. En versiones previas esta variable era usada para obtener variables que estaban en el alcance de la función.
- **let - const:** en la declaración de variables se puede definir su alcance en el contexto. En "let" hace referencia al contexto de la función, y con el "const" se pueden crear constantes que sólo se pueden leer y no modificarse a lo largo del código.
- **Template String:** con ES6 se puede interpolar Strings de una forma más sencilla. Con este aporte JavaScript introdujo las template string de forma narrativa. Estas cadenas literales de texto incrustadas en el código fuente permiten la interpolación mediante expresiones.
- **Destructing:** nueva forma de asignar valores a Arrays y a Objetos.
- **Value for default:** con esta novedad se pueden asignar valores por defecto a las variables que se pasan por parámetros en las funciones.
- **Modules:** con esta mejora, JavaScript se comienza a parecer a lenguajes de programación como Python o Ruby. Con los módulos se puede encapsular funcionalidades o características, hasta clases.

Además de las nuevas características que provee ECMAScript 6, hay otros motivos en la elección de la versión de JavaScript para implementar Blokino:

- Al tener estandarizado el código base, se pueden incorporar las nuevas características que se van estandarizando.
- Al estar actualizado, el código base se puede escalar. Este punto es importante dado que además de crear la plataforma, también se optó por aplicar buenas prácticas de programación. Cuando se aplican buenas prácticas de programación, el código y la estructura que se elija es importante, dado que el proyecto puede ser entendido tanto en lo sintáctico como en lo semántico y su funcionalidad. Además al ser un proyecto open-source, el código debe quedar escalable, legible y documentado.

4.3 Electron

Para que la aplicación de escritorio Blokino sea multiplataforma, se investigó alternativas a las usadas en otros lenguajes de programación para la implementación de aplicaciones de escritorio, como PyQt o Tkinter para Python o Swing para Java. En el caso de JavaScript, actualmente se tiene una amplio soporte para distintos tipos de desarrollo. Por ejemplo, se pueden desarrollar:

- Aplicaciones web con arquitecturas SPA, usando frameworks como Angular, Vue.js y React entre otros.
- Servidores usando Node.js, Express o Hapi.
- Aplicaciones móviles híbridas usando Ionic con Angular.
- Aplicaciones móviles nativas usando Nativescript o React Native.

Antes esto no era así, sino que se usaban lenguajes de programación específicos para cada área, como por ejemplo:

- Para las aplicaciones web, se usaba PHP en conjunto con JavaScript y Jquery.
- Para la construcción de servidores, se usaba Java o .Net.
- Para el desarrollo de aplicaciones móviles, se tenía un lenguaje específico para cada sistema operativo. Para Android se usaba Android/Java y para iOS se usaba el lenguaje Swift.

Actualmente estas opciones siguen disponibles y son usadas por una gran comunidad de programadores. Las comunidades de JavaScript de cada una de las opciones mencionadas fueron creciendo, llegando a ser opciones confiables y completas para el desarrollo.

Para la implementación de la aplicación de escritorio de la plataforma Blokino se analizaron y evaluaron dos opciones: Meteor y Electron.

- **Meteor**: es una plataforma de diseño open-source de aplicaciones multiplataforma y abarca toda la pila de desarrollo (full stack): cliente, servidor y base de datos. Además, con Meteor se pueden desarrollar aplicaciones web, móviles y de escritorio. De esta manera cubre el desarrollo backend y frontend en un solo entorno, pensado así para desarrollar aplicaciones con comunicación en tiempo real. Las tecnologías frontend como Blaze, React y Angular son usadas para el diseño de una aplicación Meteor. La Figura 4.3.1 muestra la arquitectura de las aplicaciones Meteor.

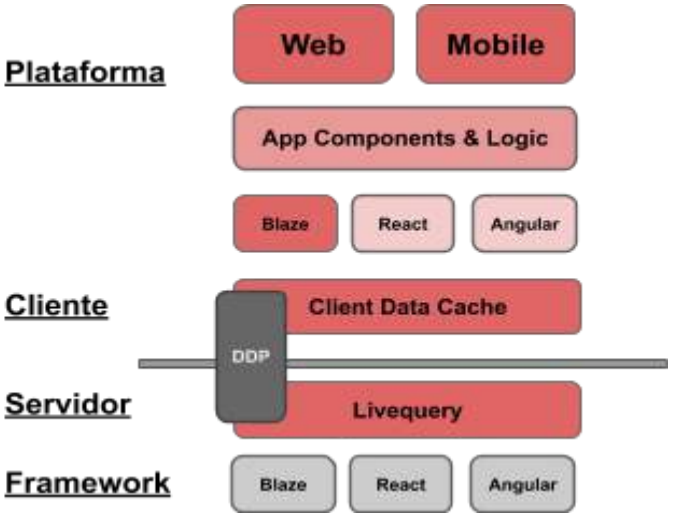


Figura 4.3.1 - Arquitectura de Meteor

Meteor usa la comunicación en tiempo real mediante un protocolo llamado Distributed Data Protocol (DDP), que es soportado por navegadores modernos que usan WebSockets o en navegadores antiguos que soportan el “long polling”. La comunicación navegador-servidor es transparente para el usuario final.

Actualmente existen desarrollos realizados con Meteor, ejemplo de ellos son: Doopoll.co, Telescope, Rocket.chat, Angular-Meteor, Dr. Mongo, etc, algunos son aplicaciones web y otras de escritorio. Meteor es una opción muy completa para el desarrollo de una aplicación fullstack u orientada a aplicaciones de tipo *enterprise*.
En el caso del desarrollo de Blokino**,** se necesita desarrollar una plataforma sin capas de desarrollo full stack como el propuesto por Meteor, por lo tanto se debió buscar otra[6] plataforma de construcción de aplicaciones que esté totalmente enfocada en la construcción de aplicaciones de escritorio. Esta búsqueda llevó al análisis de Electron.
Electron es una plataforma open-source creada por Cheng Zhao, actualmente desarrollada y mantenida por Github. Electron se usa para desarrollar aplicaciones de escritorio mediante tecnologías web (HTML,CSS y JavaScript).

4.3.1 Procesos

El funcionamiento central de Electron se basa en dos tipos de procesos, el proceso Main y los procesos Renderer. En la Figura 4.3.1.1 se muestra la estructura de una aplicación de Electron genérica.

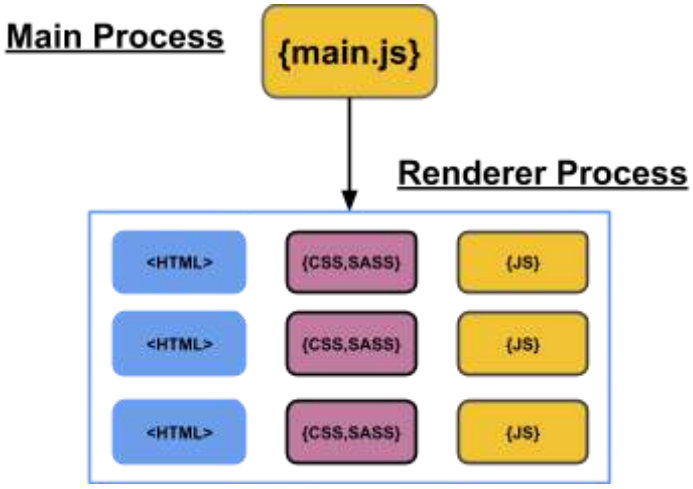


Figura 4.3.1.1 - Arquitectura de Electrón

El proceso Main tiene acceso a la mayoría de las funciones de la API de Electron, para la comunicación con el sistema operativo que hostea a la aplicación de escritorio. Este proceso es responsable de crear y administrar instancias de Browser Window[6] y varios eventos de aplicaciones. También puede registrar accesos directos a variables globales, menús y cuadros de diálogo nativos del sistema operativo, responder a eventos de actualización automática y más. El punto de entrada de las aplicaciones Electron apunta a un archivo JavaScript que se ejecuta en el proceso Main, como principal proceso del flujo de la aplicación. Un subconjunto de APIs de Electron está disponible en el proceso Main, así como un amplio conjuntos de módulos de Node.js. La documentación oficial de Electron indica como regla básica: “si un módulo está relacionado con la GUI o con un sistema de bajo nivel, entonces sólo debería estar disponible en el proceso principal". Esto es para evitar posibles problemas de pérdida de memoria. Este proceso es único en las aplicaciones Electron.

[6] Browser Windows es una API de Electrón que se encarga de crear y controlar las ventanas de la aplicación.

Dentro de Electron se usan WebViews, que es uno de los componentes más relevantes para el desarrollo de las aplicaciones Android, que constantemente se debe actualizar. Además este componente trabaja en conjunto con la tecnología Chrome que permite a las aplicaciones Android mostrar contenido web en las aplicaciones.
En el caso de Electron no se usa Chrome sino el proyecto Chromium[7]. Este proyecto es la versión de código abierto de Google Chrome, pero sin todos los codecs exclusivos y otros componentes esenciales de Google. Electron usa esta opción, debido a que la versión de Chromium usada está adaptada para la implementación de aplicaciones multiplataforma.
El proceso Renderer posee acceso a la API del DOM de la aplicación. La principal responsabilidad de este proceso es la de ejecutar la interfaz de usuario de su aplicación, o en otras palabras, una página web que es una instancia de webContents[8]. Una o más instancias de webContents pueden vivir en una sola ventana, dado que una sola ventana puede alojar múltiples webviews y cada webview es su propia instancia de webContents y proceso de representación.
La diferencia entre el proceso Renderer y el Main es que puede existir más de un proceso Renderer ejecutándose a la vez en la aplicación. Cada ventana de la aplicación cuenta con su proceso Main y procesos Renderer, donde cada proceso Renderer es independiente de otros, y similar a la arquitectura de Chromium, esto garantiza que si un proceso se cae, los restantes continúan su ejecución sin ser afectados. Tanto el proceso Renderer como el proceso Main conviven dentro del ecosistema de Node.js, aunque con distintos niveles de acceso a la API de Electron.

4.3.2 Funcionalidades de Electron usadas en Blokino

En Blokino se utilizaron las siguientes funcionalidades de Electron:

- Manejo de funciones nativas de la API: como configuraciones de menús, notificaciones, acceso al sistema de archivo y configuración base para la gestión del armado de instaladores según el sistema operativo que hostea a Blokino.
- Herramientas de debugging: se usó la consola Chrome Dev Tools de Chrome para el seguimiento de secciones de la aplicación como así también para hacer pruebas unitarias sobre datos temporales capturados en puntos de observación.

4.3.3 Aplicaciones que hacen uso de Electron

Entre las aplicaciones más populares que hacen uso de Electron se destacan:

- Slack: la aplicación por excelencia para la comunicación entre equipos. Se caracteriza por la complejidad de la misma, donde encontramos elementos como grupos privados, llamadas, etc.
- Visual Studio Code: creado por Microsoft, es un editor de código, muy usado por los desarrolladores frontend, sobre todo por la compatibilidad con TypeScript y su predicción en el tipeo que nos permite programar de una manera más fluida.
- Hyper: desarrollada por Zeit, es una terminal simple y minimalista .

[7] Chromium es un navegador web gratuito y de código abierto desarrollado por Google. En términos generales, es un proyecto paralelo de Google Chrome para obtener su código fuente.
[8] webContents es una API de Electrón que se encarga del renderizado y control de las páginas web.

- Discord: es una aplicación gratuita de VoIP que está disponible para sistemas operativos Windows, MacOS, Linux, Android e iOS. El software se lanzó inicialmente en marzo de 2015, y cuenta con más de 50 millones de usuarios registrados de todo el mundo.
- Atom: es un editor de código fuente abierto, gratuito y muy personalizable. Está escrito en HTML/CSS puro sobre Chromium, que además se compila con Electron. Atom cuenta con más de 7.000 paquetes diferentes, casi todos alojados en Github.
- WebTorrent Desktop: es el primer cliente de torrent que trabaja en el navegador y que está completamente escrito en JavaScript.

La versión que se eligió de Electron para Blokino es la 4.2.10. Se probaron versiones mayores de Electron, pero por compatibilidad con Johnny-Five no se pudo lograr un funcionamiento correcto, tanto sobre Linux como Windows. A futuro, si se logra la integración con versiones mayores de Electron, se podrían usar más funcionalidades.

4.4 Blockly

El entorno para generar los programas de Blokino está implementado mediante la librería de bloques visuales llamada Google Blockly. Esta librería está compuesta por un lenguaje de programación visual que permite encastrar bloques como si fueran piezas de un rompecabezas o un LEGO. Cada bloque representa un comando o un bloque de comandos en particular. La Figura 4.4.1 muestra un programa creado con Blockly.

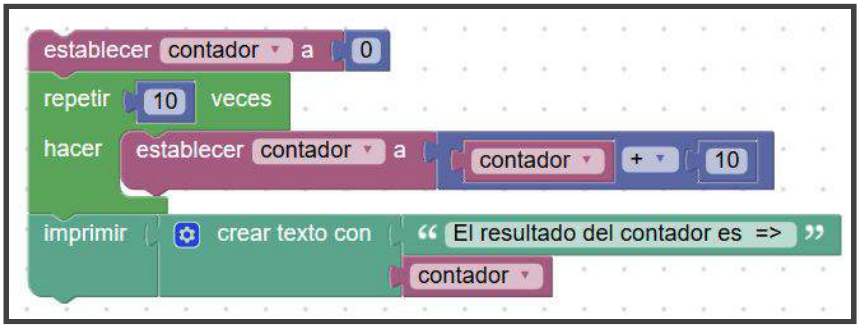


Figura 4.4.1 - Ejemplo de un programa de Blockly

Blockly tiene una identidad didáctica, con la cual es posible aprender los conceptos básicos de la programación. La estructura de esta librería se encuentra dividida en dos partes principales: una estructura visual (compuesta por XML y CSS) y otra estructura basada en JavaScript. En la Figura 4.4.2 se puede ver cómo está conformada la estructura de Google Blockly.

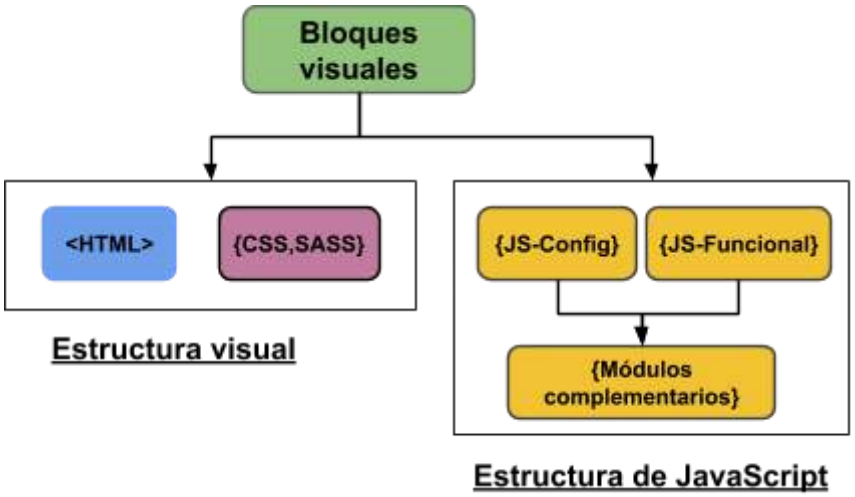


Figura 4.4.2 - Arquitectura de Google Blockly

Blockly además provee distintos lenguajes para crear los bloques, entre ellos están: JavaScript, Lua, Dart, Python y PHP.

4.4.1 Estructura basada en JavaScript

La estructura de Blockly basada en JavaScript está compuesta por tres bloques descriptivos llamados {JS-Config}, {JS-Funcional} y {Módulos complementarios}. Para la implementación de estos bloques descriptivos se usó JavaScript con la versión de ES6.
El bloque descriptivo {JS-Config} contiene la configuración base que debe tener cualquier bloque funcional de Blokino. La Figura 4.4.1.1 muestra cómo está compuesto el bloque funcional **Parpadeo (blink)** con código JavaScript.

```javascript
Blockly.Blocks["blink"] = {
    init: function() {
        this.appendDummyInput()
            .appendField("Parpadear")
            .appendField(new Blockly.FieldVariable("led"), "led")
            .appendField("Tiempo")
            .appendField(new Blockly.FieldNumber(0, 1, 100, 1), "time")
            .appendField(
                new Blockly.FieldDropdown([
                    ["milisegundos", "miliseconds"],
                    ["segundos", "seconds"],
                    ["minutos", "minutes"]
                ]),
                "period"
            );
        this.setInputsInline(false);
        this.setPreviousStatement(true, null);
        this.setNextStatement(true, null);
        this.setColour(30);
        this.setTooltip("Este bloque hace parpadear al LED.");
        this.setHelpUrl(url_documentation);
    }
}
```

Figura 4.4.1.1 - Código en JavaScript que define el bloque funcional de parpadeo de un LED de Blokino usando Blockly

Como se puede observar en la Figura 4.4.1.1, la configuración contiene los siguientes atributos claves:

- Nombre del bloque visual: en este ejemplo es “blink”.
- Valores de entrada: son valores similares a los que se usan en casi cualquier lenguaje de programación estructurado o funcional, como así también representan algunos componentes usados en lenguajes de programación de marcas como HTML:
    - Texto que simboliza a los String
    - Textos que pueden modificarse.
    - Valores numéricos.
    - Desplegables (dropdowns).
    - Casillas de verificación (checkbox).
    - Selector de colores.
    - Manejador de instancia de variables.
    - Control y configuración de imágenes.

En el código de la Figura 4.4.1.1 los valores de entrada son usados para establecer el LED que se quiere hacer parpadear, el tiempo que va a parpadear y el tipo de unidad de tiempo.

- Tipos de datos: entre los tipos de datos que maneja Google Blockly se encuentran los convencionales de cualquier lenguaje de programación estructurado o funcional:
    - any of: puede albergar varios tipos de datos distintos.
    - any: indica que el valor de entrada asignado puede recibir cualquier tipo de datos.
    - Boolean: indica que el valor de entrada asignado puede recibir tipo de datos booleanos.
    - Number: indica que el valor de entrada asignado puede recibir tipo de datos numéricos.
    - String: indica que el valor de entrada asignado puede recibir una cadena de caracteres.
    - Array: indica que el valor de entrada asignado puede recibir estructuras de datos heterogéneas. Estas estructuras de datos son conocidas como arreglo o listas de datos de distintos tipos.

En código de la Figura 4.4.1.1 los valores de los tipos de datos usados son:

- Un array para seleccionar el LED que se quiere hacer parpadear.
- Un tipo de dato numérico, para indicar el tiempo de parpadeo.
- Un array de los tipos de tiempo, que en este caso pueden ser milisegundos, segundos o minutos.
- Colour: este tipo de datos es útil para darle color a los bloques visuales. También puede generar código hexadecimal o colores primarios para usar durante la generación de los programas.

En código de la Figura 4.4.1.1 los valores Colour definen el color del bloque funcional de parpadeo (Blink). En este caso se usa un código de color 30.

El bloque descriptivo de tipo JS-Functional contiene la funcionalidad y lógica basada en código Johnny-Five. Para integrar el código se usó Template Strings (plantillas literales) debido a que la integración no se podía realizar de la manera convencional. El concepto de Template String es parte de las mejoras que incorporó ES6 y permite integrar código de manera sencilla. El manejo de plantillas es aplicable a la gran mayoría de los lenguajes de programación. Por ejemplo en Java se dispone de Thymeleaf, en .NET de

Razor y en PHP de Smarty. JavaScript también dispone de su propio universo de plantillas.
En la Figura 4.4.1.2 se muestra un fragmento de código de Blokino que contiene la funcionalidad “stop_led”.

```
1 Blockly.JavaScript["stop_led"] = block ⇒ {
2     let led = Blockly.JavaScript.variableDB_.getName(
3         block.getFieldValue("led"),
4         Blockly.Variables.NAME_TYPE
5     );
6
7     let code = `${led}.stop();`;
8     return code;
9 };
```

Figura 4.4.1.2 - Implementación del bloque funcional “stop_led”, usando template string

En la Figura 4.4.1.2. desde la línea 2 hasta la 5 se obtiene el LED al que se la quiere aplicar la funcionalidad “detener un led”. En la línea 7 se guarda en una variable la instrucción de “detener un led”, que luego será interpretado por Google Blockly cuando se use el bloque funcional en los programas de Blokino. Asimismo se muestra cómo se implementó la funcionalidad en conjunto con template string.

Dentro de la estructura de JavaScript se puede agregar lo que se conoce como módulos complementarios. Como se mencionó anteriormente los módulos son incorporaciones de ES6 que permiten encapsular funcionalidades. Dentro de la plataforma Blokino se implementaron los siguientes módulos:

- Módulos de ruteo: contiene las rutas relativas de las secciones de Blokino.
- Módulos para los desafíos: Blokino contiene una sección de desafíos que representan las consignas que las y los estudiantes deberán resolver. Al existir varios desafíos, se declaró un módulo independiente con toda la funcionalidad y lógica que requiere el desafío para poder ser usado.
- Módulos core: contienen funcionalidad general usada por toda la aplicación. En Blokino se implementaron tres módulos core:
    - Los módulos de funcionalidades útiles, usados por toda la aplicación. Están compuestos por funciones para el control de ventanas de tipo modal y la estructura de la plataforma.
    - El módulo core de los dispositivos Arduino, usado para el control, reconocimiento y preparación de los dispositivos que se conectan a la computadora.
    - El módulo core para controlar las instancias de ejecución, compuesto por funciones para la gestión de hilos de ejecución de los programas de Blokino.
- Módulos variados: dentro de la plataforma se desarrollaron módulos útiles para dar formato, obtener información de la arquitectura, comunicación con la API de Electron y otras funcionalidades usadas en toda la plataforma.

4.4.2 Estructura visual de Blokino

La estructura visual está desarrollada usando código XML y CSS, que es autogenerado mediante código JavaScript. La estructura JavaScript se traduce en bloques funcionales que conforman la estructura visual. Para poder usarlos se los debe inyectar en los archivos HTML de la aplicación, que en este caso es la plataforma Blokino.
Para poder inyectar los bloques de Google Blockly nativos o los que son definidos en Blokino, se los debe agregar a la caja de herramientas, conocida como “toolbox”.
El toolbox en Blockly, hace referencia a la sección dónde están definidos los bloques funcionales de Blokino. En la Figura 4.4.2.1, se muestra el toolbox y sus bloques funcionales definidos.

Figura 4.4.2.1 - Los bloques funcionales del toolbox

A modo de ejemplo, la configuración que compone Blockly está definida en el módulo de configuración. Este módulo de configuración puede asociarse a un toolbox identificado como “blokino-workspace”, que corresponde al espacio de trabajo en Blokino. En la siguiente Figura 4.4.2.2 se muestra cómo se configura el workspace de Blockly.

```
1 Blockly.inject("blokino-workspace", Config.blockly(Blockly, typeToolBar));
```

Figura 4.4.2.2 - Configuración del toolbox identificado como “blokino-workspace”

Google Blockly permite definir la cantidad de toolbox que se van a usar. El identificador asociado al momento de inyectar un bloque es útil para poder reconocerlo en la estructura HTML de la aplicación.
Además también se configuró el estilo del workspace de Blockly. En la siguiente figura, se pueden ver los ajustes de colores y saturación de workspace. En la Figura 4.4.2.3, muestra las propiedades de estilo definidas para el workspace de Blockly.

```
blockly: (Blockly, typeToolBar) => {
    Blockly.FieldColour.COLOURS = COLOURS;
    Blockly.FieldColour.COLUMNS = 3;
    Blockly.HSV_SATURATION = 0.5;
    Blockly.HSV_VALUE = 0.7;
    return settings.selectToolBox(typeToolBar);
}
```

Figura 4.4.2.3 - Configuración del toolbox de Google Blockly

Las configuraciones que se modificaron en Blockly son las siguientes:

- El color de las filas del toolbox.
- El color de las columnas del toolbox.
- La saturación que tomarán los bloques de manera general.
- Mostrar o no iconos como la papelera de Blockly.
- Iconos de zoom para el toolbox.
- Código XML de los bloques que se van a cargar en el navbar del menú de bloques funcionales del toolbox.

Todo lo referente a la estructura visual de Blockly está representado mediante el metalenguaje XML. Una de las desventajas de los archivos en formato XML es el control de su código fuente, por este motivo, para la categorización de los bloques de los menús de Blokino se optó por el encapsulamiento de los menús mediante los template string de JavaScript. Esto permitió manipular el contenido del código XML de una manera más cómoda y adaptable. En la Figura 4.4.2.4 se muestra cómo se encapsula la configuración de un menú del toolbox de Blockly usando template string. Con el uso de template string se puede indentar el código XML, manteniendo la legibilidad aún cuando los menús son extensos y complejos.

```
let menuFunctions = {
  menu: () => {
      return {
          test: {
              code:
                  `<xml>
                      <category name="Bloques" colour="270">
                          <category name="Variables" custom="VARIABLE" colour="210"/>
                          <sep gap="32"></sep>
                      </category>
                  </xml>`
          }
      }
  }
}
module.exports = menuFunctions;
```

Figura 4.4.2.4 - XML embebido en template string de ES6

Para establecer la apariencia de cada bloque funcional del menú se pueden usar las directivas propias de Google Blockly, como “colour” o el tag “sep”. Otra opción es usar el lenguaje de hojas de estilo CSS, definiendo las clases y/o selectores que se quieran aplicar, y de esta manera personalizar los bloques. Cada uno de los menús en Blokino es

un conjunto de bloques funcionales encapsulados en módulos independientes con sus respectivas dependencias. Con el uso de template string, además de poder indentar el código XML de los menús, también se puede encapsular cada menú en un módulo independiente.

4.4.3 Licencia

Google Blockly es software de código abierto bajo la licencia Apache 2.0, alojado en un repositorio de software libre. Está compuesto por Google Code, que permite manipular componentes de control y lógica para crear programas, que pueden exportarse a lenguajes como JavaScript, Python, XML y Dart.
Google Blockly es usado actualmente en distintos proyectos, incluyendo herramientas educativas. La idea de Google es que los desarrolladores puedan usar Blockly en proyectos personales y lograr aplicaciones del estilo de MIT App Inventor o Scratch, ambas orientadas al aprendizaje de programación de niñas, niños y jóvenes. Entre las aplicaciones que hacen uso de Google Blockly tenemos:

- Puzzle: es un juego que permite aprender cómo funcionan los bloques. Con esta opción de Blockly se pueden resolver rompecabezas usando bloques descriptivos.
- Maze: es un juego donde hay que encontrar la salida a un laberinto. Los bloques funcionales son usados para generar salidas.
- Gráfico: usa la calculadora gráfica de Blockly. Con esta funcionalidad se pueden realizar cálculos matemáticos para generar gráficos.
- Código: es una aplicación que permite exportar un programa Blockly en JavaScript, Python o XML.

De la misma manera que otros proyectos de Google, Google Blockly tiene como respaldo a la comunidad de desarrolladores de Google Developers. Blokino tiene integrada la última versión de Google Blockly para poder hacer uso de las mejoras y correcciones que va agregando la comunidad de Google Developers constantemente.

4.5 Johnny-Five

Johnny-Five es un proyecto de software relacionado a robótica, de código abierto, escrito en JavaScript y creado por Rick Waldron (Rick Waldron, 2012). Este proyecto fue publicado mediante la consultora de plataformas web Bocoup (Bocoup - Web Platform Consulting, 2012) en el 2012. Luego de la publicación, se fue generando una comunidad de desarrolladores de software e ingenieros de hardware apasionados por la robótica y JavaScript, que dan mantenimiento al proyecto sumando mejoras, correcciones o nuevas funcionalidades. Los programas de Blokino, están compuestos por componentes escritos con Johnny-Five.
Así como Johnny-Five, existen otras alternativas que usan JavaScript u otros lenguajes como GO o Python. Entre las alternativas se encuentran: Gobot, Cylon, Node-Red, y node serialport.
Se evaluó Johnny-Five con el propósito de usarlo como librería de control de los componentes electrónicos de Blokino. La evaluación consistió en POCs[9] con las distintas opciones de JavaScript Robotics, haciendo foco en la estabilidad y performance al

[9] POCs (Pruebas de concepto) es una implementación, a menudo resumida o incompleta, de un método o de una idea, realizada con el propósito de verificar que el concepto o teoría en cuestión es susceptible de ser explotada de una manera útil

ejecutar la plataforma Blokino en las computadoras. Estas POCs consistieron en los siguientes puntos:

- Instalar Blokino en computadoras con arquitecturas de 64 bits: se seleccionó esta arquitectura, debido a la compatibilidad de Electron con Johnny-Five en conjunto con las dependencias de NPM.
- Instalar Blokino en computadoras con sistemas operativos Linux y Windows. En el caso de MacOS, las pruebas no fueron exhaustivas como las otras dos plataformas, debido a que no se disponía del hardware.
- Ejecutar programas de complejidad incremental, desde cómo encender un led, hasta el uso en conjunto de distintos componentes electrónicos.

El cuadro comparativo que se presenta a continuación, muestra los resultados obtenidos con las opciones evaluadas.

Gobot, Cylon, Node-Red y Node-Serialport	**Johnny-Five**
● Componentes electrónicos: sin soporte para algunos. ● Placas Arduino: soporte solamente para la placa Arduino UNO. ● Crear instalador: Conflicto de librerías multiplataforma, para la generación de instaladores.	● Soporte para placas Arduino UNO, MEGA y NANO. ● Soporte para una amplia gama de componentes electrónicos, entre ellos están: LEDs, Servomotores, Matrix-LCD, Motores y sensores de movimiento entre otros. ● Johnny-Five se pudo integrar con Electrón, pero para poder lograr un mejor rendimiento, se debieron desarrollar módulos de configuración.

En base a estos resultados se decidió usar Johnny-Five como la librería de robótica, para controlar los componentes electrónicos y las placas Arduino.

4.6 Bloques Funcionales

Para poder encapsular el código Johnny-five generado mediante template string se usó Blockly, esté código encapsulado es funcional en su totalidad, es por este motivo que recibe el nombre de bloques funcionales.

La programación basada en elementos visuales ha ido tomando fuerza con el pasar de los años. El objetivo de la programación visual, es mejorar la comprensión de los programas y simplificar la programación en sí. El paradigma de programación visual promueve la construcción de programas simples o complejos sin escribir una línea de código. A modo de síntesis se presentan las principales características de los lenguajes visuales:

- Usa una representación visual, tal como gráficos, dibujos, animaciones o iconos, parcial o completamente.
- Manipula información visual o soporta interacción visual, o permite programar con expresiones visuales.

Con este tipo de lenguajes, se generaron técnicas visuales para expresar relaciones o transformaciones en la información, y no solamente orientado a funcionalidad.

El génesis de los paradigmas de programación visual ocurrió en 1975 con la publicación de David Canfield Smith “Pygmalion: A Creative Programming Environment". Con Pygmalion se incorporó un paradigma de programación basado en iconos en el cual los objetos creados podían ser modificados, y conectados juntos, para realizar cómputos y funcionalidades. Los lenguajes visuales modernos emplean un acercamiento basado en iconos, similar al planteado por Smith.
Con Blockly se pueden implementar bloques funcionales que incluyen la parte visual y funcional. En la Figura 4.6.1 se muestra la estructura que se implementó para implementar los bloques funcionales.

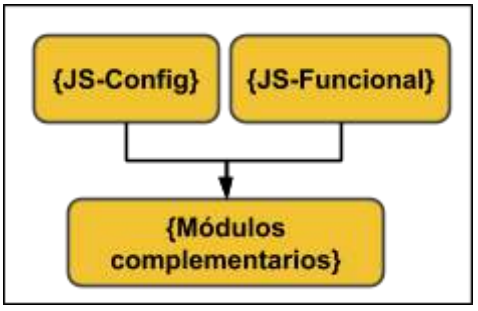


Figura 4.6.1 - Estructura de bloques funcionales

Como se mencionó anteriormente, los bloques funcionales están compuestos por 3 bloques descriptivos, que son los siguientes:

- {JS-Config}: contiene la configuración de los bloques funcionales.
- {JS-Funcional}: contiene toda la lógica de los bloques funcionales.
- {Módulos complementarios}: contiene todos los módulos de npm, usados para complementar los bloques funcionales. Estos módulos son variados, ya sea para sumar funcionalidades o para dar soporte.

Para que Blokino reconozca un programa como válido se debe seguir un orden lógico de acoplamiento. Este orden de acoplamiento, genera una lógica funcional y debe seguir el siguiente orden:

- Primero se deben declarar los bloques que representan a las variables.
- Para poder usar los bloques funcionales se deben asignar a una variable.
- Para poder ejecutar las funcionalidades se deben usar los bloques funcionales de cada componente electrónico.
- Siempre se debe respetar el siguiente orden: declaración de variables, asignación de componentes y por último ejecución de los bloques funcionales de cada componente electrónico.

El gráfico de la Figura 4.6.2 describe cómo se deben crear los programas en Blokino.

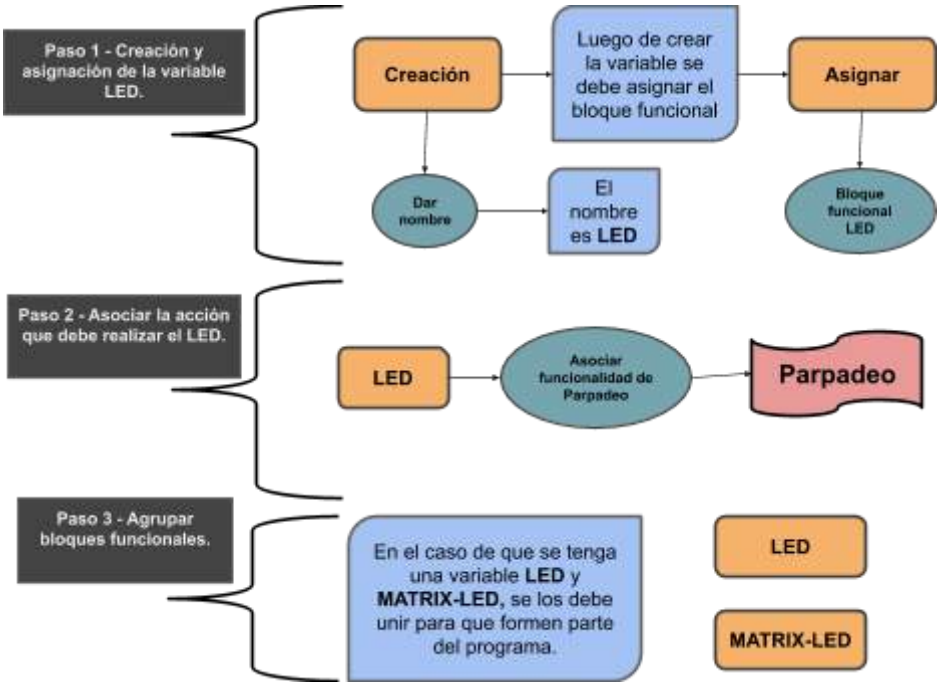


Figura 4.6.2 - Ejemplo de estructura de un programa en Blokino.

Para eso se establecieron los siguientes pasos:

- Paso 1: en este paso se crea una variable llamada LED. Luego de crearla, se la debe asignar el bloque funcional LED, en conjunto con sus atributos que contiene este bloque funcional.
- Paso 2: una vez creada y configurada la variable LED, se debe asociar la funcionalidad. En este caso se debe usar el bloque funcional “parpadear”, para que el LED pueda efectivamente parpadear.
- Paso 3: en este paso se agrupan todos los bloques funcionales, generando un bloque de código que ya puede ser ejecutado en las placas Arduino.

4.6.1 Crear variables

Para poder usar los bloques funcionales dentro de los programas de Blokino, se los debe guardar en espacios de memoria llamados variables. Dentro de las variables se pueden no sólo guardar los bloques funcionales, sino también tipos de datos simples o compuestos y hasta funciones. Para poder crear variables en la plataforma, se debe usar el opción de menú llamada “Crear Variables”. La opción “Crear variable”, desplegará una ventana de tipo modal que permitirá el ingreso del nombre que se asociará a la variable.
Luego de ingresar el nombre de la variable se debe realizar una validación basándose en nomenclaturas de JavaScript:

- Son sensibles a mayúsculas y minúsculas.
- Deben empezar con una letra o el carácter underscore (‘_’).
- No puede ser una palabra reservada dentro de JavaScript:
  - var, const, super, function, return, class.
  - case, catch, break.
  - export, extends
  - if, else, for, switch, while, do.
- No puede tener como primer carácter a un carácter especial, como: $ o %.

Al realizar las validaciones pueden ocurrir dos escenarios de error. El primero se presenta cuando no se cumple con la nomenclatura Javascript. En este caso, se mostrará una ventana modal con el mensaje de error, como se muestra en la siguiente Figura 4.6.1.1.

Figura 4.6.1.1 - Ventana modal mostrando un error por no cumplir con la nomenclatura Javascript

El segundo escenario de error se puede presentar cuando se creó la variable y se quiere crear otra con el mismo nombre, en este caso se muestra una ventana modal con el mensaje de error como se observa en la Figura 4.6.1.2

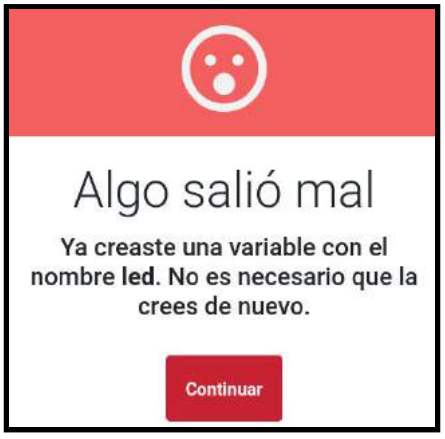


Figura 4.6.1.2 - Ventana modal mostrando un error de variable duplicada

Nuevamente se debe clickear en el bloque de “Crear variable”, esto mostrará la ventana modal para editar el nombre de la variable. Esta vez, con el nombre de variable ingresado válido, se la puede ver en el listado de variables junto con sus bloques relacionados. En la Figura 4.6.1.3, se muestra la sección de las variables creadas.

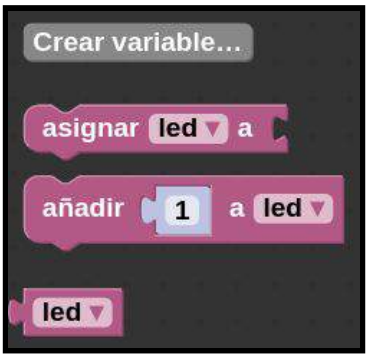


Figura 4.6.1.3 - Variables creadas y disponibles con sus bloques relacionados

A continuación se describen los bloques de la Figura 4.6.1.3:

- El primer bloque hace referencia al bloque que se usa para desplegar el modal, para crear variables.
- El segundo bloque se usa para asignar a la variable **led**, algún tipo de datos o bloque funcional.
- El tercer bloque es otra manera de asignación de datos a la variable **led.**
- El último bloque hace referencia a la variable, internamente tiene la dirección de memoria asociada a la variable. Con este bloque se puede asignar a otra variable o algún tipo de estructura de datos.

4.6.2 Tipos de datos

Luego de crear las variables, se les debe asignar tipos de datos como los ordinarios usados en la mayoría de los lenguajes de programación (numérico, cadena de caracteres, listas homogéneas o tipos booleanos) o los bloques funcionales de los componentes electrónicos. Cada uno de estos componentes electrónicos tiene datos de entrada, que son esenciales para la inicialización del bloque funcional. Los valores de entrada tienen un tipo de dato, en general son valores numéricos y caracteres. Los tipos de datos que soporta la plataforma son:

- Numéricos: usados para indicar unidades de cantidad o números de pin asociados a los componentes electrónicos conectados a la placa electrónica de Arduino.
- Cadena de caracteres: usados para definir textos, frases o representar entradas de pines de Arduino.
- Listas: usadas para agrupar tipos de datos simples o bloques funcionales, pero siempre de un sólo tipo de dato.
- Booleanos: usados para representar los estados verdadero o falso.

En el toolbox de Blokino existe una sección “Tipos de datos” donde se encuentran disponibles los tipos de datos mencionados anteriormente. Luego, cada tipo de dato tendrá asociado un conjunto de bloques. En la Figura 4.6.2.1 se puede ver está sección del toolbox de Blokino.

Figura 4.6.2.1 - Listado de bloques de tipos de datos

A modo de ejemplo, el tipo de dato “Número”, tiene relacionado bloques que representan operaciones aritméticas, como se puede observar en la Figura 4.6.2.2.

Figura 4.6.2.2 - Listado de bloques asociado al tipo de dato “Número”

Del mismo modo, el bloque que representa el tipo de dato “Textos”, tiene relacionado un conjunto de bloques que permiten operar sobre texto: modificar, verificar contenido, transformar en mayúsculas, etc, tal como se observa en la Figura 4.6.2.3.

Figura 4.6.2.3 - Listado de bloques asociados al bloque “Texto”

En la Figura 4.6.2.4 se muesta un conjunto de bloques asociados que permiten la creación, asignación valores y verificación del tipo de dato “Lista”.

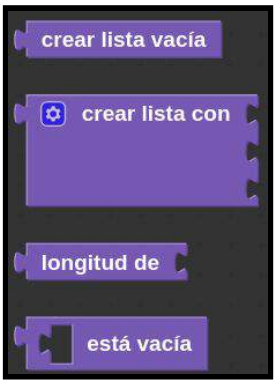


Figura 4.6.2.4 - Listado de bloques asociados al bloque “Lista”

Como se mencionó previamente, los bloques funcionales son usados para representar componentes electrónicos digitales y analógicos, por tanto, la creación e inicialización será diferente según sea el caso y se podrán asociar los bloques que representan a los tipos de datos según corresponda.
A los componentes electrónicos se los puede clasificar según sus tipos de datos de entrada en:

- Digitales: este tipo de dato de entrada son señales fijas de datos. En la siguiente imagen, se ve como se usa el tipo de dato numérico, para crear un LED asociado al pin número 12.

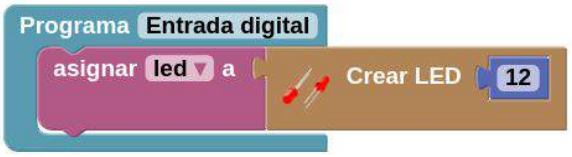


Figura 4.6.2.5 - Listado de bloques de tipos de datos Lista

- Analógicos: este tipo de dato de entrada son señales variantes. Los pines de entrada son representados por tipos de datos alfanuméricos. En el siguiente ejemplo de la Figura 4.6.2.6, se crea un **JOYSTICK** con sus pines de entrada representados por bloques de tipo texto.

Figura 4.6.2.6 - Creación del componente analógico JOYSTICK.

4.6.3 Estructuras de control

Durante el desarrollo del programa, se deben tomar decisiones, acciones de control o de verificación mediante condiciones. Para poder representarlas se deben usar estructuras de control. Las estructuras de control que normalmente aparecen en cualquier lenguaje de programación, también tienen su representación mediante bloques en Blokino:

- Condiciones lógicas: que retorna un valor de verdadero o falso.

- Decisiones: son estructuras de selección que hacen uso de las condiciones, y en función del valor retornado eligen un flujo de ejecución. En Blokino están representadas por el bloque “si”.
- Bucle: es una sentencia que ejecuta un bloque de código, hasta que se cumpla una condición y pueden ser:
  - Repetitivos
  - Iterativos

En Blokino está representado por el bloque “Repetir” y sobre el mismo se indica el tipo de bucle. En la Figura 4.6.3.1 se puede obsevar la clasificación de los bloques mencionados anteriormente.

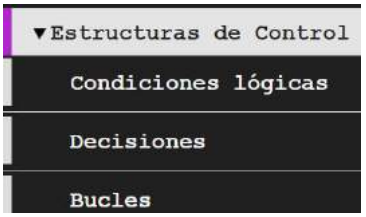


Figura 4.6.3.1 - Listado de bloques de control

La Figura 4.6.3.2 muestra un ejemplo de uso del bloque de decisión, y cómo se pueden agrupar más bloques funcionales dependiendo del resultado de la evaluación del condicional:

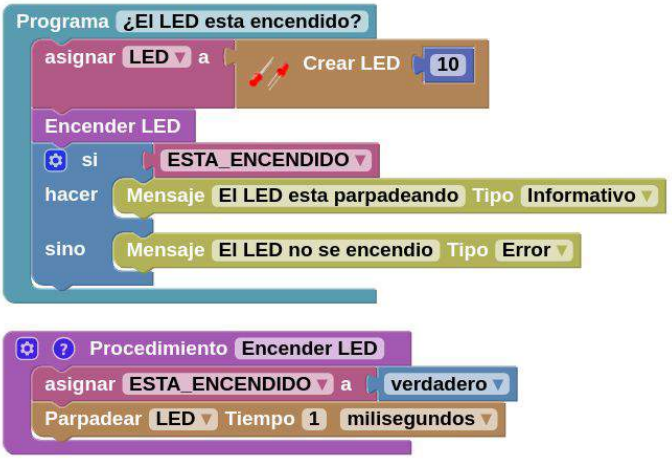


Figura 4.6.3.2 - Ejemplo de cómo usar la estructura de control de decisión

- Se crea un LED y es asignado a una variable.
- Se crea una variable de tipo booleana, para controlar si un LED está encendido o no.
- Luego, en un procedimiento se modifica la variable booleana y el estado del LED.
- Por último se usa el bloque de control decisión, para poder validar si el LED está encendido.
  - Si el LED está encendido, se envía un mensaje informativo a la consola de Blokino.
  - Si el LED no está encendido, se envía un mensaje de error a la consola de Blokino.

Otro de los tipos de bloque control es el bloque de control de bucle, que se basa en controlar una iteración hasta que se cumpla una condición, mientras esto no suceda se

ejecuta constantemente un conjunto de bloques funcionales. A modo de ejemplo en la Figura 4.6.3.3 se realiza la siguiente funcionalidad:

- Se crea una variable para contar la cantidad de movimientos.
- Se crea una variable de tipo texto, para controlar el tipo de movimiento que se va hacer con el joystick.
- Se crea un Joystick, indicando sus correspondientes pines.
- Se realiza un control mediante el bucle de control “repetir”, que controla los movimientos que se realizan con el JOYSTICK.
    - Mientras el movimiento del JOYSTICK no sea para “ARRIBA”, se contabilizan los movimientos.

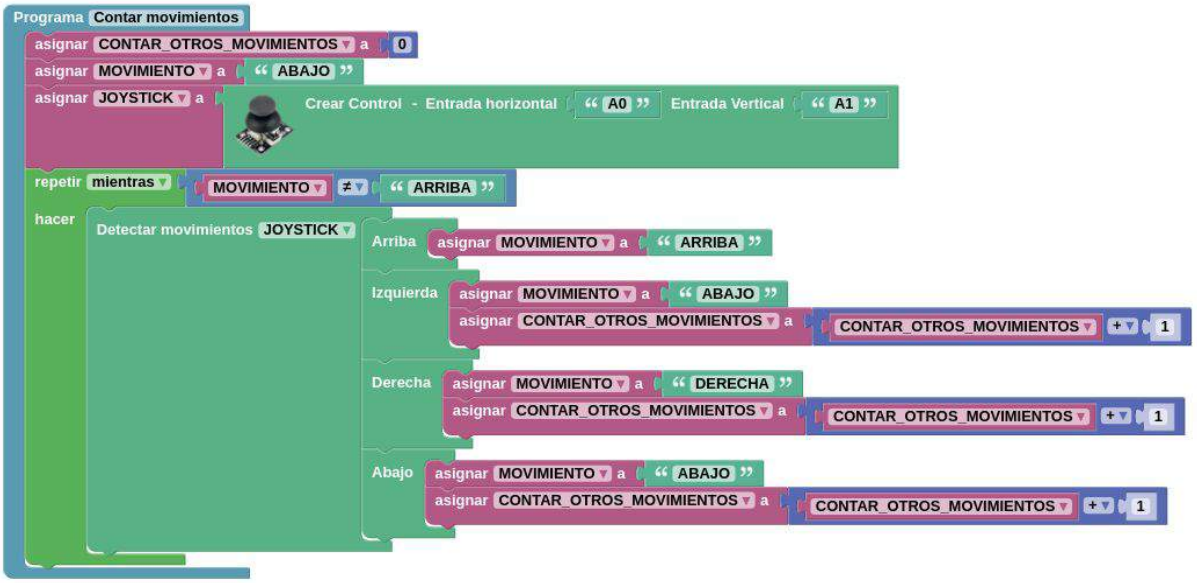


Figura 4.6.3.3 - Bloque condición de tipo Bucle

Otro bloque de control de tipo bucle se puede observar en la Figura 4.6.3.4, donde se realiza lo siguiente:

- Mediante un tipo de Lista, se crea un listado de LEDs con sus respectivos pines digitales.
- Se crea un procedimiento que recorre, usando el bloque de control, el listado.
- A medida que se va recorriendo el listado, se hace parpadear a cada uno de los LEDs, en un tiempo medido en milisegundos.

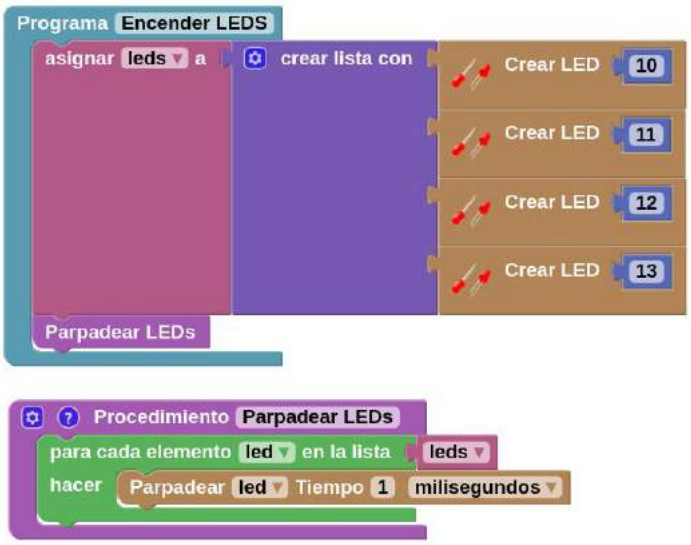


Figura 4.6.3.4 - Bloque condición de tipo Bucle

Con este tipo de bloques de estructuras de control, se pueden agrupar funcionalidades para un conjuntos de variables, logrando una reducción de código y complejidad.

4.6.4 Procedimientos

Los procedimientos son uno de los bloques principales de Blokino, dado que encapsulan bloques de código, que pueden ser reutilizables. Los procedimientos se encuentran en el toolbox de Blokino como una sección. En la Figura 4.6.4.1 se muestran las subsecciones asociadas a esta sección.

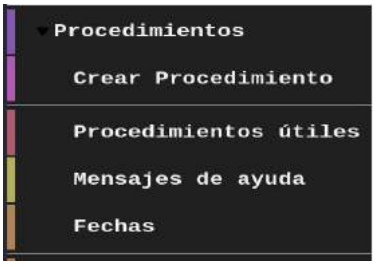


Figura 4.6.4.1 - Los bloques de la sección de procedimientos

- Crear Procedimientos: es una subsección de los procedimientos, donde se pueden administrar los procedimientos que se crearon previamente, como así también el modo en el que se lo invoca dentro del programa.
- Procedimientos útiles: dentro de esta subsección se encuentran procedimientos predefinidos, como por ejemplo procedimientos con temporizador.
- Mensajes de ayuda: en esta subsección se encuentran bloques que permiten visualizar mensajes de ayuda en la consola de Blokino. Estos bloques reciben como argumento texto literal o el resultado de la concatenación de variables definidas.
- Fechas: estos bloques sirven para obtener la fecha y hora de la zona horaria donde se esté usando la plataforma.

Dentro de los bloques de procedimientos personalizados, se encuentran dos tipos de procedimientos:

- Procedimientos sin argumentos: estos procedimientos sólo tienen un nombre y un bloque de código, como se puede observar en la Figura 4.6.4.2.

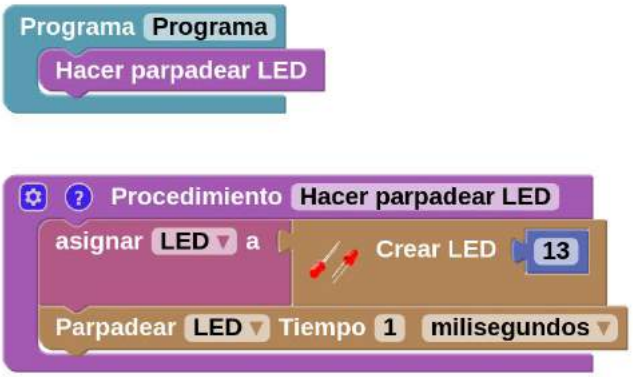


Figura 4.6.4.2 - Procedimiento “Hacer parpadear LED”, que crea un led y lo hace parpadear

- Procedimientos con argumentos: este tipo de procedimiento recibe, a través de sus argumentos, valores que se asocian en el momento de la invocación del procedimiento. En la Figura 4.6.4.3, se muestra la definición del argumento con la palabra clave “con” y cómo se asocian los parámetros a los procedimientos.

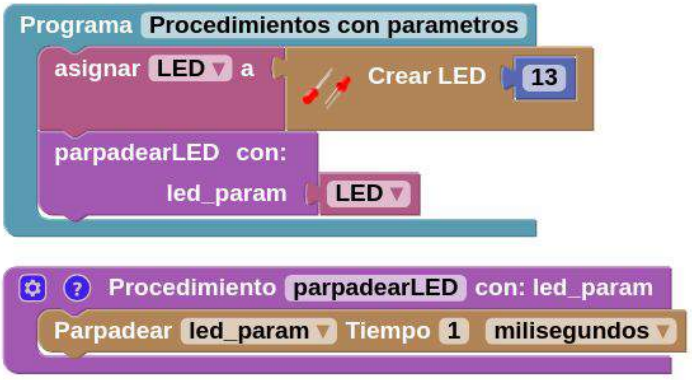


Figura 4.6.4.3 - Procedimiento con parámetro

Luego de creado el procedimiento aparecerá disponible en el toolbox de Blokino. En la Figura 4.6.4.4, se muestra el bloque funcional que se genera cuando se crea un procedimiento.

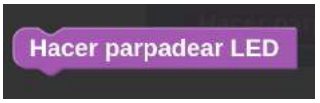


Figura 4.6.4.4 - Bloques complementarios de los procedimientos

Dentro de los procedimientos, se pueden agregar comentarios informativos para dejar documentados los procedimientos, como se muestra en la Figura 4.6.4.5.

Figura 4.6.4.5 - Procedimiento con comentarios

La Figura 4.6.4.6 muestra el código JavaScript de un procedimiento con parámetros con comentarios.

```
let led_param, LED;
/**
 * Con este procedimiento se hace parpadear un LED.
 */
function parpadearLED(led_param) {
    led_param.blink(100);
}
LED = (new five.Led({
    pin: 13,
    custom: {
        type: 'LED',
        blink: 0
    }
}));
parpadearLED(LED);
```

Figura 4.6.4.6 - Código JavaScript generado con los procedimientos

El uso de los bloques asociados a procedimientos sigue la misma lógica que el resto de los bloques funcionales. En las siguientes figuras, se muestra el uso de algunos de los bloques de procedimientos.

Figura 4.6.4.7 - Procedimiento para enviar mensajes a la consola de Blokino

Figura 4.6.4.8 - Procedimiento para esperar un tiempo antes de ejecutar una acción

4.6.5 Bloques funcionales de componentes electrónicos

Los bloques funcionales de componentes electrónicos son usados para representar de manera visual, los circuitos que se arman sobre las placas Arduino. Estos bloques funcionales se encuentran en el toolbox de Blokino, separados en secciones según su funcionamiento y el tipo de componente electrónico. Como se mencionó anteriormente, los componentes electrónicos del kit de Blokino están clasificados en 2 tipos: componentes digitales y analógicos, como se muestra en la Figura 4.6.5.1.

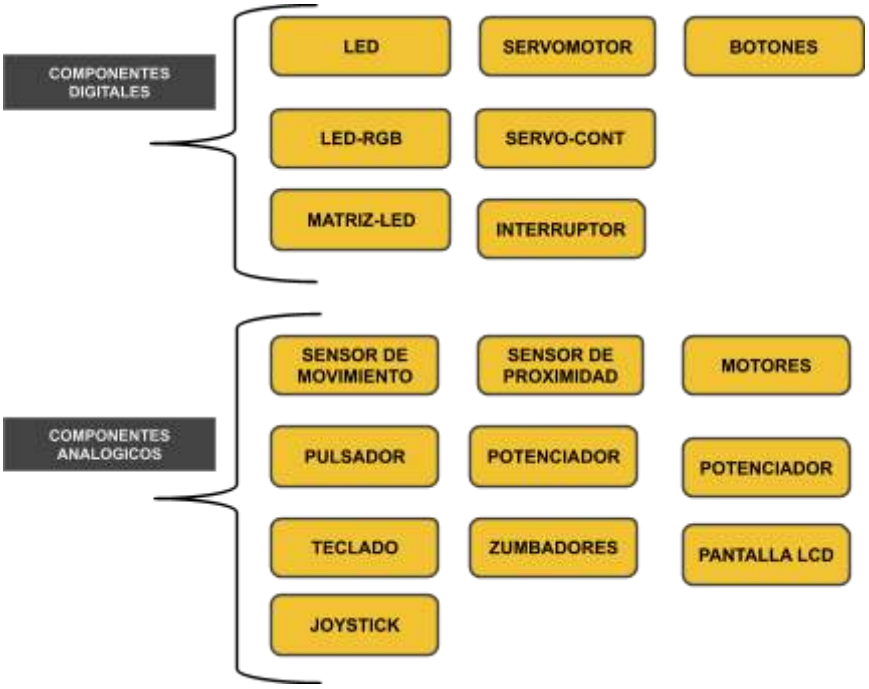


Figura 4.6.5.1 - Listado de bloques de control

Este tipo de bloques tiene asociado un circuito, especificado en la documentación oficial de Blokino, en donde se indica:

- Descripción del componente electrónico.
- Especificación de los pines de entrada.
- Creación e instanciación.
- Bloques funcionales asociados al componente electrónico.
- Explicación del código JavaScript generado con el componente electrónico.

4.6.6 Integración de JavaScript

Para crear todos los tipos de bloques funcionales de Blokino, se usa la librería Google Blockly. Blokino usa en particular JavaScript, uno de los lenguajes provistos por Blockly, para implementar la configuración de los bloques. La implementación visual, se genera por medio de módulos de JavaScript integrados en Blokino.

El código Javascript está encapsulado en módulos independientes, logrando así una estructura ordenada y escalable. Dentro de cada módulo están definidos dos métodos principales:

- Blocks: esta función contiene toda la configuración visual del bloque funcional. La librería de Google Blockly provee objetos globales para crear bloques y modificar el workspace donde se aloja el toolbox de Blokino. El objeto global Blockly permite crear bloques y definir:
  - el nombre del bloque.
  - los valores de entrada, como así también el tipo de dato asociado.
  - el tipo de anidamiento dentro del flujo del programa. Está propiedad indica cómo debe acoplarse o asociarse un bloque. Los bloques pueden ser asignados a otro bloque, o ser anidados de distintas maneras.
  - el color del bloque usando colores predefinidos mediante código hexadecimal o palabras claves.
  - mensajes flotantes de información.
  - mensajes de ayuda, que en este caso son enlaces a la plataforma web de Blokino.

La Figura 4.6.6.1 muestra el uso de la función Bloques (Blocks), y las propiedades definidas para el bloque funcional parpadear (Blink), que permite hacer parpadear un LED.

```
Blockly.Blocks["blink"] = {
    init: function() {
        this.appendDummyInput()
            .appendField("Parpadear")
            .appendField(new Blockly.FieldVariable("LED"), "current_led")
            .appendField("Tiempo")
            .appendField(new Blockly.FieldNumber(0, 1, 100, 1), "time")
            .appendField(
                new Blockly.FieldDropdown([
                    ["milisegundos", "miliseconds"],
                    ["segundos", "seconds"],
                    ["minutos", "minutes"]
                ]),
                "blink_period"
            );
        this.setInputsInline(false);
        this.setPreviousStatement(true, null);
        this.setNextStatement(true, null);
        this.setColour(30);
        this.setTooltip("Asigna el parpadeo al led.");
        this.setHelpUrl(url_documentation);
    }
};
```

Figura 4.6.6.1 - Uso de la función Bloques, donde se define el bloque funcional de parpadeo

- JavaScript: en esta función se encuentra la funcionalidad lógica de cada bloque funcional. Esta sección agrega los comandos de JavaScript Robotics usando template string. Con esta mejora de ES6 se pueden asociar los comandos de Johnny-Five con los valores de entrada, si es que los tiene disponibles el bloque funcional.
  En la Figura 4.6.6.2 se muestra cómo se implementan las funciones de “code”, en la cual se establece la lógica y del uso del bloque funcional.

```
Blockly.JavaScript["blink"] = block => {
    let led = Blockly.JavaScript.variableDB_.getName(
        block.getFieldValue("current_led"),
        Blockly.Variables.NAME_TYPE
    );
    let time = 0;
    let blink_period = block.getFieldValue("blink_period");
    let blink_time = block.getFieldValue("time");
    switch (blink_period) {
        case "miliseconds":
            time = blink_time * 100;
            break;
        case "seconds":
            time = blink_time * 1000;
            break;
        case "minutes":
            time = blink_time * 60000;
            break;
    }
    let code = `${led}.blink(${time});
                ${led}.custom.blink=${time};`;

    return code;
};
```

Figura 4.6.6.2 -Función del código JavaScript

Además de integrar los bloques funcionales con Javascript, también se estableció una estructura ordenada y autónoma. En la Figura 4.6.6.3 se muestra cómo se organizó la funcionalidad en archivos JavaScript.

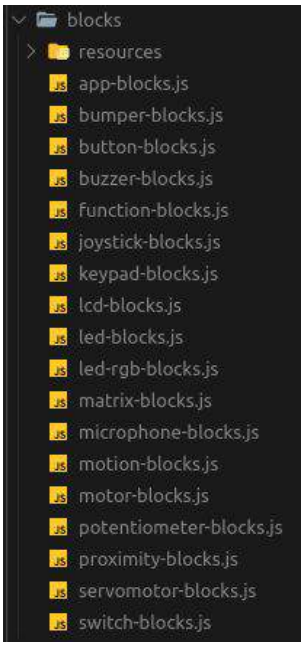


Figura 4.6.6.3 - Estructura de los bloques dividida en archivos JavaScript de acuerdo a su funcionalidad

De esta manera, cuando se quiera agregar, modificar o eliminar los bloques funcionales, sólo se debe acceder al archivo JavaScript correspondiente.

4.7 Configuración de los dispositivos físicos

Una de las problemáticas que se presentó durante las pruebas de los programas de Blokino, fue la ejecución de los programas sobre las placas Arduino. Esto se debe a que Arduino cuenta con 32 KB de memoria, espacio que puede no ser suficiente dependiendo del tamaño de programas que queremos alojar. A modo de ejemplo, JQuery (minificado) pesa exactamente 32KB, si agregamos el intérprete (como podría ser un navegador o un servidor) necesitamos espacio adicional. En el caso del servidor Node.js se requiere 16MB, es decir, por lo menos 500 veces más de lo provisto por Arduino. Para solucionar esto se debe establecer comunicación externa, es decir enviar las instrucciones del programa de Blokino mediante comunicación serial y además que Arduino sepa controlar este nuevo bloque de código. Es aquí donde surge la necesidad del uso de Firmata.

4.7.1 Firmata

Firmata es un protocolo genérico para comunicación con microcontroladores desde cualquier software instalado en un ordenador. Puede implementarse en cualquier arquitectura de microcontroladores, así como en cualquier paquete de software. Firmata permite controlar completamente Arduino desde software instalado en un ordenador, sin escribir una sola línea de código de Arduino.
El creador de este protocolo fue Hans-Chistoph y lo publicó en los repositorios de Arduino en el año 2006. Para construirlo se basó en varios protocolos de comunicación y principalmente en el protocolo MIDI (Musical Instrument Digital Interface). Mediante el uso del protocolo MIDI se representa la API de Arduino, sus tipos de datos y comandos que se envían del ordenador a la placa Arduino.

El protocolo MIDI fue elegido como el núcleo central de Firmata debido a su eficiencia, facilidad y por la gran cantidad de implementaciones existentes de libre acceso. MIDI además de ser un protocolo, también es una interfaz digital y un tipo de conector, que se usa para comunicar y conectar ordenadores, instrumentos eléctricos y otros dispositivos entre sí.

El protocolo establece mensajes estándares para la conexión/desconexión y comunicación. Firmata adaptó estos mensajes, creando un nuevo conjunto de mensajes para los tipos de datos analógicos y digitales, y el control de los mismos (pinMode, digitalWrite, etc). Además da soporte a más de 16 pines analógicos con una resolución de 14 bits y más de 128 pines digitales. Esto hace posible representar la API de Arduino usando mensajes de Firmata.

El proyecto de Firmata es de código abierto, y se han implementado diferentes versiones que dan soporte a varias funcionalidades de los microcontroladores. La versión original se llama StandardFirmata, que está incluida en el entorno de desarrollo oficial de Arduino y Wiring. La versión actual de Firmata incluye soporte para las siguientes características:

- Entradas y salidas analógicas.
- Salidas PWM.
- Comunicación entre entradas y salidas analógicas.
- Servomotores.
- Matrices LEDs.
- I2C.

Por otro lado, distintos lenguajes de programación tienen soporte con su propia versión adaptada de Firmata:

- Pyduino (Python).
- Processing.
- C++/openFrameworks
- Solid Soils (C#)
- Sharpduino (C#)
- 4ntoine (Java)
- Johnny-Five (JavaScript / Node.js).
- Hardbap (Ruby)
- Carica-firmata (PHP)

Para poder usar cualquiera de las versiones de Firmata con algún lenguaje de programación distinto a Arduino o Wiring, se debe instalar usando Arduino IDE.

4.7.1.1 Instalación

Antes de iniciar la instalación (Prototyping a Smart Device, 2016) se debe conectar una placa Arduino UNO, MEGA o NANO al ordenador. Con la placa conectada al ordenador, y con el software Arduino IDE en funcionamiento, acceder a las siguientes opciones del menú: Arduino IDE => File (Archivo) / Examples (Ejemplos) / Firmata / StandardFirmata.

En la Figura 4.7.1.1.1 se muestran las opciones de Arduino para instalar Firmata.

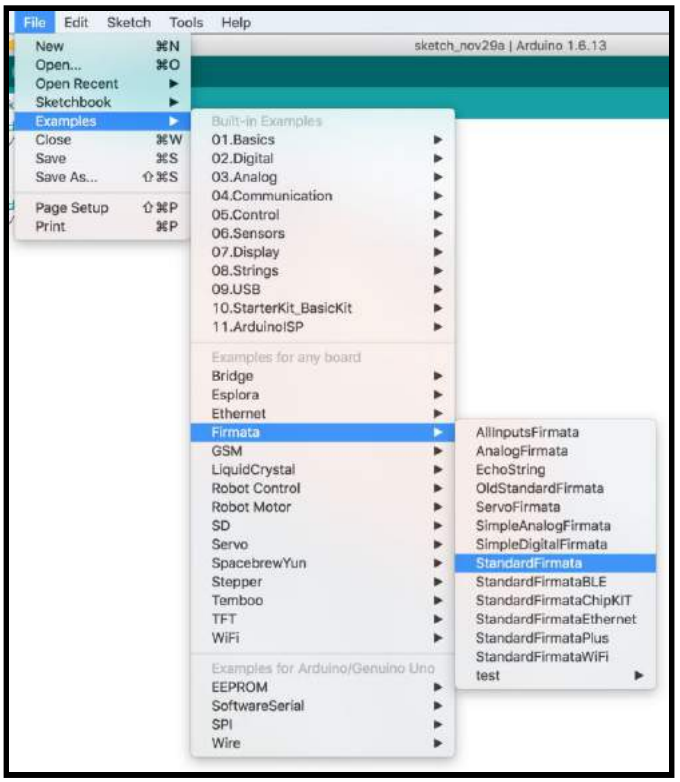


Figura 4.7.1.1.1 - Se selecciona la librería StandardFirmata

A continuación se abrirá una nueva ventana con el sketch de Standardfirmata. Para compilar e instalar Firmata en el Arduino simplemente debemos hacer pulsar en el segundo icono, que muestra una flecha en dirección a la derecha. La Figura 4.7.1.1.2 muestra él código fuente de StandardFirmata[10].

Figura 4.7.1.1.2 - Archivo de StandardFirmata

Luego, hay que seleccionar el modelo de la placa Arduino como así también el puerto donde está conectada la placa. Si esta acción no se realiza correctamente se generarán errores de compilación, accesibles en la consola de Arduino. En las Figuras 4.7.1.1.3 y

[10] Sketch son los programas o proyectos que son usados en Arduino, para ejecutar los programas en la plataforma Arduino IDE.

4.7.1.1.4, se muestra cómo se selecciona el dispositivo Arduino al cual se le va instalar StandardFirmata.

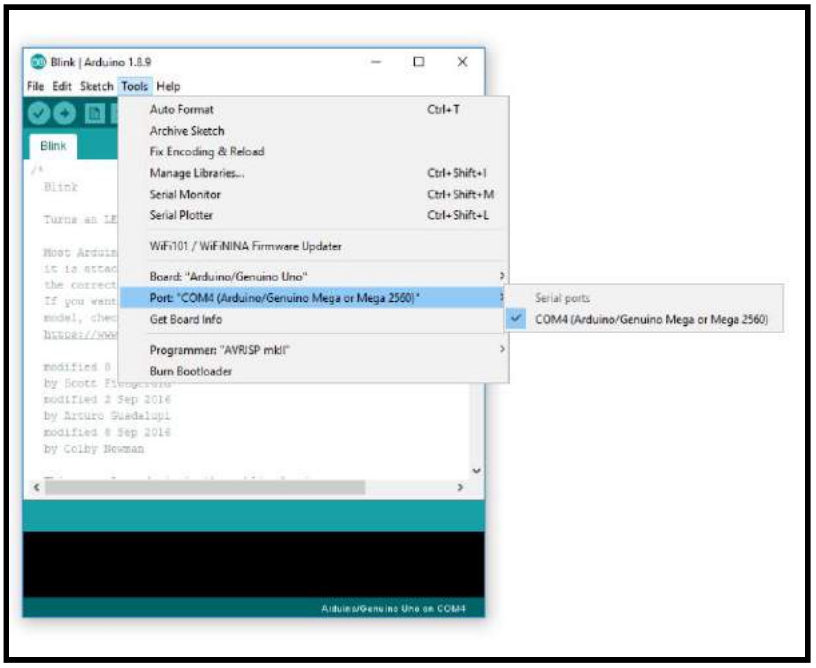


Figura 4.7.1.1.3 - Menú emergente que permite seleccionar la placa Arduino

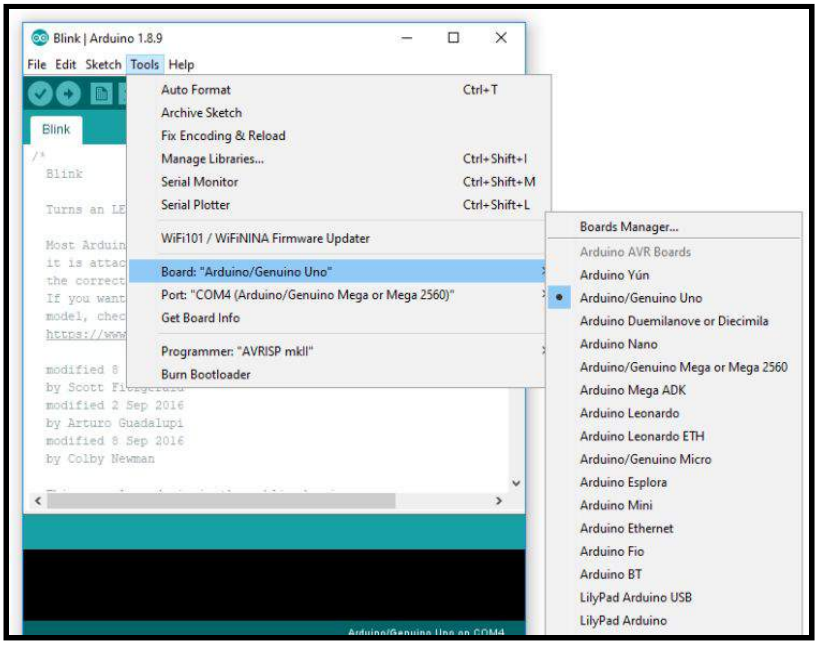


Figura 4.7.1.1.4 - Menú emergente que permite seleccionar el modelo de la placa Arduino

Luego de unos segundos la instalación se completa y el dispositivo ya es capaz de ejecutar los programas hechos con Blokino. Si bien este método se podría usar para ejecutar los programas de Blokino, considerando que la plataforma va a ser usada por estudiantes de escuelas secundarias, se decidió evitar este escenario complejo de

configuración de varios pasos. Para brindar opciones alternativas más sencillas se plantearon las opciones que se muestran en la Figura 4.7.1.1.5.

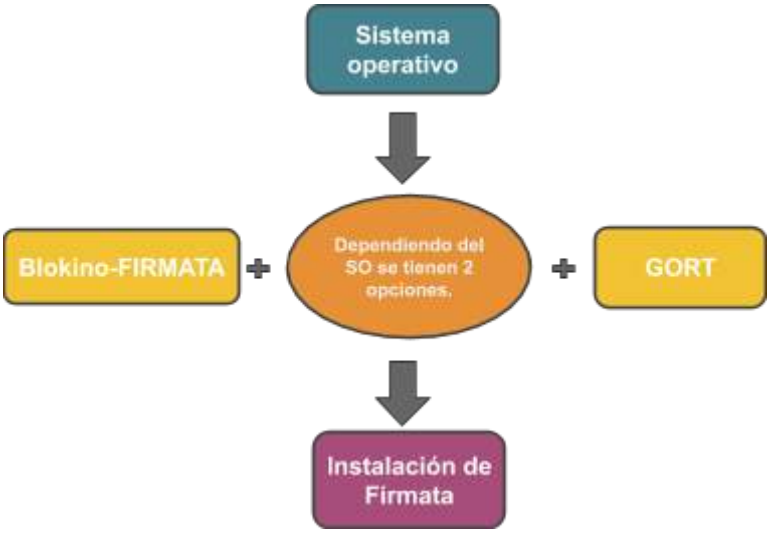


Figura 4.7.1.1.5 - Opciones de instalación de Firmata

Dependiendo del sistema operativo dOnde se instale Blokino, la configuración de las placas Arduino es distinta:

- Para las placas UNO y MEGA se usan Blokino-firmata.
- Para las placa NANO, se usa GORT.

Otro motivo para implementar estas alternativas de instalación de Firmata, es que Blokino no debería usar herramientas externas para poder ejecutar los programas que se estructuran con la plataforma. Tanto Blokino-firmata cómo Gort están implementados dentro de módulos de JavaScript reutilizables y escalables.

4.7.2 Gort

Como se mencionó anteriormente, GORT es el software usado en Blokino para la configuración de placas Arduino NANO. GORT provee un kit de herramientas usado desde la línea de comandos para instalar librerías sobre placas Arduino.

GORT tiene distintas opciones de uso, a través de:

- Instaladores para los sistemas operativos Linux, Windows y OSX.
- Archivos binarios precompilados, para todas las plataformas.

La opción de archivos binarios fue la elegida para integrarse a Blokino y se usaron en conjunto con módulos de JavaScript. Al igual que distintos módulos de Blokino, se creó el módulo **gort-firmata**. Dentro de este módulo se encuentra toda la configuración necesaria para los soportes de las placas en los distintos sistemas operativos. En la Figura 4.7.2.1, se muestra el módulo para instalar Firmata con GORT.

```
gortSetup: function(device) {
    let lib = "",
        os_arch = platform.arch();
    let options = {
        name: "Electron"
    };

    switch (os_arch) {
        case "Linx86":
            sudo.exec(
                "" +
                    path.resolve(__dirname) +
                    "/gort_lin_x86 arduino upload firmata " +
                    device +
                    "",
                options,
                function(error, stdout, stderr) {
                    if (error !== "") {
                        log(chalk.white.bgGreen.bold("FIRMATA respuesta: " +
stdout));
                        log(
                            chalk.white.bgGreen.bold(
                                "Dispositivo ",
                                device,
                                ": Firmata inatalado."
                            )
                        );
                    } else {
                        //Se presento un error
                        log(chalk.black.bgRed.bold("FIRMATA error: " + stderr));
                    }
                }
            );
            break;
    }
}
```

Figura 4.7.2.1 - Módulo para instalar Firmata con Gort

En la Figura 4.7.2.1, se muestra la configuración para que se puedan ejecutar los comandos de GORT sobre computadoras con arquitecturas x86 sobre el sistema operativo Linux. El módulo completo gort-firmata abarca las plataformas Windows y Linux, en conjunto con las arquitecturas x86 y x64. Dentro del módulo de gort, se encuentran funciones útiles usando comandos propios de gort aplicados al entorno de ejecución de Electrón y se usan los archivos binarios correspondientes al sistema operativo y el tipo de arquitectura donde se esté usando Blokino. En la Figura 4.7.2.2 se muestran las funciones del módulo de gort.

```
let gortFirmata = {
    devices: async function(callback) {
        let devices = serialportCommand.listDevices();
        let os_arch = platform.arch();
        let devices_gort = [];
        let promise = null;
        switch (os_arch) {
            case "Linx86":
            case "Linx64":
                devices_gort = await this.deviceGortLinux();
                promise = new Promise((resolve, reject) => {
                    setTimeout(() => resolve(this.setNameDevices(devices,
devices_gort)), 100);
                });
                await promise;
                break;
        }

        return callback(devices);
    },
    setNameDevices: function(devices, devices_gort) {
        devices.forEach(function(device) {
            devices_gort.forEach(function(dev) {
                if (dev.includes(device.port)) {
                    if (dev.includes("Uno")) {
                        device.name = "Arduino UNO";
                    }
                    if (dev.includes("Mega")) {
                        device.name = "Arduino MEGA";
                    }
                    if (dev.includes("ttyUSB")) {
                        device.name = "Arduino NANO";
                    }
                }
            });
        });
    }
}
```

Figura 4.7.2.2 - Opciones de instalación de Firmata

Gort posee una interfaz de línea de comandos (CLI) que permite a los usuarios ejecutar instrucciones por medio de una línea de texto simple. Con este CLI se puede instalar Firmata sobre las placas Arduino, y además es compatible con Cylon.js, Gobot, Artoo y otros. Estos últimos mencionan y recomiendan Gort en sus respectivos repositorios de Github. Una de las características de Gort, es que necesita internet para poder configurar las placas.

4.7.3 Blokino-firmata

Gort brinda soporte a Blokino para trabajar sobre placas UNO y NANO, pero no para las placas Arduino MEGA. Para cubrir este último caso se optó por crear un módulo propio de configuración, usando módulos de Arduino disponibles en NPM. Este módulo personalizado tiene el nombre de **blokino-firmata** y es usado para flashear placas Arduino. A diferencia de GORT, no necesita conexión a Internet debido a que usa los módulos localmente instalados mediante npm. Estas son las características que contiene el módulo blokino-firmata:

- Las librerías internas de npm están disponibles localmente en el entorno de Blokino.
- Tiene soporte sólo para placas cómo UNO y MEGA.

Para flashear placas Arduino se usa el módulo avrgirl-arduino. Avrgirl es una biblioteca de NodeJS, con la que se puede cargar sketchs precompilados sobre placas Arduino. En este caso el sketch que se usa es el de StandardFirmata.

- En la Figura 4.7.3.1 se muestra como está construido el módulo **blokino-firmata**.

```
flash: (options, callback) => {
    let avrgirl = new Avrgirl(options);
    let avrgirlDir = path.dirname(require.resolve("avrgirl-arduino"));
    let firmataDir = path.resolve(avrgirlDir, "junk", "hex", options.board);
    let firmataPath;
    fs.readdir(firmataDir, (err, files) => {
        if (err) {
            return log(chalk.white.bgRed.bold("Error: \n" + err));
        }
        for (let i = 0, len = files.length; i < len; i++) {
            let filename = files[i];
            if (filename.indexOf("StandardFirmata") > -1) {
                firmataPath = path.join(firmataDir, filename);
                break;
            }
        }
        if (typeof firmataPath === "undefined") {
            return log(
                chalk.white.bgRed.bold(
                    "Error: No se esncontró el Standard Firmata para la placa: " +
options.board
                )
            );
        }
        avrgirl.flash(firmataPath, callback);
    });
}
```

Figura 4.7.3.1 - Módulo de blokino-firmata

De esta manera en conjunto con gort-firmata se logró dar soporte a los tres tipos de placas Arduino.

4.8 Funcionalidades principales de Blokino

Con la plataforma Blokino además de crear programas usando usando la API de Blockly, lo que habilita la creación de bloques funcionales que luego se traducen en un lenguaje de programación, se trabajó en otras funcionalidades como el manejo de hilos de ejecución, el uso de archivos, encriptación y desencriptación.

4.8.1 Crear programas

Como se mencionó en secciones previas, la plataforma Blokino provee soporte para definir variables, estructuras de control y los tipos de datos básicos para la composición de programas, adicionalmente brinda soporte para el uso de los componentes electrónicos, mediante el uso del paradigma de programación visual.
Los programas en Blokino se construyen componiendo bloques, que representan instrucciones, y en el caso de Blokino la ejecución de estos programas se realiza sobre las placas electrónicas Arduino. Dentro de los bloques principales, se encuentran bloques que permiten crear variables, que representan tipos de datos, estructuras de control o que habilitan la definición de procedimientos. Las pautas que deben respetarse para crear programas en Blokino son las siguientes:

- Antes de usar variables se las debe declarar.
- Antes de usar un bloque de componentes electrónicos se los debe crear con sus respectivos tipos de datos.

- De la misma manera que cualquier tipo de datos, la representación de un componente electrónico se debe asignar a una variable para poder ser utilizado durante el programa.
- Usar estructuras de control, para definir flujos de ejecución.
- Usar los procedimientos para encapsular y reutilizar bloques de código.

En la Figura 4.8.1.1 se muestra un ejemplo de cómo usar el componente electrónico de la consola de mensajes.

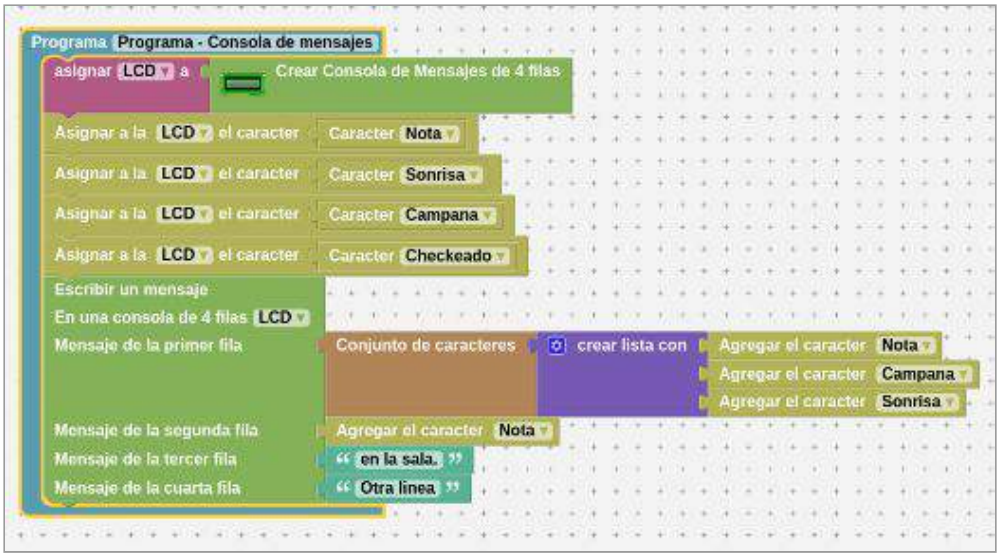


Figura - 4.8.1.1 - Ejemplo de uso de la consola de mensajes

- Primero se crea una variable con el nombre LCD, para alojar el bloque funcional del componente electrónico de la consola de mensajes.
- Con los bloques funcionales del componente electrónico de la consola de mensajes se pueden escribir mensajes estáticos o movimientos, pero además también se les puede agregar caracteres especiales. Estos caracteres son figuras, formas, animales o números. Con estos caracteres se puede formar un mensaje llamativo.
- Para poder usar los caracteres especiales en las consolas de mensajes, se deben cargar los caracteres que se quieran usar, para que luego puedan ser usados en una cadena de texto. Para realizar esto, se debe usar el bloque funcional **Asignar <variable> el carácter <carácter>**. En este bloque funcional se debe seleccionar la variables que hacen referencia a la consola de mensajes y también agregar el carácter especial que se quiere cargar en la consola de mensajes.
- Luego de tener el LCD configurado con los caracteres especiales, se crean los textos que se van a mostrar en la consola de mensajes.
- Una vez que todos los bloques funcionales quedaron unidos, se tiene un programa donde el componente electrónico del LCD tiene cargado un mensaje con caracteres especiales.

4.8.2 Ejecución de código en Blokino

Para poder ejecutar el código generado a partir de bloques en Blokino, se debe seleccionar un dispositivo conectado al ordenador. Una vez seleccionado el dispositivo,

se realizará una validación sintáctica para que el código se pueda ejecutar correctamente. Esta validación se realiza mediante el motor v8 de JavaScript.

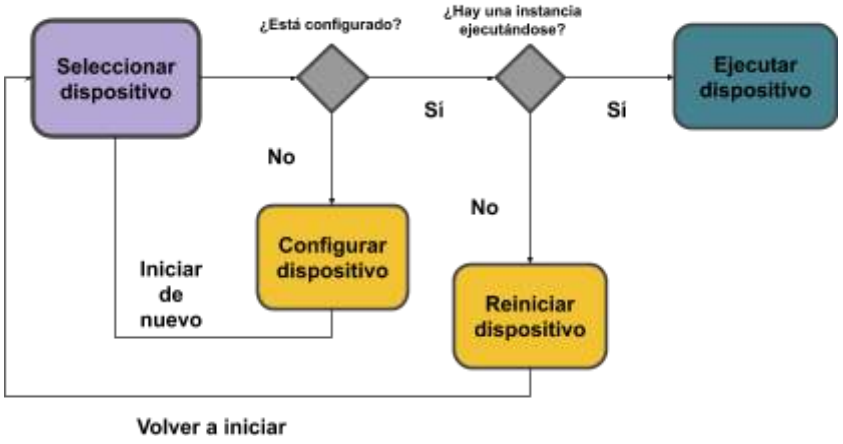


Figura - 4.8.2.1 - Validaciones antes de la ejecución de un programa en Blokino

4.8.2.1 Node.Js y los sub procesos

Para ejecutar comandos dinámicamente sobre las placas Arduino se usó Node.js, en particular sus **child process**. En la Figura 4.8.2.1.1 se muestran los tipos de procesos child process.

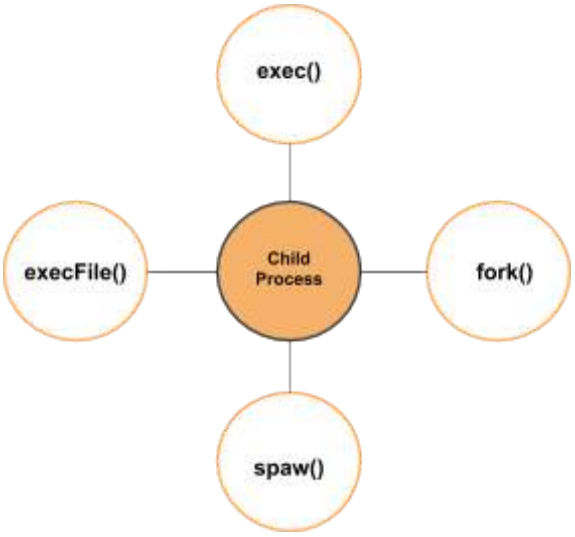


Figura 4.8.2.1.1 - Los tipos de Child Process de Node.js

Un proceso en una CPU no será suficiente para manejar la carga de trabajo de una aplicación, que en este caso las distintas instancias de ejecución de los programas de Blokino.
Node.js está diseñado para construir aplicaciones distribuidas con muchos nodos, de ahí su nombre. El uso de múltiples procesos es la mejor manera de escalar una aplicación Node.
Para contar con nodos de ejecución se optó por usar el módulo de child process de Node.js. Los nodos se ejecutan en segundo plano, con un sistema integrado de mensajería para comunicarse de manera bidireccional o por broadcast de mensajes.

Los child process se usaron en Blokino para desplegar nodos de ejecución de Johnny-Five y para establecer una comunicación asincrónica con el motor Chrome V8 que verifica la correcta compilación del código fuente. Al igual que todas las configuraciones o funcionalidades que pueden ser reutilizables, se agrupó la funcionalidad y documentación referente al despliegue de nodos en un módulo de JavaScript llamado “handler-child-process”.

4.8.2.2 Manejo de los hilos de ejecución

Para poder ejecutar los programas de Blokino, además de tener el dispositivo Arduino configurado con Firmata, se deben controlar las instancias de ejecución. Éstas están representadas con **child process** de Node.js como se mencionó anteriormente. En la Figura 4.8.2.2.1 se muestra cómo se planteó la arquitectura de **child process**.

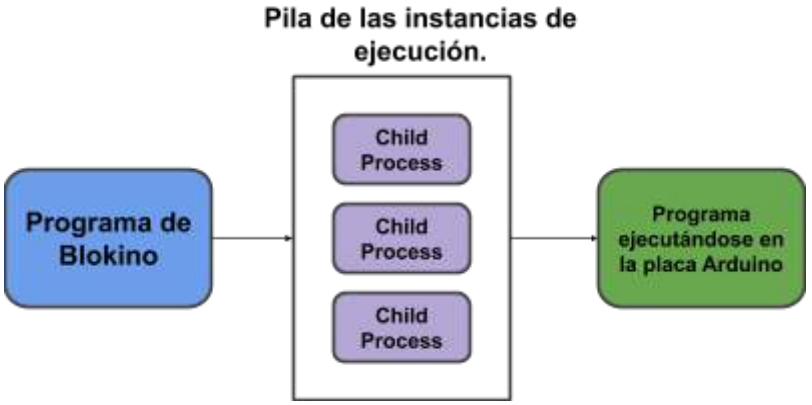


Figura 4.8.2.2.1 - Arquitectura de ejecución mediante child process de Node.js

Está arquitectura permite ejecutar distintos programas sin la necesidad de desconectar y conectar la placa Arduino. Para poder ejecutar un programa se ejecutan los siguientes pasos:

- Cada instancia de child process es almacenada en un pila homogénea de procesos. Cada uno de estos procesos tiene un identificador, que es usado para gestionarlo.
- Cuando se ejecuta un programa nuevo, se eliminan todas las instancias de ejecución guardadas, las cuales se reconocen en base a sus identificadores.
- Una vez que la pila de instancias de ejecución se vació, se puede ejecutar el programa de Blokino.

Con este tipo de organización se evitó que se generen errores de uso de memoria y de control de ejecución de programas construidos con Blokino. Además permitió solucionar un problema relacionado a la ejecución de varios programas donde era necesario desconectar y reconectar las placas Arduino.

4.8.3 Validación de código

En los lenguajes convencionales de programación como Java, Python y C los tipos de datos son importantes, porque definen qué valores entenderá e interpretará el motor de validación que contiene cada uno de los lenguajes. En Blokino esto no es diferente, dado que si no se definen bien los tipos de datos se pueden generar inconsistencia de datos o

perjudicar la integridad del programa. Por este motivo Blokino tiene integrado dos tipos de módulos, uno usado para la validación interna y otro para validación externa:

- En la **validación interna** se usa la librería Esprima[11], para validar código JavaScript. Esta librería se usa para análisis de infraestructura de ECMAScript. Con la validación se obtiene un árbol de validación que representa el programa escrito. Una vez obtenido el árbol validado del programa, para que los programas puedan ejecutarse deben pasar por patrones de validación implementados en Blokino. La validación interna realiza una revisión sintáctica y semántica del código JavaScript, sin generar errores en tiempo de ejecución.
- Luego de realizar la validación interna, se debe aplicar la **validación externa** usando el motor Chrome v8 de JavaScript**.** La diferencia entre la validación interna y la externa, es que la validación externa es capaz de generar errores en tiempo de ejecución, de manera que se los puede manejar.

Estas validaciones trabajan en conjunto para lograr una validación completa de los programas de Blokino.

4.8.3.1 Validación interna

Al principio se realizaron validaciones usando el árbol de código de XML del programa que se genera con Blockly. Con este tipo de validación se recorre el árbol de XML de manera de verificar inconsistencias. En la Figura 4.8.3.1.1 se muestra la estructura de validación manual planteada como primera aproximación a la validación de programas de Blokino.

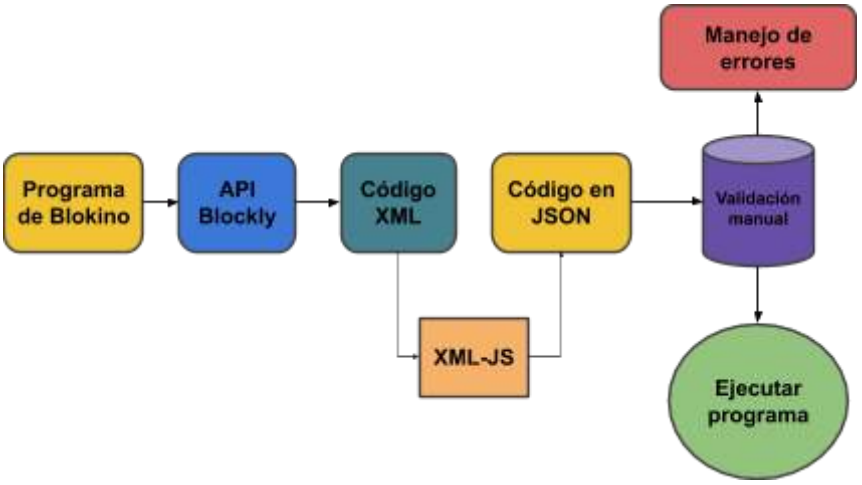


Figura 4.8.3.1.1 - Primera propuesta de validación manual de código en Blokino

- Al programa de Blokino se le aplican funciones de parseo usando la API de Blockly. Con esto se obtiene el código XML del programa de Blokino.
- Usando la librería **XML-JS** se transforma el código XML a código en formato JSON.
- Se recorrió la estructura del código JSON, siguiendo un orden para poder analizar línea por línea el código del programa, buscando inconsistencias tanto semánticas como sintácticas:
  - Se identifican los procedimientos.

[11] Sitio oficial de Esprima: https://esprima.org/

    - Se identifican las variables definidas en el programa.
    - Se identifican las estructuras de control.
    - Se verifica las variables definidas antes de usarlas.
    - En el caso de encontrar bloques funcionales:
        - Verificar que estén definidos con sus valores iniciales, por ejemplo los pines de la conexión con la placa Arduino.
        - Si hace referencia a una funcionalidad asociada a un componente electrónico, verificar que esa funcionalidad corresponda al tipo de componente electrónico.
- En el caso de presentarse algún tipo de problema, se construyó un grupo de ventanas modales para informar el tipo de error que se presentó.

Este tipo de validación se encapsuló en un módulo independiente, tal como se realizó para las demás funcionalidades. Esta solución es útil para validaciones acotadas, como por ejemplo para validar si un LED está encendido y tiene el pin de datos de tipo entero. Sin embargo para el caso de los desafíos o el modo experto de Blokino, los distintos flujos de validación se incrementan notablemente, dejan de ser 1 o 2. Por ello el módulo de validación basado en niveles dejó de ser una opción para controlar los programas de Blokino. Además resultó en un módulo complejo, escasamente extensible, dado que la validación de código se realizada en forma lineal por niveles, quedando múltiples acciones que no se pueden controlar.

4.8.3.2 Validación externa

Como se mencionó anteriormente, el motor Chrome V8 se usó para la validación externa en Blokino. Chrome V8 es un motor de código abierto para JavaScript creado por Google, siendo su programador jefe Lars Bak. Está escrito en C++ y es usado en Chromium, Google Chrome y en Microsoft Edge a partir del 2019. También está integrado en el navegador de Internet del sistema operativo Android 2.2 “Froyo”. Su primera versión fue lanzada en el 2008 y desde entonces, gracias a la política de distribución como software de código abierto y con la posibilidad de ser integrado en cualquier tipo de aplicación, se ha usado en proyectos relevantes como:

- Navegadores Opera y Vivaldi.
- La base de datos NoSQL Couch DB.
- La plataforma Node.js para el uso de JavaScript en el servidor.
- El framework Electron, para el desarrollo de aplicaciones de escritorio con tecnologías web.

Existen implementaciones de Chrome V8 para las principales plataformas: Windows, MacOS y Linux, de manera que podemos considerarlo como base para el desarrollo de aplicaciones multiplataforma.

Chrome V8 es considerado como uno de los motores Javascript de más alto rendimiento. Basa su funcionamiento en dos componentes: un intérprete denominado *ignition* y un compilador optimizado de código denominado *turbofan*.

El intérprete *ignition* genera los bytecodes de la máquina virtual de JavaScript, mientras que el *turbofan* es un compilador JIT que genera código optimizado para la plataforma en que se ejecute. Chrome V8 crea un hilo especial que realiza un *profiling* de la ejecución del código generado para detectar las partes “importantes”, es decir que las que más se ejecutan y consumen más tiempo de CPU y de esta manera mejorar la optimización del código de máquina generado.

4.8.3.2.1 Método de validación

Uno de los aspectos más importantes de Chrome V8 es que ha sido diseñado con la idea de usarlo independientemente del navegador e integrarlo en aplicaciones desarrolladas en C/C ++. Precisamente esta característica es la que ha permitido el desarrollo del proyecto Node.js que convierte a JavaScript en un lenguaje de propósito general, como así también poder aplicarlo en entornos de programación como las aplicaciones Electron, teniendo al motor Chrome V8 integrado localmente. En la siguiente Figura 4.8.3.2.1.1 se muestra la estructura planteada para trabajar con el motor Chrome V8 y child-process.

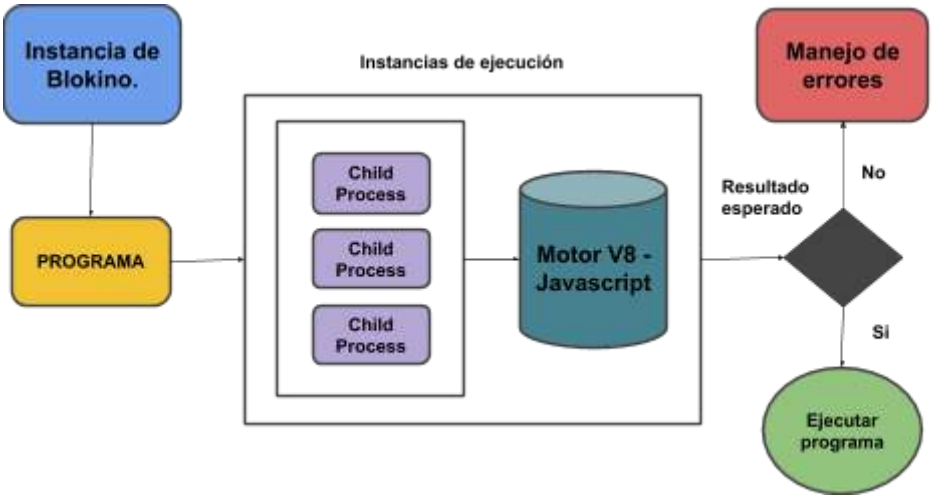


Figura 4.8.3.2.1.1 - Estructura de validación usando V8

Para realizar la validación con el motor Chrome V8, se reconoció el problema que las instancias de Johnny-Five no pueden ejecutarse en el mismo child process que está usando Blokino para ejecutar la aplicación con Electron. Una opción para salvar este problema podía ser descartar el motor Chrome V8, y usar funciones como eval(), que se encargan de interpretar código pero sin la posibilidad de devolver excepciones y con problemas de seguridad por posibles inyecciones de dependencias. Para evitar todos estos problemas, es que se optó por usar el motor V8.
En la Figura 4.8.3.2.1.1 se pueden observar varias instancias de child process, esto se debe, como se mencionó previamente, a que el código de Johnny-Five no puede convivir en el mismo child process de Electron. Para validar el código de los programas de Blokino con Chrome V8 se realiza la siguiente secuencia de pasos:

- Por cada instancia de ejecución del programa de Blokino, se debe crear un child process.
- Cada child process está guardado en una estructura de datos de tipo lista, almacenados con identificadores para poder gestionarlos desde el módulo de ejecución de Blokino.
- Cuando se ejecuta un child process con el código de Johnny-Five, se crea un nuevo nodo de ejecución de Node.js, este nodo se está ejecutando dentro del entorno de Electron.
- Las instancias pueden ser múltiples, mientras no exceda el límite de la memoria RAM de la computadora en la que se está ejecutando Blokino. Como la ejecución se

realiza sobre una sola placa Arduino, sólo se tiene un nodo secundario en segundo plano ejecutándose en el entorno de Electron.

- Cuando se ejecuta un programa, se eliminan todos los nodos de ejecución secundarios que están ejecutándose en el Blokino. De esta manera no se genera ninguna excepción relacionada a que se no se tiene espacio o que la placa se encuentra ocupada por otra instancia de Johnny-Five.
- Luego de vaciar la lista de child process y de ejecutar la instancia de Johnny-Five, se pueden generar dos escenarios:
  - El primero es que se ejecute el programa correctamente y devuelva en el canal de comunicación un mensaje notificando esta situación.
  - El segundo escenario es que no se puede ejecutar el programa y se manda por otro canal de comunicación un mensaje notificando que se presentó algún tipo de error.

4.8.3.2.2 Buffer de comunicación

Para la comunicación entre los nodos secundarios (instancias que se están ejecutando) y el primario que es Electron, se aplicó el concepto de buffers de comunicación asincrónicos. La Figura 4.8.3.2.2.1 muestra cómo se implementaron los buffers de comunicación.

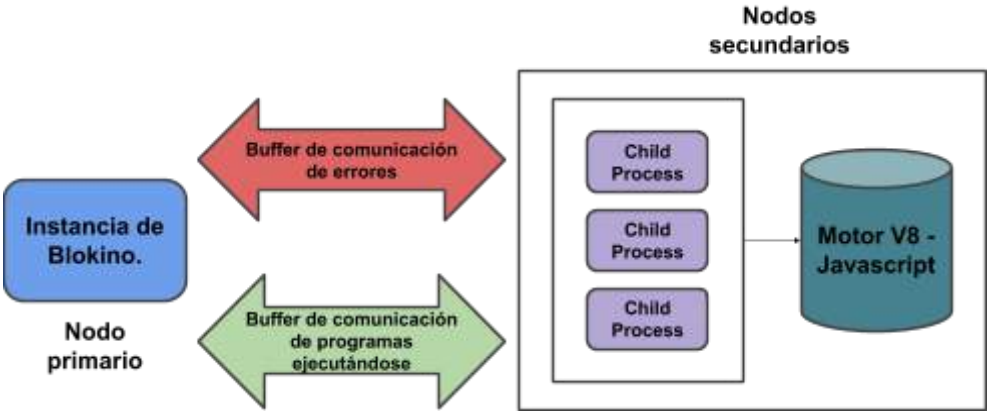


Figura 4.8.3.2.2.1 - Estructura de buffers de comunicación

Los buffers de comunicación tienen una función en común, que es esperar mensajes para poder desencadenar una acción informativa para el usuario que está usando Blokino. Mediante estos mensajes se controlan las ventanas modales de información por ejemplo para los siguientes casos:

- Cuando se genera un error de sintaxis o semántica en el código que se está ejecutando sobre un child process de Node.js.
- Para controlar el resultado de la ejecución del child process del programa de Blokino.
- Para controlar el tipo de error que se generó cuando se quiso ejecutar el código del child process con Johnny-Five.

Al usar los buffers de comunicación, se logró un control de excepciones muy amplio. Se puede cubrir la mayoría de los escenarios de excepción de las respuestas de la ejecución de los child process. Una mejora que se planteó durante el desarrollo es crear módulos más detallados para el control de excepciones.

4.8.4 Descargar un proyecto generado en Blokino

La plataforma Blokino tiene la funcionalidad de guardar proyectos en el sistema de archivos de la plataforma. La apariencia de la ventana de diálogo que se abre cuando se pulsa el botón “Descargar proyecto” dependerá del sistema operativo donde se ejecute.
La descarga de un proyecto generado en Blokino se realiza en dos pasos, transparentes para el usuario final. La Figura 4.8.4.1 muestra la estructura que se planteó para esta funcionalidad sobre los proyectos de Blokino.

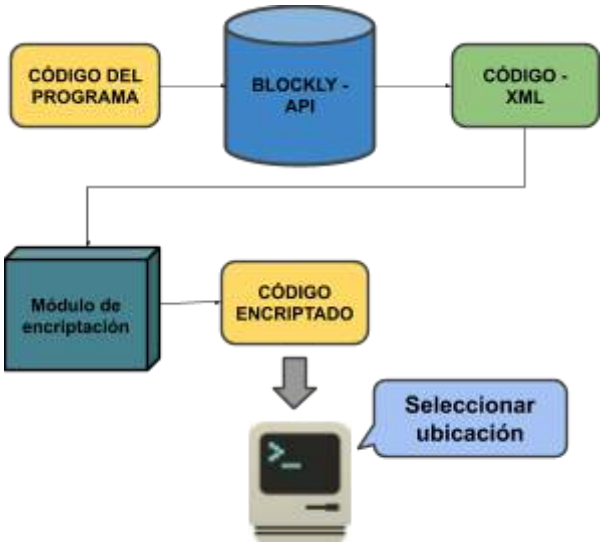


Figura 4.8.4.1 - Estructura del método de encriptación y descarga

En esta estructura se presentan pasos esenciales que ocurren antes de guardar el proyecto:

- En el primer paso se debe seleccionar el programa usando la API de Blockly.
- La API de Blockly recibe el workspace de Blokino, que es donde se encuentran los programas construidos con Blokino. La API devuelve el código XML del proyecto.
- Con el código XML de la estructura del proyecto, se trabaja usando encriptación para la seguridad de la plataforma y del entorno donde se use. Esto se realiza a modo preventivo, y así evitar posibles inyecciones de dependencias cuando se vuelva a cargar el proyecto en Blokino.

4.8.4.1 Método de encriptación

Para realizar la encriptación se usó la librería estándar de encriptación llamada CryptoJS. Esta librería contiene una colección algoritmos criptográficos estándares y seguros, implementados en JavaScript y utilizando las mejores prácticas. Los algoritmos en CryptoJS son rápidos y tienen una interfaz consistente y simple.
El módulo de encriptación en Blokino está compuesto por la funcionalidad necesaria para la encriptación de los archivos de Blokino.

```
downloadCode: (Blockly, workspace) => {
    dialog.showSaveDialog(fileName => {
        if (fileName === undefined) {
            log(CHALK.gray.bgRed.bold(MESSAGES.modals().files.errors.upload));
            return;
        }
        let xmlBlokino = Blockly.Xml.workspaceToDom(workspace);
        let xmlPlaneText = Blockly.Xml.domToText(xmlBlokino);
        let codeEncrypt = CRYPTOJS.AES.encrypt(xmlPlaneText, PASS);
        let name = fileName + ".blokino";
        FS.writeFile(name, codeEncrypt, err => {
            if (err) {
                utilFunctions.setModalError(
                    "",
                    MESSAGES.modals().proyect.errors.download,
                    MESSAGES.modals().try_again
                );
            }
            utilFunctions.setModalSuccess(MESSAGES.modals().proyect.download);
            utilFunctions.addMessage("info", `Se descargó correctamente el
proyecto`);
        });
    });
},
```

Figura 4.8.4.1.1 - Método para descargar proyectos encriptados

En la Figura 4.8.4.1.1 se muestra cómo se implementó el método de encriptación usando el algoritmo AES de CryptoJS. Para que este algoritmo funcione se estableció una contraseña privada de Blokino. Todos los proyectos que se descargen de Blokino, tendrán asociada la clave privada de CryptoJS. Además tendrá una extensión .blokino, esto es útil cuando se quiera importar un proyecto Blokino. Dado que además de validar la encriptación del proyecto, también se valida el tipo de extensión.

4.8.5 Abrir un proyecto en Blokino.

La plataforma Blokino ofrece la funcionalidad que permite abrir proyectos previamente desarrollados en esta misma plataforma.
El método para poder abrir los proyectos de Blokino, consiste internamente de varios pasos. La Figura 4.8.5.1 muestra la estructura que se planteó para esta funcionalidad sobre los proyectos de Blokino.

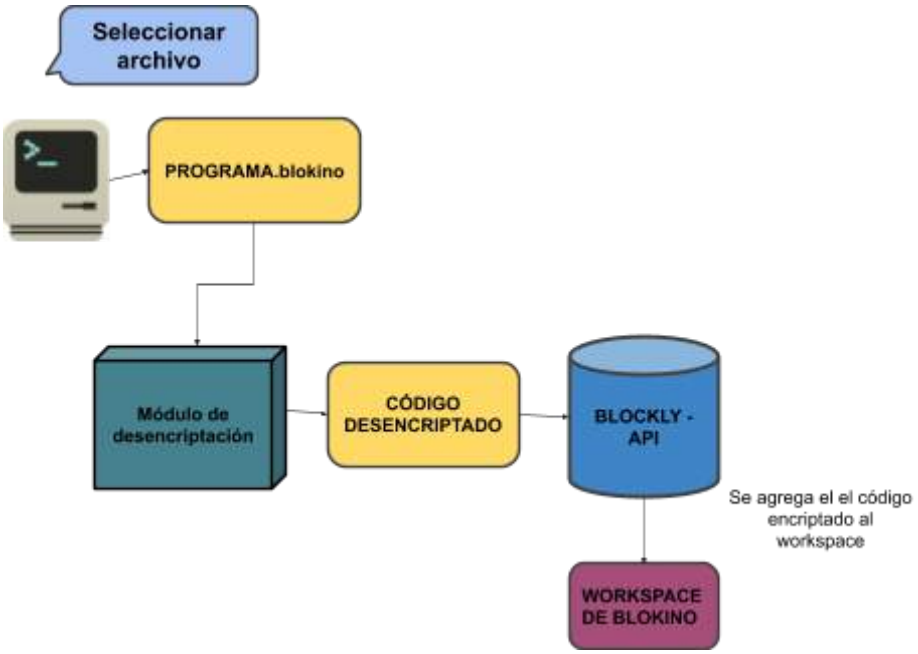


Figura 4.8.5.1 - Estructura del método de desencriptación

- El primer paso consiste en seleccionar el proyecto Blokino desde el sistema de archivos. Una vez seleccionado, comienza la validación de la desencriptación.
- El archivo será reconocido mediante la API de Dialog de Electron. El contenido del proyecto se debe desencriptar mediante una contraseña privada con el que originalmente se guardó el proyecto.
- La API de Blockly recibe los byte-codes desencriptados del proyecto. La API devuelve el código XML del proyecto.
- Al obtener el código XML de la estructura del proyecto, se puede agregar al workspace de Blokino. Concluida esta incorporación, el proyecto se mostrará en la plataforma.

4.8.5.1 Método de desencriptación

Como se mencionó antes para la encriptación en Blokino, la librería CryptoJS también fue usada en la desencriptación y esta funcionalidad se encuentra en el respectivo módulo.

```
uploadCode: (Blockly, workspace) => {
    dialog.showOpenDialog(fileNames => {
        if (fileNames === undefined) {
            log(CHALK.gray.bgRed.bold(MESSAGES.modals().files.errors.upload));
            return;
        }
        let readStream = FS.createReadStream(fileNames[0]);
        readStream.on("error", err => {
            $("#modal-error-upload-program").modal();
        });
        readStream.on("data", codeEncrypt => {
            try {
                let bytesCode = CRYPTOJS.AES.decrypt(codeEncrypt.toString(), PASS);
                let xmlCode = bytesCode.toString(CRYPTOJS.enc.Utf8);
                let xmlBlokino = Blockly.Xml.textToDom(xmlCode);
                Blockly.mainWorkspace.clear();
                Blockly.Xml.domToWorkspace(xmlBlokino, workspace);
                utilFunctions.setModalSuccess(MESSAGES.modals().proyect.upload);
                utilFunctions.addMessage("info", `Se abrió correctamente el
proyecto`);
            } catch (error) {
                utilFunctions.setModalError(
                    "",
                    MESSAGES.modals().format,
                    MESSAGES.modals().try_again
                );
            }
        });
    });
},
```

Figura 4.8.5.1.1 - Método para importar proyecto Blokino

En la Figura 4.8.5.1.1 se muestra cómo se implementó el método de desencriptación usando el algoritmo AES de CryptoJS. Los proyectos que se quieran importar, deben tener la clave CryptoJS privada de Blokino incorporada en el código. Además deben tener la extensión .blokino.

4.8.6 Borrar un proyecto

Otra de las funcionalidades que provee Blokino es la de reiniciar el workspace donde los bloques funcionales se usan para crear los programas. La acción de reiniciar el workspace permite borrar el proyecto en curso, esto se logra haciendo uso de la API de Blockly**.** En la Figura 4.6.8.1 se muestra el procedimiento **cleanCode,** que es el encargado de reiniciar el workspace de Blokino.

```
cleanCode: (Blockly, document, workspace) => {
    Blockly.mainWorkspace.clear();
    $("#loader").css("display", "block");
    setTimeout(() => {
        $("#loader").css("display", "none");
        let xml = Blockly.Xml.textToDom(CODE_BASE.application().program);
        Blockly.mainWorkspace.clear();
        Blockly.Xml.domToWorkspace(xml, workspace);
        utilFunctions.addMessage("info", `Se limpió el tablero de bloques`);
    }, 500);
}
```

Figura 4.8.6.1 - Módulo para reiniciar el workspace de Blokino

En este procedimiento de reinicio, se realizan las siguientes acciones:

- Se modifican reglas de CSS usando selectores provistos por Jquery.
- Mediante los namespace de Blockly, se hace uso de las funcionalidades relacionadas a la manipulación de documentos XML del API de Blockly y se reinicia el workspace principal de la plataforma**.**
- Se inserta una estructura vacía de XML. De esta manera el proyecto es eliminado finalmente del workspace.

4.8.7 Ver/Ocultar código JavaScript

Blokino provee funcionalidad que permite ver u ocultar el código JavaScript generado a partir de los bloques. Otra funcionalidad incorporada permite copiar en el portapapeles o descargar el código del programa. En la Figura 4.8.7.1, se puede observar cómo se planteó la estructura para poder visualizar el código JavaScript.

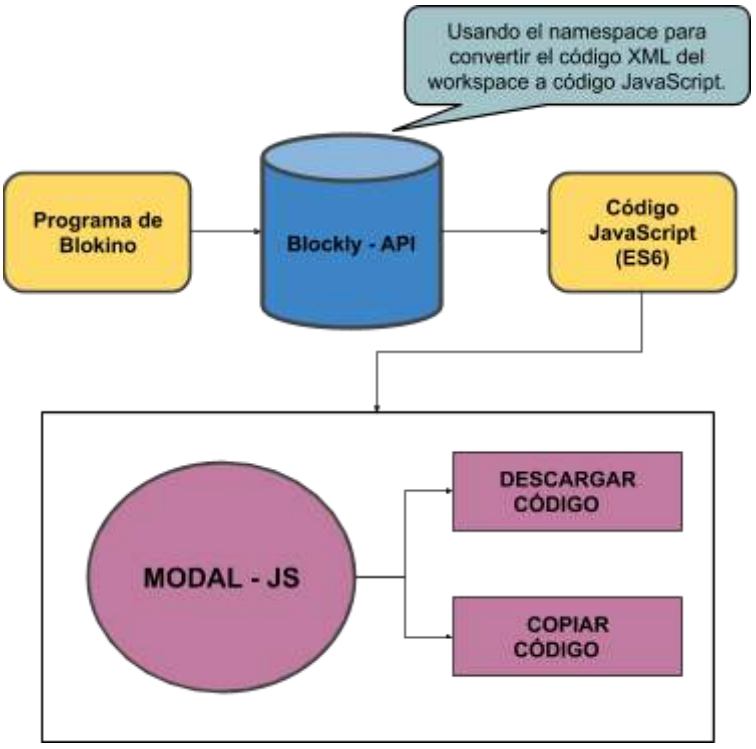


Figura 4.8.7.1 - Estructura para ver el código JavaScript

La estructura está compuesta por distintas funcionalidades como:

- El código base del workspace de Blokino está en XML y para obtener el código JavaScript se debe usar la API de Blockly.
- Una vez obtenido el código Javascript, se puede mostrar en la ventana modal destinada a este propósito. Dentro de esta ventana se tienen funcionalidades extras como:
    - Descargar código: usando el módulo fs de NPM se encapsula el código en un archivo temporal, luego se seleccionará una ubicación en el ordenador. Esta funcionalidad involucra el módulo fs de NPM, como así también la API de Electron para manejar las ventana de diálogo propias del sistema operativo.

- Copiar código: para poder usar el código externamente sin descargarlo, también se lo puede copiar en el portapapeles del sistema operativo. Para poder controlar el portapapeles, se usó el API de Electron.

4.9 Instaladores

La plataforma está soportada para arquitecturas de x64. Para realizar la generación de instaladores se usó electron-builder.

4.9.1 Creación del instalador

La distribución final de Blokino es generada haciendo uso de la librería Electron-builder. Esta librería ofrece una solución completa para empaquetar y crear una aplicación lista para la distribución de Electron, Proton Native para macOS, Windows y Linux con soporte de actualización automática. Para Blokino, las librerías relacionadas con Electrón deben ser compatibles con la versión ^4.0.3 de Electron. Si las librerías que se agregan son incompatibles con esa versión, la aplicación entra en un estado de incompatibilidad de módulos. Existen algunos casos particulares como el de la librería SerialPort (IoT) relativa a robótica y usada en Blokino, que si bien no especifica puntualmente compatibilidad con la versión de Electrón requerida, pudo ser usada sin inconvenientes. Además se incorporó Electron-Rebuild, que es un módulo de NPM usado para corregir módulos de JavaScript en la creación de instaladores.
SerialPort cuenta con una licencia open source MIT. Para poder integrarlo al proyecto Blokino se incorpora como una subdependencia de Johnny-Five, para esto, se debe ejecutar previamente scripts de regulación que provee Electrón-Rebuild. Luego de ejecutar los scripts pertinentes para la regulación e instalación correcta de SerialPort, ya se puede generar un instalador sin conflictos de dependencias y de subdependencias.
Como se mencionó anteriormente, la plataforma Blokino tiene soporte para Windows y Linux con arquitectura de 64 bits, la Figura 4.9.1.1 muestra la configuración de Electron-Build.

```
"build": {
    "appId": "Blokino-v1.0",
    "asar": false,
    "linux": {
        "target": [
            "deb"
        ],
        "files": [
            "!resources/circuits/diagrams/*",
            "!resources/examples-code/*",
            "!resources/installers/*",
            "!resources/notes/*",
            "!resources/README.md"
        ]
    },
    "dmg": {
        "contents": [
            {
                "x": 110,
                "y": 150
            },
            {
                "x": 240,
                "y": 150,
                "type": "link",
                "path": "/Applications"
            }
        ]
    },
    "win": {
        "target": "NSIS",
        "icon": "installers/icons/icon.ico",
        "files": [
            "!resources/circuits/diagrams/*",
            "!resources/examples-code/*",
            "!resources/installers/*",
            "!resources/notes/*",
            "!resources/README.md"
        ]
    }
}
```

Figura 4.9.1.1 - Objeto de configuración

El instalador generado se guardará en la carpeta dist, listo para ser usado.

4.9.2 Ejecutar plataforma

Para poder ejecutar Blokino se debe acceder al archivo ejecutable, esto varía dependiendo del sistema operativo donde se ejecute. Entre las opciones que soportan Blokino están: Windows 10 y Linux Debian.

En el caso de la plataforma Windows se debe ejecutar **Blokino.exe**. La ejecución se puede hacer de manera convencional o vía administrador.

En linux, para poder ejecutar Blokino se debe desempaquetar el archivo Blokino.desktop.

A diferencia de Windows, en Linux se debe ingresar la contraseña de administrador para poder darle los permisos necesarios a Blokino.
En el caso de la plataforma Linux, se debe ejecutar el archivo que se encuentra dentro del empaquetado que se debe bajar de la página oficial de Blokino. Al ejecutar la aplicación de Blokino.desktop, se debe ingresar la clave de administrador. Si la clave no se ingresa y se abre de manera tradicional, Blokino no funcionará correctamente, dado que los hilos de ejecución de Johnny-Five no se podrán ejecutar por falta de permisos y no se podrán detectar los dispositivos conectados desde Johnny-Five.

4.10 Secciones

La plataforma Blokino está compuesta por secciones diferentes, donde cada una de éstas hace referencia a funcionalidades e información para el uso de Blokino. La Figura 4.10.1 muestra estas secciones.

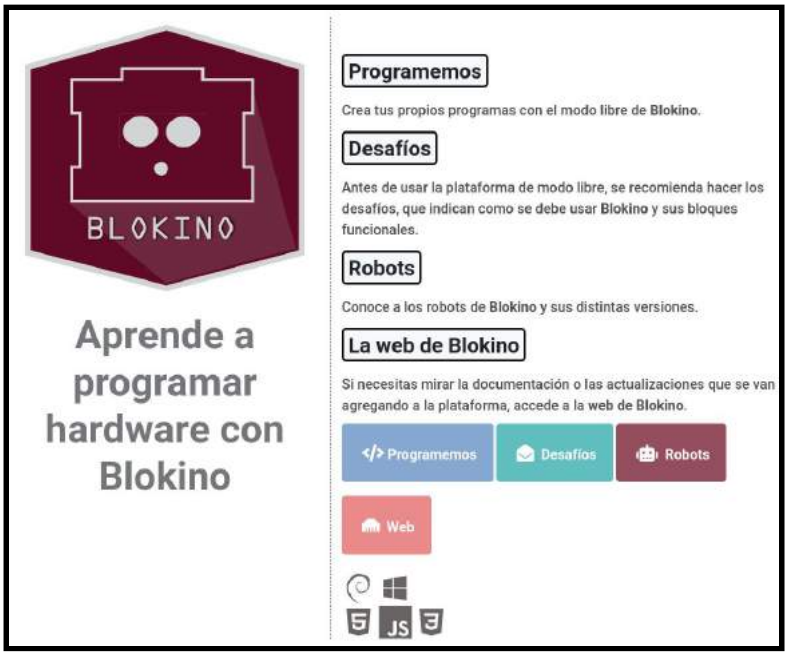


Figura 4.10.1 - Pantalla de inicio de Blokino

A continuación se describen las secciones que conforman la pantalla inicial de Blokino:

- Programemos: es la sección libre de Blokino
- Desafíos: es la sección dedicada al aprendizaje incremental mediante desafíos.
- Robots: en esta sección se tiene toda la información necesaria para armar los NodeBots de Blokino.
- Web: es el enlace a la documentación oficial de Blokino.

4.10.1 Desafíos

Para aprender a usar la plataforma, se desarrolló una sección donde están definidos ejercicios incrementales de aprendizaje llamados desafíos. Para poder acceder, se debe pulsar el botón “Desafíos” de la pantalla inicial de Blokino.
Para que el estudiante pueda hacer proyectos complejos con Blokino, se debe saber cómo crear variables, usar estructuras de control y sobre todo cómo crear los bloques funcionales. En la Figura 4.10.1.1 se muestra la descripción de la sección “Desafíos”.

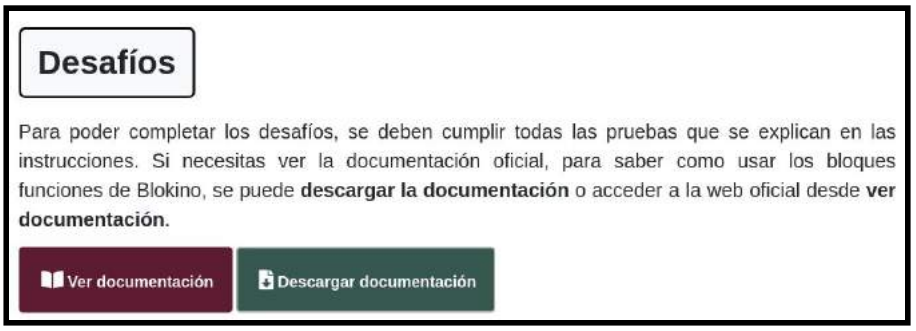


Figura 4.10.1.1 - Descripcion de los desafíos

Cada bloque funcional de Blokino contiene una configuración distinta para usar con los bloques funcionales y el armado del circuito. Por eso motivo cada bloque funcional tiene su propia sección de desafíos.
Debido a que se tienen distintos tipos de bloques funcionales que tienen una complejidad de uso distinta, se decidió agruparlos en niveles de complejidad.
Para los bloques funcionales como Leds, Leds-RGB, botones, pulsadores y los interruptores se los agrupó en el nivel Inicial. En la Figura 4.10.1.2, se muestra el nivel inicial de los desafíos.

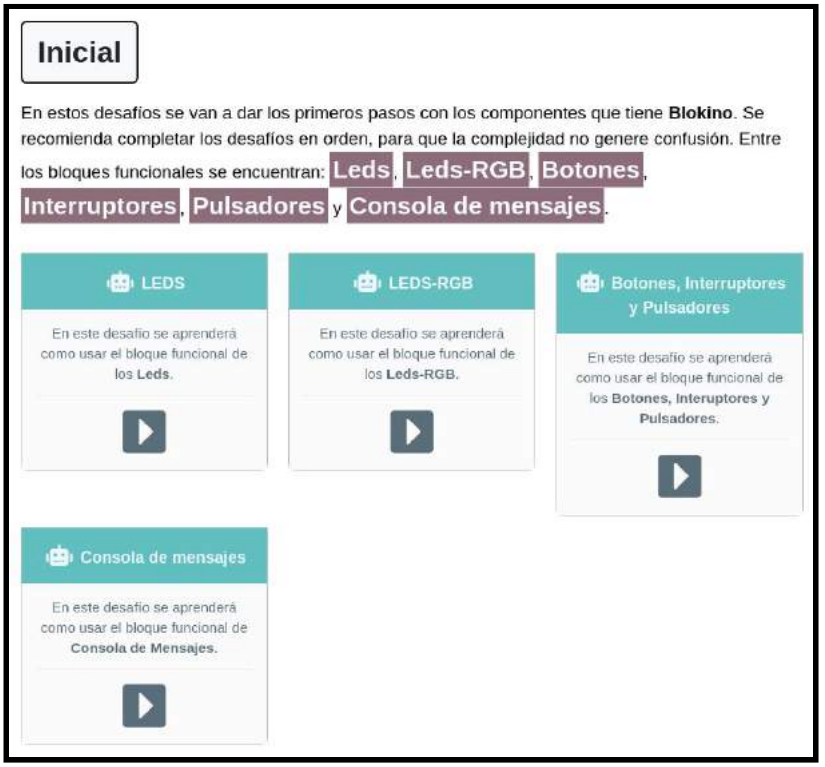


Figura 4.10.1.2 - Sección de nivel Inicial

Para los bloques funcionales como Potenciometro, Joystick, Zumbadores se los agrupó en el nivel Intermedio. En la Figura 4.10.1.3 se muestra el nivel intermedio de los desafíos.

Figura 4.10.1.3 - Sección de nivel intermedio

Para los bloques funcionales como Matriz-LEDS, Servomotores, Sensores de movimiento, Sensores de proximidad, Teclado y Motores se los agrupó en el nivel Avanzado. En la Figura 4.10.1.4, se muestra el nivel avanzado de los desafíos.

Figura 4.10.1.4 - Sección de nivel avanzado

Sí los niveles anteriores son completados, o si el estudiante se siente listo en cuanto al uso de bloques funcionales, está disponible el nivel Experto. En este nivel se tienen todos los bloques funcionales disponibles para hacer diversas aplicaciones con Blokino. En la Figura 4.10.1.5 se muestra la sección experta de Blokino

Figura 4.10.1.5 - Sección de nivel experto

Cada desafío contiene pruebas anticipatorias que deben cumplirse para completar el uso de los bloques funcionales. Dependiendo del desafío, las pruebas pueden ser 1, 2 o 3. Todas deben cumplirse para completar el desafío.
Cada desafío contiene una descripción de los elementos que se usarán en el desafío. En la Figura 4.10.1.6, se muestra la descripción del desafío de los LEDs.

Figura 4.10.1.6 - Panel de descripción de los desafíos

En el extremo izquierdo se encuentra el panel “Instrucciones” que describe la consigna del desafío en un estilo guiado. La Figura 4.10.1.7 muestra el panel de instrucciones del desafío de Leds.

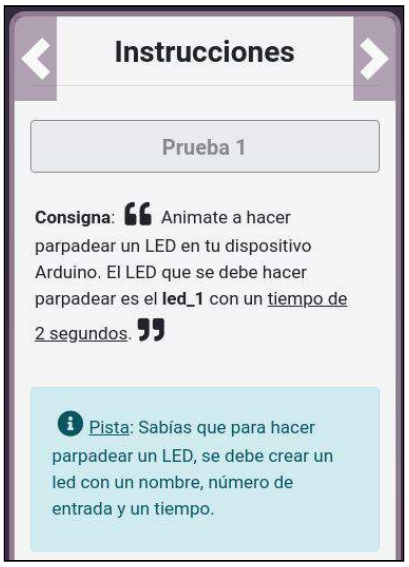


Figura 4.10.1.7 - Panel de instrucciones del desafío de Leds

Dependiendo del desafío que se haya seleccionado, el workspace va a cargar los bloques funcionales que se deban usar. De esta manera se acota la cantidad de bloques funcionales disponibles para la prueba, la intencionalidad es presentar al usuario solo los bloques que se necesitan para resolver el desafío con la intencionalidad de promover un aprendizaje gradual. La Figura 4.10.1.8 muestra un ejemplo de bloques funcionales disponibles en el desafío de los Leds.

Figura 4.10.1.8 - Panel de bloques funcionales

Entre los bloques funcionales que se muestran en la Figura 4.10.1.8 se pueden observar los bloques funcionales de Crear LED y Parpadear, que son los principales para manipular los Leds.

4.10.1.1 Diagramas de componentes electrónicos

Para cumplir con los desafíos es necesario armar los circuitos o componentes electrónicos asociados y así poder ejecutar los programas de Blokino sobre estos componentes. Para ello Blokino ofrece diagramas que ayudan a realizar este armado. En la Figura 4.10.1.1.1 se muestra cómo armar el circuito del desafío de los leds.

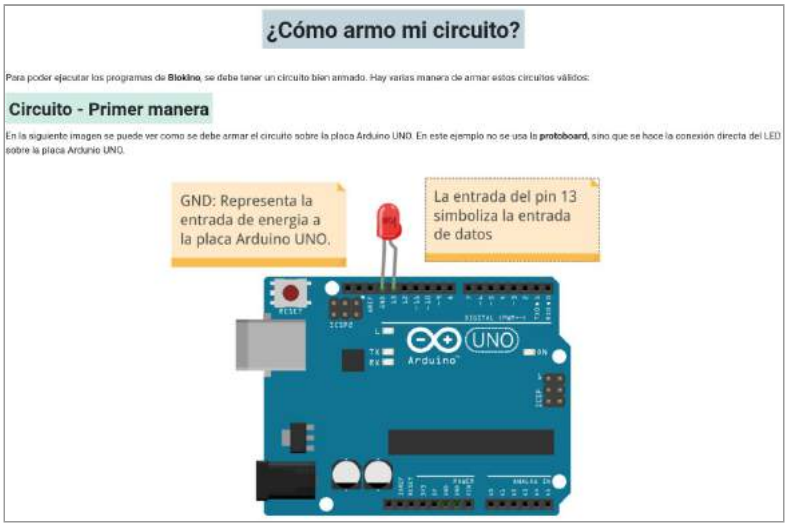


Figura 4.10.1.1.1 - Armado del circuito del desafío de los leds

Así como los programas de Blokino se pueden resolverse de distintas maneras, los circuitos también pueden ensamblarse de distintas formas. Algunos desafíos tienen más de un diagrama para que los puedan probar con sus respectivas soluciones. En la Figura 4.10.1.1.2 se muestra otro diagrama para completar el desafío de los leds.

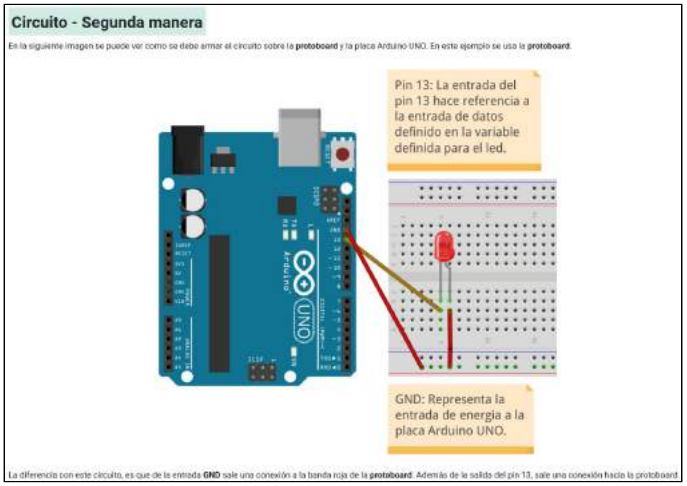


Figura 4.10.1.1.2 - Armado del circuito del desafío de los leds

4.10.2 Programemos

En esta sección de la plataforma se encuentra el modo libre de Blokino. En este modo se tienen disponibles todos los bloques funcionales y sus bloques complementarios de la plataforma. En la Figura 4.10.2.1 se muestran los bloques funcionales disponibles.

Figura 4.10.2.1 - Los bloques funcionales del modo libre de Blokino

Contar con todos los bloques funcionales los programas que se pueden crear con Blokino son más complejos y completos.
En el modo libre se agregó la consola de mensajes de Blokino. Con esta consola se puede capturar y mostrar los mensajes asincrónicos de los child process que se ejecutan en la plataforma. En la figura 4.10.2.2, se muestra la consola de mensajes de Blokino.

Figura 4.10.2.2 - Consola de mensajes asincrónicos

Para poder capturar los mensajes asincrónicos se usan canales de comunicación asincrónicos bidireccionales. Estos canales de comunicación son provistos por Electron, y configurados desde la plataforma.
De esta manera, todo lo que el estudiante fue aprendiendo en los desafíos se puede aplicar, como así también puede mejorar sus programas Blokino, dado que acá se pueden usar todos los tipos de bloques funcionales que soporta la plataforma.

4.10.3 Robots

Para que los estudiantes puedan aplicar Blokino con varios componentes electrónicos, se diseñaron NodeBots adaptados para funcionar con la plataforma. De esta manera en el

modo experto de Blokino se pueden usar para encender Leds, crear movimiento y acciones en los NodeBots.

4.10.3.1 NodeBots

Dentro del mundo IoT, la robótica es una de las áreas que mayor atención ha despertado dentro de los amantes de la programación y la tecnología. El uso del hardware libre como Arduino o Raspberry Pi combinado con tecnologías como Node.js, ha llevado a usar JavaScript a otra área anteriormente dominada por lenguajes como Java, C, C++ y Processing.
El armado de circuitos y uso de componentes electrónicos con JavaScript, llevó a que a mediados del 2013 se creará la comunidad de desarrolladores llamada NodeBots. Está comunidad está compuesta por varios países cómo: Colombia, Brasil, EEUU, México y Nueva Zelanda.
Los NodeBots usan Node.js, el cual permite ejecutar un servidor local, dónde se puede usar la librería de robótica que se requiera. Entre las librerías más usadas están: Johnny-Five, Cylon.js, Node-Red, JerryScript y Node-serialport entre otros.
Estas librerías son usadas en conjunto con Node.js, y fueron apareciendo a medida que la comunidad de NodeBots fue creciendo. Los primeros programadores que dieron los primeros pasos en JavaScript Robotics, fueron Nicolai Onken y Jorn Zaefferer. Durante la conferencia JSConfEU del 2010, posicionaron a JavaScript donde otros desarrolladores especializados en robótica ya lo situaban: es posiblemente la sintaxis que con menos esfuerzo y líneas de código puede hacer realidad el llamado IoT. En la Figura 4.10.3.1.1 se muestra el código que se expuso en la conferencia JSConfEU donde se le da un porcentaje de brillo a un elemento del DOM que tenga el identificador **“lights”**.

```
$("livingroom").bind("motion", function() {
  $(this).find("lights").brightness("75%").dimAfter("120s");
});
```

Figura 4.10.3.1.1 - Código ejemplo de la JSConfEU 2010

Otro de los investigadores de NodeBots es Chris Williams, que aquel día de 2010 en la JSConf fue uno de los asistentes de la charla de Onken y Zaefferer. Años después desarrolló node-serialport, que permite interactuar con microcontroladores mediante Node.js.
Otros programadores que hicieron aportes a la comunidad de NodeBots y Javascript Robotics, fueron Julian Gautier y Rick Waldron. Julian Gautier creó el protocolo Firmata, que permite la comunicación con microcontroladores a través de un software de ordenador, tableta o móvil. Hoy en día, el protocolo Firmata es el más completo para la plataforma Arduino. Firmata tiene varias bibliotecas de cliente en varios lenguajes además de JavaScript, como: Processing, Python, Ruby, Perl, Clojure, Java y PHP entre otros. Por otro lado está Rick Waldron, que tomó el protocolo Firmata y lo convirtió en un marco de desarrollo de robótica en JavaScript e IoT llamado Johnny-Five. Este framework permitió a los desarrolladores en NodeBots dotar a sus nodebots funcionalidades como: movimiento, control de luces y emisión de sonidos, usando

comandos reservados para los componentes electrónicos que se quieran usar en los programas de Johnny-Five.

4.10.3.2 M14

Durante el desarrolló de Blokino se planteó que además de los desafíos, se contase con un NodeBot que incluya distintos componentes electrónicos, con el objetivo de:

- Comprender cómo armar los circuitos, sin generar interrupciones sobre la placa Arduino.
- Armar los programas de Blokino, combinando los bloques funcionales de los componentes electrónicos con los bloques que permiten capturar eventos.

4.10.3.2.1 Versiones

Durante el desarrollo de los NodeBots se realizaron distintas versiones antes de llegar a las dos versiones finales de fibrofacil y plástico impreso. Las versiones previas se basaron en el uso de carton, bandas elásticas y cinta adhesiva. La Figura 4.10.3.2.1.1 muestra el primer NodeBot hecho con cinta adhesiva y una caja de cartón.

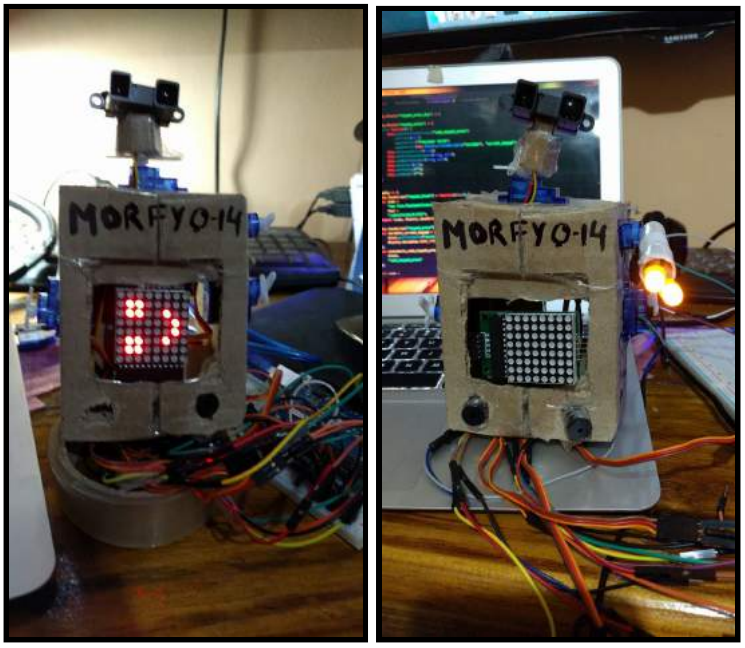


Figura 4.10.3.2.1.1 - Primer NodeBot hecho con un caja de cartón y cinta adhesiva

Este NodeBot se armó con una caja de cartón común de víveres y algunos componentes electrónicos compatibles con Blokino. Estos son los componentes electrónicos que se usaron:

- 2 Buzzers para emitir sonidos.
- 1 sensor de proximidad para detectar objetos.
- 3 servomotores para mover las extremidades.
- 2 Leds para emitir parpadeos.
- 1 Matriz-LEDs para dibujar figuras y mensajes.

A diferencia de los M-14. este NodeBot no puede movilizarse dado que no posee extremidades inferiores. Los problemas encontrados con esta primera versión fueron los siguientes:

- Al no tener movilidad, tenía pocas acciones a realizar con los componentes electrónicos.
- El cartón es muy frágil, con el movimiento constante de los brazos y la cabeza se debió agregar más cinta adhesiva para lograr estabilidad.

La Figura 4.10.3.2.1.2 muestra el segundo NodeBot realizado. A esta versión de NodeBot se la agregaron extremidades inferiores y superiores.

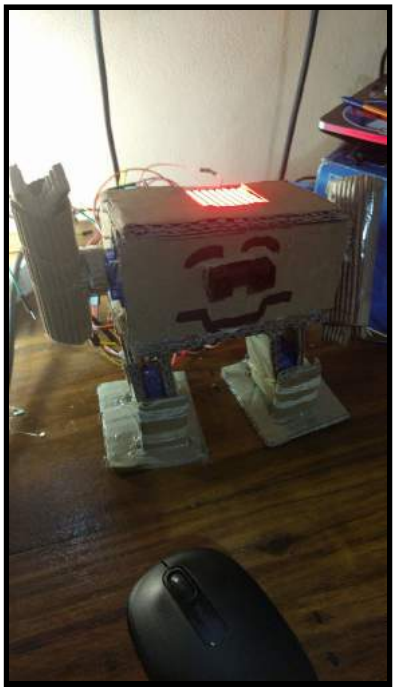


Figura 4.10.3.2.1.2 - Segundo NodeBot.

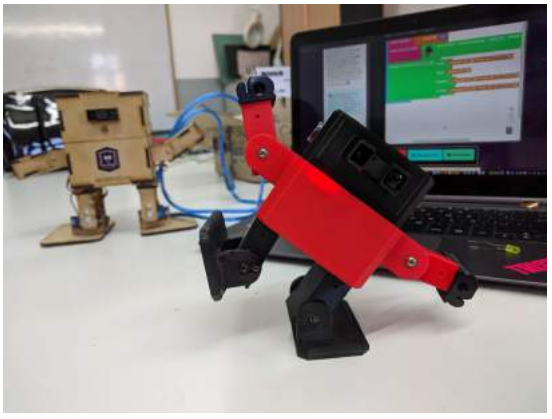
A diferencia de la versión anterior, este NodeBot puede moverse pero con dificultad. Se usaron pedazos de cartón más gruesos para conseguir estabilidad y firmeza. Con los constantes movimientos el cartón comenzó a ceder su contextura, por tanto el NodeBot terminó perdiendo estabilidad. Para solucionar este problema se buscó otra opción, se recurrió a armarlos los NodeBots con fibrofacil. El fibrofacil está compuesto por varias capas de cartón comprimidas, lo que dió esa contextura necesaria para que el NodeBot no pierda la estabilidad al moverse. Esta versión incluye lo siguiente:

- Una placa Arduino UNO: la placa electrónica que se usa para conectar los componentes electrónicos y cables duponts.
- 1 mini protoboard: que sirve como extensión, para el armado del circuito.
- Cable usb: para conectar la placa Arduino UNO con la computadora.
- 6 servomotores: para mover las extremidades del NodeBot.
- 1 sensor de proximidad: para detectar 4 tipos de distancias, soportadas por Blokino.
- 1 buzzer de sonidos: para emitir sonidos simples o compuestos.
- 1 matrix-leds: útil para generar figuras, mensajes u emoticones personalizados.

Esta versión contiene la placa arduino UNO, debido al tamaño del NodeBot. Al ser más grande, se usó la protoboard para poder conectar todos los componentes electrónicos.
Por otro lado se construyó otra versión del M-14 con plástico impreso con una impresora 3D. A diferencia de la versión de fibrofacil, este NodeBot está compuesto por:

- Una placa Arduino NANO: la placa electrónica que se usa para conectar los componentes electrónicos.
- Un adaptador Nano Shield: debido a que el Arduino Nano es demasiado pequeño para conectar en forma simple los cables dupont, módulos, sensores, etc. Este adaptador hace fácil llevar a cabo todas las conexiones, como así también para poder alimentar al Arduino con cualquier fuente de 7.5V a 12V.
- 6 servomotores: para mover las extremidades del NodeBot.
- 1 sensor de proximidad: para detectar 4 tipos de distancias, soportadas por Blokino.
- 1 buzzer de sonidos: para emitir sonidos simples o compuestos.

Esta versión contiene el Arduino Nano y el adaptador, que logró simplificar el armado de circuitos ofreciendo una versión más compacta, extensible y ordenada al armar los circuitos del NodeBot. La Figura 4.10.3.2.1.3 muestra las dos versiones de los NodeBots.



4.10.3.2.1.3 - NodeBots de fibrofacil y plástico

En la Figura 4.10.3.2.1.4 se puede ver la sección de los NodeBots donde se da una breve descripción del NodeBot M-14, como así también de las distintas versiones y modo de armado.

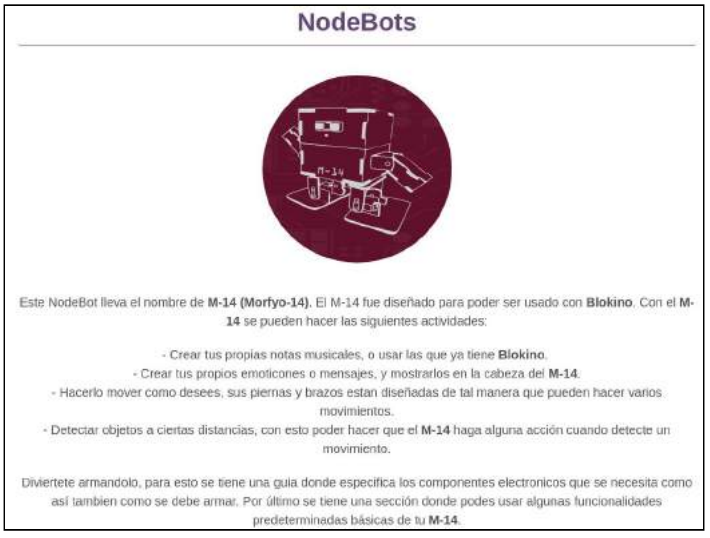


Figura 4.10.3.2.1.4 - Pantalla con la descripción de los NodeBots

4.10.4 La web de Blokino

Para poder acceder a las novedades y mejoras de Blokino se cuenta con un sitio web. Dentro del mismo se pueden encontrar las mejoras de la plataforma, como así también la documentación completa sobre el uso los bloques funcionales. Para acceder al sitio web de Blokino se debe accionar el botón Web de la plataforma. En la Figura 4.10.4.1, se muestra la portada principal de la web de Blokino.

Figura 4.10.4.1 - Pantalla de inicio de la página web de Blokino

Capítulo 5 - Evaluación y difusión

Se realizaron pruebas de campo con Blokino con estudiantes de 4to año de 2 escuelas secundarias técnicas con especialidad informática: la EEST N° 2 “Ing. Rebuelto” de Berisso y EEST N° 5 ”General Manuel N. Savio” de Villa Elvira, La Plata. Participaron de las mismas un total de 14 estudiantes. La primera de las pruebas se realizó en la escuela, en la EEST N° 2 de Berisso, y participaron 6 estudiantes y la segunda se realizó en la Facultad de Informática, con 8 estudiantes de la EEST N° 5 de Villa Elvira.
A su vez, se hicieron muestras de Blokino en 2 espacios profesionales: un *meetup interna* en la empresa Globant, en la cual me desempeño como programador full stack, y otra *meetup* para la comunidad platense de JavaScript.

5.1 - Objetivos

Durante las pruebas de campo se establecieron objetivos que permitieran contar con una primera evaluación de la experiencia de las y los estudiantes con el uso de Blokino. A continuación se describen los objetivos de la evaluación.
Lograr que las y los estudiantes puedan:

- Navegar por la plataforma Blokino y reconocer las secciones que la componen.
- Identificar los componentes electrónicos que componen el kit de Blokino.
- Comprender los usos de los componentes electrónicos y las combinaciones que se pueden realizar con ellos.
- Resolver los desafíos de cada nivel propuestos por Blokino. Dentro de los desafíos se encuentran enunciados con consignas detalladas y guiadas sobre cómo abordar el problema planteado. A medida que va incrementando la dificultad de los desafíos, los enunciados hacen referencia qué se debe resolver y no cómo hay que hacerlo. El objetivo es que las y los estudiantes puedan apropiarse de lo aprendido completando los desafíos de manera incremental en cuanto a la dificultad planteada.
- Programar en el modo “experto” de Blokino y de esa manera construir sus propios objetos físicos programados.
- Programar funciones para los NodeBots.

5.2 - Metodología

Previamente al encuentro con los estudiantes se instaló Blokino en las computadoras. Tanto en la escuela como en la Facultad de Informática las computadoras que se utilizaron funcionan con Windows 10.
Una vez instalado Blokino se hicieron pruebas con las placas Arduino para confirmar que todo funcionara correctamente. Las placas Arduino usadas fueron las disponibles en la plataforma Blokino y en el caso de las pruebas en la EEST N° 2 de Berisso se utilizaron también placas proporcionadas por la escuela. Esto sirvió para comprobar que los módulos de firmata trabajan correctamente con distintas placas de la familia Arduino.
A continuación se describe la metodología de trabajo de las pruebas de campo:

- Antes de iniciar la prueba se hizo una breve presentación acerca de Blokino, su objetivo, cómo funciona y en qué consistía la prueba de la que iban a participar, además de agradecerles a las y los estudiantes y a sus docentes por aceptar colaborar en la evaluación de la herramienta. Se explicó que Blokino es una

herramienta desarrollada como parte de una tesina de grado y que me permitiría obtener mi título de Lic. en Sistemas.

- Se propuso a las y los estudiantes que trabajasen en grupos de 2 o 3. Se detectó que algunos estudiantes contaban con mejor manejo de electrónica y programación, por tanto se intentó formar grupos integrados por diferentes perfiles de estudiantes con la intención de que se ayuden y trabajen en equipo.
- Para la resolución de los desafíos las y los estudiantes contaron con mi asistencia tanto para atender consultas sobre las consignas de los desafíos como para el armado de los circuitos electrónicos.
- Se facilitó a las y los estudiantes todos los materiales y herramientas necesarios del kit de Blokino para que puedan realizar los desafíos.
- Se asignaron 20 minutos para completar cada desafío. En este tiempo debían leer y entender la consigna del desafío y resolverlo.
- A los desafíos están clasificados en niveles de complejidad: inicial, intermedio y avanzado.
- El total de desafíos a resolver en estas pruebas fueron 9.
- Antes de comenzar con la resolución de cada uno los desafíos se propuso a las y los estudiantes que prueben los componentes electrónicos. Para ello Blokino proporciona un conjunto pruebas anticipatorias de cada uno de los componentes electrónicos que se usarán en los desafíos.
- Para completar un desafío se deben completar una o más pruebas de funcionamiento de los componentes electrónicos y así conocer y comprender el funcionamiento de los mismos antes de usarlos en el desafío.
- La resolución de los desafíos se diseñó para que el uso de los bloques funcionales se haga de manera incremental. Se comenzó creando variables asociadas a los componentes electrónicos y construyendo circuitos simples en los desafíos más sencillos, y se terminó usando varios componentes a la vez, aplicando procedimientos y armando circuitos donde conviven varios componentes electrónicos, en los desafíos más complejos.
- Luego de completar todos los desafíos, se dejó un tiempo para usar Blokino en modalidad “libre”. En este tiempo usaron los componentes electrónicos para hacer sus propios programas y poner en práctica lo aprendido.
- Se entregaron NodeBots, para que puedan manejarlos mediante programas de Blokino.

Finalmente, luego de completar las pruebas, se administró una encuesta acerca de la experiencia de usar Blokino, cuyos resultados fueron utilizados como para incorporar mejoras tanto funcionales como visuales.

5.3 - Instalación de Blokino

Durante la instalación de Blokino en las computadoras no se encontraron complicaciones. Se observó que los controladores de Arduino no daban la información correcta para poder ver los dispositivos Arduinos conectados a la computadora, debido a que las placas Arduino que proveyeron las escuelas eran de marcas distintas a las que soporta Blokino. Las placas Arduino compatibles con Arduino deben ser de marca Atmel,

Intel o cualquier placa Arduino con microcontroladores AVR[12]. Esto se debe a que las librerías usadas para configurar las placas Arduino, sólo reconocen y configuran este tipo de placas mencionadas.
Las librerías usadas para gestionar las placas Arduino conectadas a la computadora, son del proyecto de código libre llamado Gort y Serialport de NPM. Con estas librerías se obtiene la información de los dispositivos conectados, pero depende mucho de la computadora donde se conecte y del sistema operativo que tenga, debido a que algunos sistemas operativos debido a su política de permisos no brindan información completa de los dispositivos conectados mediante puertos USB, éste es el caso de Windows.
Esta actividad de instalación de Blokino sirvió para generar una mejora a futura que consistirá en implementar un módulo propio para la detección de hardware conectado por puerto usb. La instalación la debe realizar el docente del curso o un referente técnico de la escuela, dado que se requiere por ejemplo:

- En el caso de Windows instalar primero los controladores AVR, y luego Blokino
- Y en el caso Linux instalar Arduino por comandos de consola o usando el código fuente que provee la web oficial de Arduino.

5.4 - Introducción de Blokino

Para que los estudiantes puedan conocer la plataforma en el tiempo estipulado para las pruebas, se hizo una breve introducción de Blokino y los componentes electrónicos que se pueden usar. En esta introducción se respondieron las siguientes preguntas:

- ¿Qué es Arduino?
- ¿Qué son los componentes electrónicos?
- ¿Qué es Blokino?
  - ¿Para qué lo puedo usar?
  - ¿Qué son los bloques funcionales?
  - ¿Qué es JavaScript ?
- ¿Qué son los desafíos?
  - ¿Cómo los completo?
  - ¿Qué logró haciéndolos?
- Los NodeBots de Blokino
  - ¿Cómo los puedo usar?
  - ¿Cómo puedo hacer mis propios robots?

Concluida la introducción el grupo de estudiantes se encontraba en condiciones de usar Blokino y comenzar a resolver los desafíos propuestos.

5.5 - Resolver desafíos

De acuerdo a lo comentado en el apartado metodología, luego de la introducción de Blokino, se propuso a las y los estudiantes que trabajen en grupo intentando conformar equipos en los que participarán estudiantes con más manejo y conocimientos de placas Arduino y programación, con estudiantes con menos experiencia. De esta manera se logró armar grupos variados y equilibrados, que pudiesen colaborar y ayudarse en la resolución de los desafíos.

[12] AVR: son los microcontroladores usados en la placas Arduino. Son parte de la familia de microcontroladores RISC del fabricante estadounidense Atmel.

Durante el desarrollo de los desafíos, las y los estudiantes demostraron mucho interés, aceptación al diseño y el modo de uso de Blokino. El objetivo de lo desafíos es que los estudiantes puedan ir aprendiendo a usar Blokino, desde desafíos iniciales sencillos a desafíos complejos, dónde se usan por ejemplo varios componentes electrónicos funcionando conjuntamente para cumplir el desafío.
Todos los participantes pudieron realizar las pruebas desde los niveles iniciales hasta los más complejos.
En los desafíos iniciales, las pruebas a resolver se centraron en la creación de variables, procedimientos y el manejo de los componentes electrónicos cómo **LEDs**, **LEDs-RGB** y **consola de mensajes**. En cada uno de los desafíos se presentaron distintos escenarios:

- En el desafío de los LEDs, se inició con crear variables que identifican a los LEDs, usar bloques funcionales de LEDs y hacer parpadear LEDs con distintas unidades de tiempo. En las últimas pruebas se introdujeron los procedimientos, para que puedan aplicarse y mejorar el código. Este desafío resultó sencillo.
- El desafío de los LEDs-RGB se centró en el armado del circuito dado que en el desafío de los LEDs la complejidad del circuito era muy básico. Las pruebas de este desafío eran similares a los del LED, pero con la diferencia que acá se puede asignar colores. Este desafío aunque resultó simple de resolver, presentó algunas dificultades en el armado del circuito dado que es un componente electrónico que no conocían. Tuvieron que usar la guía de armado de circuito que está vinculada a cada desafío.
- En el desafío de la consola de mensajes se presentó el primer componente electrónico con un módulo acoplado. El módulo acoplado reduce la cantidad de pines usados para el armado del circuito. Este componente electrónico no lo conocían, fue necesario recurrir a la guía para el armado del circuito. Para la resolución de las pruebas no se presentó dificultad.

En términos generales, los desafíos iniciales sirvieron para dar los primeros pasos con Blokino y el uso de los distintos recursos de ayuda e información que tiene la plataforma.
En los desafíos intermedios las pruebas a resolver se centraron en las primeras interacciones entre el usuario y los componentes electrónicos, como así también la emisión de sonidos. En este desafío se usaron los componentes electrónicos del **Joystick** y **Zumbadores.** En estos desafíos se manifestó lo siguiente:

- En el desafío del Joystick se presentó el primer componente electrónico que permite interacción con el usuario. El circuito del Joystick no es complejo, debido a que se usan pocos pines, por tanto en general no fue necesario recurrir a la guía de circuitos. Para completar este desafío se debieron hacer pruebas dónde se usan los conceptos de los desafíos iniciales, pero además en este desafío se comenzaron a mezclar componentes electrónicos, se combina el Joystick con LEDs. Esto requirió un poco más de tiempo y de asistencia para el armado de circuitos. Este fue el primer desafío que presentó cierto grado de complejidad en el armado del circuito de componentes combinados.
- En el desafío de los Zumbadores, se presentó el componente electrónico **Buzzer,** que su funcionalidad se basa en emitir sonidos dependiendo las escalas que se le aplica al bloque funcional. El circuito no es complicado y no se requirió de asistencia.

Los desafíos del nivel intermedio resultaron útiles como primera aproximación a componentes electrónicos con interacción y uso de varios componentes electrónicos en el mismo circuito.

Por último en los desafíos del nivel avanzado se aplicaron los conceptos usados en los niveles previos. Los componentes electrónicos utilizados en este nivel son: **Matrix-LEDs, Servomotores, Sensores de movimiento y sensores de proximidad.**

- En el desafío de Matrix-LEDs se presentó otro de los componentes electrónicos que tiene un módulo incorporado para la reducción de pines. Este desafío no presentó dificultad en las pruebas anticipatorias sin embargo se observó dificultad al momento de armar los circuitos debido a que el módulo incorporado tiene 6 pines con nombres definidos con abreviaturas que generaron confusión al momento de armar los circuitos de las pruebas. Los estudiantes tuvieron que acudir una vez más a la guía de ayuda y pedir asistencia. Durante la resolución de las pruebas del desafío, se los notó muy entretenidos debido a que con este componente electrónico podían crear figuras, emoticones y mensajes.
- En el desafío de los **Servomotores** se realizaron movimientos de estos componentes electrónicos. Las pruebas anticipatorias dentro del desafío no resultaron complejas a pesar de combinar componentes electrónicos. No se observaron inconvenientes en la resolución de las pruebas ni con el armado del circuito.
- En el desafío **Sensores de movimiento** se presentó el primer componente que detecta objetos en un radio no muy amplio. Las pruebas no resultaron complejas, se combinó con el uso de LEDs. El armado del circuito es un poco complicado debido a que el sensor de movimiento tiene pines definidos. Se observó el interés de las y los estudiantes en el funcionamiento del sensor, en cómo captaba sus movimientos y el control de los LEDs.
- En el desafío de los **Sensores de proximidad** al igual que el desafío anterior también se usa un componente para detectar objetos, pero con la diferencia de que se detectan objetos a determinada distancia del componente electrónico. La experiencia previa en el desafío con el sensor de movimiento facilitó la resolución de este desafío.

En estos últimos desafíos se observó que los estudiantes pudieron resolver las pruebas sumándole la posibilidad de aumentar su experiencia al interactuar con los componentes electrónicos y los programas. En la Figuras 5.5.1 se muestra a los estudiantes resolviendo las pruebas de los desafíos.

Finalizados los desafíos se propuso continuar de manera “libre” en la sección de blokino-experto. En esta sección de la plataforma están todos los bloques funcionales de Blokino. En los desafíos se hace un recorte de los bloques funcionales disponibles de acuerdo a la consigna del desafío para enfocarse en la resolución del mismo, pero en la sección blokino-experto las y los estudiantes tuvieron a disposición una gran variedad de bloques. Esto permitió mejorar sus programas, de los distintos niveles, al advertir que disponían de otros bloques funcionales.

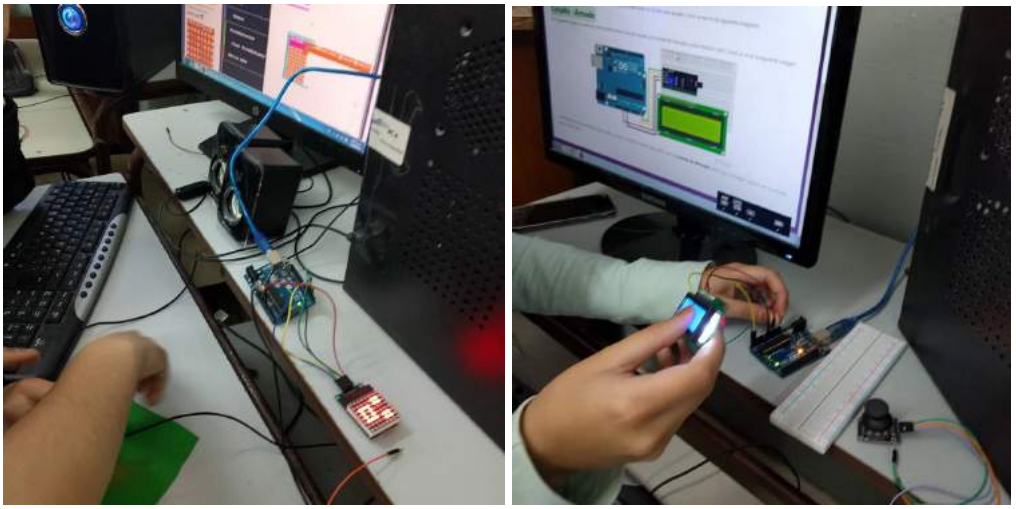


Figura 5.5.1 - Los estudiantes haciendo las pruebas de los desafío

5.6 - Experiencia con NodeBots

Finalmente se ofrecieron los NodeBots llamados M-14. Cada estudiante creó sus propios programas de Blokino usando los componentes electrónicos asociados a los NodeBots. En la Figura 5.6.1 se muestra los NodeBots usados para las pruebas de campo.

Figura 5.6.1 - Los NodeBots usados en las pruebas de campo

La diferencia entre los NodeBots es que cada uno tiene distintos componentes electrónicos, como así también es diferente la placa Arduino que usan. Ambos NodeBots cuentan con servomotores para mover las extremidades y un sensor de proximidad para detectar objetos.
Estas son las funcionalidades que pueden hacer los NodeBots:

- Mover pies.
- Mover piernas.
- Mover brazos.
- Hacer bailar en el lugar.

- Caminar hacia adelante.
- Caminar hacia atrás.
- Caminar hacia los lados.
- Moverse en el mismo lugar.
- Hacer sonar el buzzer pasivo.
- Mover un brazo cuando se detecta objetos cercanos o lejanos.
- Creación de emoticones con la matriz-leds.

El NodeBot más chico, de color rojo y negro, usa una placa Arduino Nano y un conector para pines, y el construido con fibrofacil usa una placa Arduino UNO. Esta diferencia no es tan radical, debido a que la placa Arduino Nano es la versión compacta de la placa UNO. Otra diferencia es que el NodeBot más grande tiene una Matriz-LEDs que permite dibujar figuras.

Las y los estudiantes se interesaron al advertir que Blokino se puede usar para crear proyectos propios, usando elementos que están a su alcance, como por ejemplo el uso de cartón para unir componentes electrónicos. Las primeras versiones de los NodeBots que se hicieron para Blokino fueron construidas con cartón de distintos grosor y cinta para unir las piezas, hasta que quedaron las últimas versiones de fibrofacil y plástico impreso con una impresora 3D.

Además de los movimientos que se pueden hacer con los NodeBots también se mostró cómo se los había armado. En la Figura 5.6.2 se muestra a las y los estudiantes programando con Blokino sobre el NodeBot.

Figura 5.6.2 - Estudiantes probando sus programas en el NodeBot

Las y los estudiantes pudieron observar que la placa Arduino del NodeBot de plástico es diferente que la del construido en fibrofacil. Debido a las dimensiones del NodeBot se eligió una placa Arduino NANO. Este tipo de placas son ideales para espacios pequeños, además funcionan de igual manera que una placa Arduino UNO. La desventaja de esta placa es que es más complicado conectar pines, por lo que se recurrió a usar un adaptador de placas Nano. Este adaptador facilita la conexión de la placa Arduino Nano. Los estudiantes no conocían el concepto de “shield” aplicado a Arduino. Luego de entender cómo se usa esta extensión, pudieron hacer el circuito del NodeBot. En las 2 pruebas de campo esta última actividad les resultó muy entretenida, dado que pudieron hacer varias actividades con ambos NodeBots de manera libre.

5.7 - Encuesta

Finalizadas las jornadas de la prueba de campo se administró una encuesta para obtener información sobre la experiencia de usar Blokino con los componentes electrónicos y los NodeBots. La encuesta consistió en 6 preguntas de respuesta cerrada y 3 de respuestas abiertas. Además se reservó un espacio para dejar comentarios sobre la experiencia de usar Blokino.

5.7.1 - Preguntas

La cantidad de estudiantes que participaron de las pruebas de campo, fue un total de 14 estudiantes. De la Figura 5.7.1.1 se puede concluir que la experiencia de uso de Blokino resultó muy buena, el 90% de las y los estudiantes eligieron la opción “Me encantó” evidenciando un algo grado de satisfacción.

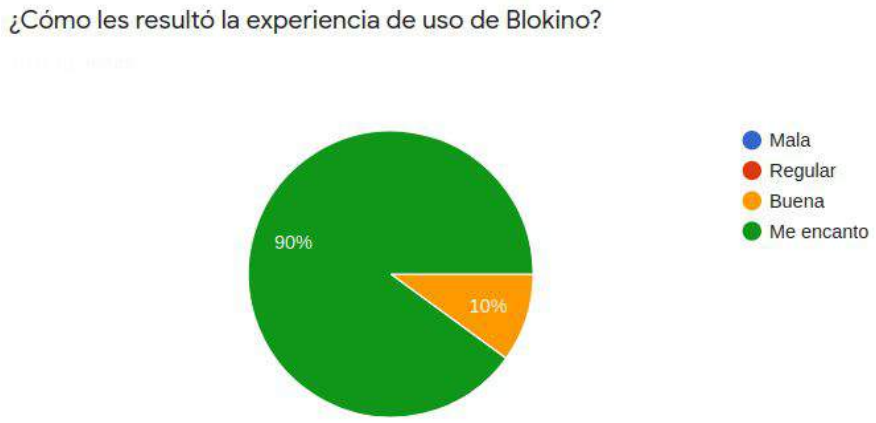


Figura 5.7.1.1 - Encuesta de Blokino

La Figura 5.7.1.2 muestra que la totalidad de los estudiantes pudieron completar los desafíos. Durante el desarrollo de las pruebas se presentaron diferentes dudas, por ejemplo el armado de circuitos o explicaciones sobre las consignas, que pudieron ser atendidas. Por otro lado, el criterio de organización de los equipos colaboró en el desarrollo de la prueba y en que todos pudiesen completar las actividades.

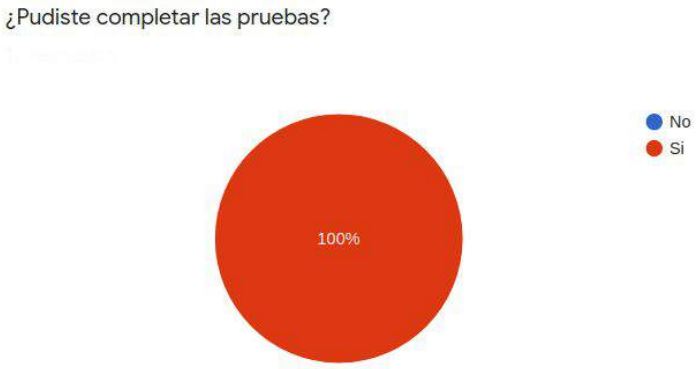


Figura 5.7.1.2 - Encuesta de Blokino

En la Figura 5.7.1.3 se muestra la precepción de las y los estudiantes sobre el grado de dificultad de los desafíos. Para el 70% de los estudiantes el desafío de matriz leds resultó el más complejo, debido al armado los circuitos. A pesar de haber elegido un módulo integrado que acota la cantidad de pines necesarios para el armado del circuito, ésto no fue suficiente para simplificar la construcción de los circuitos. Otro de los componentes en la misma situación es la pantalla LCD.
Los demás componentes que aparecen en el gráfico de tortas como complejos, lo fueron debido a que los enunciados de los desafíos no resultaron del todo claros. Este gráfico aporta a la mejora de Blokino.

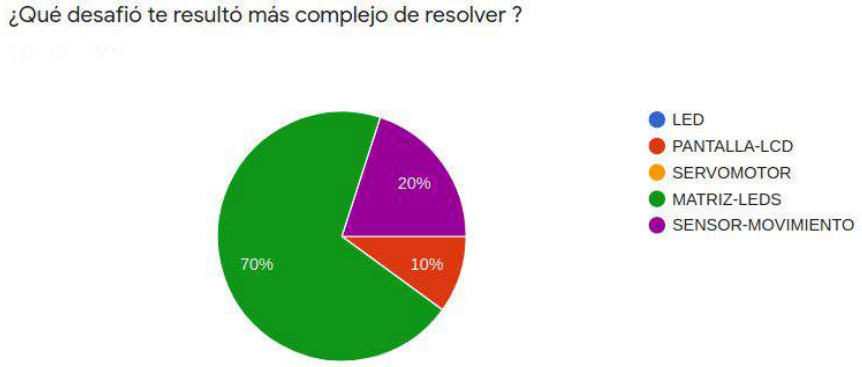


Figura 5.7.1.3 - Encuesta de Blokino

Entre los comentarios recibidos acerca de lo que resultó complejo de los desafíos, se destaca el armado de los circuitos. A continuación se transcriben las expresiones de las y los estudiantes: *"...mejor explicación de la conexión de los componentes"*, *"...algunos circuitos son confusos"* y "...*muchos cables dupont*". Durante las pruebas de los desafíos se dio soporte constante, pero de igual manera el armado de los circuitos resultó complicado al principio.
De las respuestas obtenidas en la pregunta de la Figura 5.7.1.4, se puede observar que todos los estudiantes comprendieron el armado de circuitos, sin embargo no todos lograron hacerlo funcionar. Necesitaron soporte en la resolución de los desafíos, entre las consultas más habituales se registraron las siguientes:

- El uso de resistencias para evitar quemar los componentes electrónicos.
- Cómo usar las protoboards, para poder extender las soluciones de los desafíos.
- Para qué sirven las bandas y secciones de las protoboards.

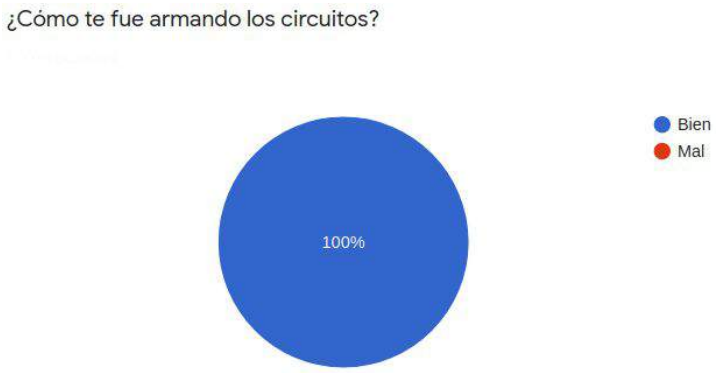


Figura 5.7.1.4 - Encuesta de Blokino

La Figura 5.7.1.5 hace referencia al uso de los NodeBots de Blokino. Todos los estudiantes pudieron utilizarlos, tanto para hacerlo mover como generar sonidos y crear emoticones. Durante esta etapa de las pruebas de campos se mostraron algunos programas de ejemplo que sirvan de guía para poder usar los NodeBots, aunque algunos estudiantes no la necesitaron dado que a esa altura de la prueba habían aprendido a usar Blokino de manera libre.

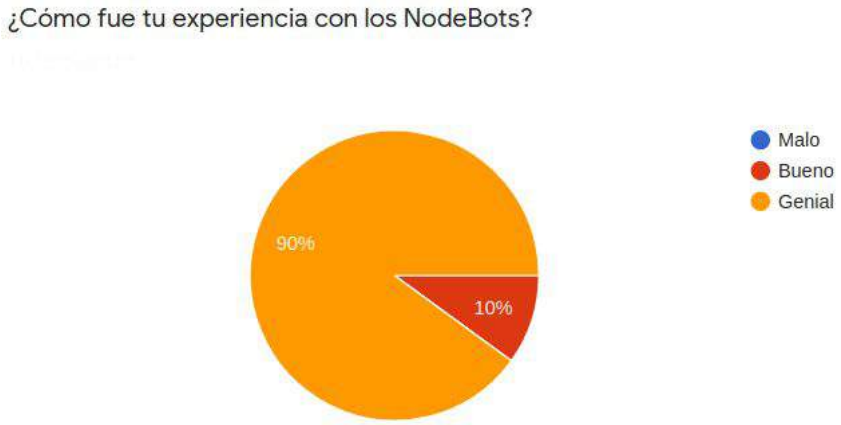


Figura 5.7.1.5 - Encuesta de Blokino

5.6.2 - Comentarios

En la parte final de la encuesta se reservó una sección para dejar comentarios de libre expresión. La mayoría de los comentarios fueron alentadores y constructivos. Entre los comentarios alentadores se pueden citar los siguientes:

"Me gustó, puede ser usado por alguien que no posee muchos conocimientos debido a que es intuitiva y tiene una interfaz amigable, clara y sencilla." (Alumna de EEST N° 2)

"Muy buena y fácil de aprender con los bloques" (Alumno de EEST N° 2)

"Nada para mejorar, me parece que está muy completo" (Alumna de EEST N° 2)

"Fue una experiencia genial y muy útil" (Alumna de EEST N° 5)

"Fue una muy buena experiencia, porque usé circuitos que no había usado antes" (Alumno de EEST N° 5)

"Fue una experiencia divertida e informativa" (Alumna de EEST N° 5)

"Excelente experiencia, fácil de aprender" (Alumno de EEST N° 5)

"Fue muy buena" (Alumno de EEST N° 5)

Entre los comentarios con propuestas constructivas se pueden citar los siguientes:
“Cada vez que se completa una serie de desafíos, no se pueden leer bien los botones inferiores del cartel que salta.” (Alumna de EEST N° 2)
“Que sea más dIfícil” (Alumna de EEST N° 2)
“Que haya más información para el armado de los circuitos.” (Alumno de EEST N° 2)
“Diferentes maneras para completar los ejercicios” (Alumno de EEST N° 5)
“Mejor explicación de la conexión de los componentes” (Alumno de EEST N° 5)
“Muchos cables dupont” (Alumno de EEST N° 5)
“Algunos circuitos son confusos” (Alumno de EEST N° 5)
“Que haya más cantidad de circuitos” (Alumna de EEST N° 5)
Ambos tipos de comentarios sirvieron para mejorar la plataforma. Luego de cada prueba de campo se revisaron los comentarios de las encuestas para corregir y mejorar la plataforma Blokino.

5.7 Difusión

Para obtener otros puntos de vista de Blokino se realizaron *meetups* con distintos grupos de programadores platenses. Blokino podría ser una herramienta a utilizar en ámbitos educativos no formales, como clubes de robótica o simplemente para personas interesadas.

5.7.1 Charlas internas

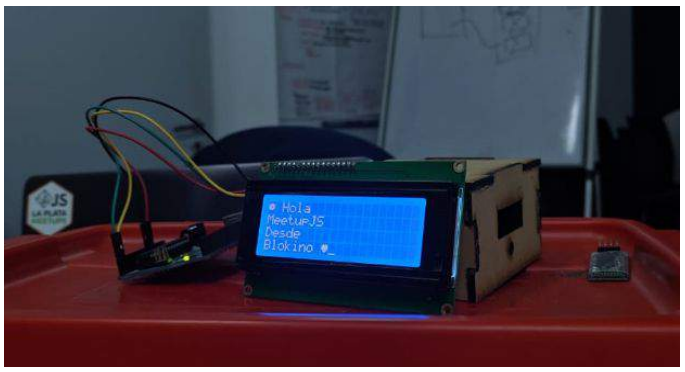
Para poder dar a conocer Blokino, en la empresa Globant que es dónde trabajo como programador, me ofrecieron un espacio para realizar una charla técnica. Esta charla se enfocó en la presentación de Blokino y el uso de JavaScript para controlar objetos electrónicos. Durante la charla se hizo una presentación técnica sobre cómo fue construido Blokino, se describió Electron y JavaScript. La devolucion fue satisfactoria, la comunidad interna de programadores desconocía que con JavaScript se pueden manipular componentes electrónicos y que no es necesario usar Processing o C/C++, siendo ésta una devolución interesante.

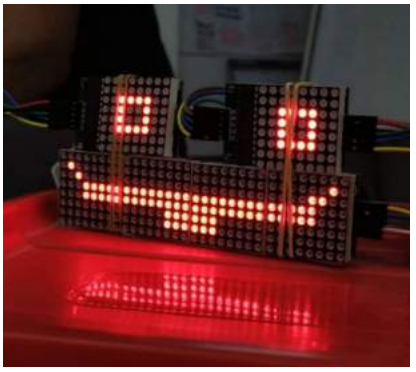
5.7.2 Comunidad Javascript platense

Otro ámbito donde se mostró Blokino fue en la *meetup* de la comunidad JavaScript de La Plata, llamado LaPlataJS. En este caso se trata de un evento organizado por programadores apasionados por JavaScript, que surgió hace varios años en La Plata. Cada año las charlas se van realizando por distintos miembros de la comunidad sobre temas de diversas plataformas usando JavaScript, cómo puede ser programación web, mobile, cross-mobile, técnicas de integración, unit testing y desarrollo de aplicaciones nativas.
La comunidad otorgó el espacio para presentar Blokino y los NodeBots. La presentación consistió en una charla técnica, se hicieron muestras de cómo funcionan los NodeBots y otros ejemplos de cómo usar Blokino combinando distintos componentes electrónicos. En la Figura 5.7.2.1 se muestra una foto de la *meetup* de Blokino realizada el 6/12/2019.

Figura 5.7.2.1 - Convocatoria de la charla de Blokino

La *meetup* se realizó en el espacio de la Municipalidad de La Plata llamado **"La Plata Emprende, Crowdworking La Catedral"**. Este espacio tiene como objetivo otorgar un espacio de trabajo para iniciativas que necesiten un lugar donde desarrollar su idea. En ese sentido propone el desarrollo de proyectos locales y ofrece aulas, espacios de crowdworking, SUM, mobiliario y tecnología para potenciar propuestas tecnológicas. La comunidad de LaPlataJS gestionó un salón de este espacio para desarrollar la *meetup*.
Durante esta ***meetup*** se desarrollaron algunos ejemplos con los componentes electrónicos, para que la comunidad pudiera apreciar que cosas se pueden hacer.
La charla se planificó para una hora, sin embargo debido a la cantidad de preguntas se extendió a 2. Se realizaron consultas sobre cómo se manejó la arquitectura y el control del hardware, para ello se realizaron diagramas y explicaciones teóricas detalladas.
Al finalizar la charla la comunidad de LaPlataJS demostró interés con Blokino y propuso la publicación oficial en la página web, para poner disponible Blokino.
La siguientes fotos muestran algunos momentos de la charla de Blokino a la comunidad LaPlataJS.



5.7.3 Metodología de las charlas

Durante ambas charlas se desarrollaron los siguientes temas:

- Tecnologías implementadas.
- La arquitectura de Blokino.
- Introducción sobre el armado de los circuitos.
- Distintas pruebas con los componentes electrónicos.
- Demostraciones del funcionamiento de Blokino, con los robots m-14.
- Mejoras y nuevas funcionalidades planificadas para Blokino.
- Brainstorming para aplicar a Blokino.

Finalmente se recogieron las siguientes ideas:

- Implementar módulos independientes para ganar en escalabilidad.
- Obtener más información de los dispositivos conectados.
- Desarrollar un manejador de errores para circuitos mal implementados.
- Ampliar la guía de armado de circuitos complejos.

Las charlas lograron acercar Blokino a las diferentes comunidades de desarrolladores platenses. Blokino fue bien recibido, acompañado de varios comentarios haciendo referencia a lo sencillo que fue crear programas para ejecutar sobre las placas Arduino, sin tener la experiencia en el armado de circuitos y creación de programas para Arduino

5.8 - Conclusiones

Luego de realizadas las prueba de campo y las *meetups* de Blokino, se pudieron recolectar devoluciones que sirvieron para corregir y mejorar la plataforma.
A continuación se sistematizan las devoluciones:

- Las pruebas de Blokino realizadas en la EEST N° 2 de Berisso y en la Facultad de Informática de La Plata con estudiantes de la EEST N° 5, permitieron comprobar que Blokino puede usarse con las placas de la familia Arduino que cuentan las escuelas. De esta manera podría ser una plataforma para poner a disposición de las escuelas y que pueda colaborar en la enseñanza de programación y electrónica.
- Durante las pruebas de campo la problemática que tuvieron varios estudiantes fue el armado de los circuitos con distintos componentes electrónicos. Esto debe mejorarse, para que no sea un impedimento para usar Blokino.
- A pesar que las y los estudiantes trabajaron en grupo y que algunos tenían conocimientos de programación, necesitaron ayuda para resolver algunas pruebas.
- La resolución de los desafíos simples y guiados, presentados en niveles de dificultad gradual, permitió a las y los estudiantes entender el funcionamiento de los componentes electrónicos, cómo armar los circuitos y cómo programarlos. El enfoque de ciclos espiralados favoreció el proceso aprendizaje. Esto se evidenció cuando los estudiantes usaron los NodeBots, no manifestando dificultades para programarlos con Blokino.
- La experiencia con los NodeBots fue muy bien recibida, debido a que su aspecto de robot llamó la atención y despertó más aún el interés de crear programas para moverlos o que haga otra acción usando los componentes electrónicos que tienen incorporados.
- Los distintos comentarios sirvieron para mejorar la plataforma, dado que se mencionaron detalles tanto estéticos como funcionales que no se habían controlado en la versión usada en las pruebas de campo.
- Las difusiones de Blokino sirvieron para poder llegar a entornos profesionales. Despertó interés y llamó atención de programadores. Este tipo de público manifestó consultas con respecto al rendimiento y la estructura de Blokino que resultan interesantes para considerar en futuras versiones.

Capítulo 6 - Problemas encontrados

Durante el desarrollo de la plataforma Blokino se presentaron inconvenientes de hardware y software, que resultan relevantes describir para ser abordados en futuros trabajos.

Problemas de hardware:

- **Detección de dispositivos**: para poder detectar los dispositivos Arduino y configurarlos, primero se debe detectar el dispositivo que se conectó a las entradas usb de la computadora. Para poder detectarlos se usaron librerías de NPM (Node Package Manager) de las que hay poca documentación. Al no contar con la información de cada placa Arduino conectada, fue necesario desarrollar funciones personalizadas para armar la información necesaria para configurar el dispositivo. No se llegó a completar esta solución: por un lado en Linux se tiene más control de permisos para obtener información de dispositivos que en Windows, y esto hizo posible en Linux detectar el modelo de cada placa conectada, como así también ignorar cualquier otro dispositivo que no sea Arduino. Sin embargo en Windows no se logró generar un listado de dispositivos con el tipo de placa conectada.
- **Compatibilidad con las placas Arduino**: para poder ejecutar las aplicaciones de Blokino se buscó configurar la plataforma Arduino UNO, MEGA y NANO. Durante el desarrollo se buscaron librerías para poder configurar con Blokino las placas, sin usar herramientas externas como por ejemplo Arduino IDE. Esta problemática condujo al desarrollo de diversas alternativas para hacer una configuración en segundo plano de las placas.
- **Instalador de Blokino**: crear el instalador de Blokino en conjunto con todas las librerías necesarias para que la plataforma pueda ejecutarse en los sistemas operativos Windows (x64) y Linux (x64). Para poder resolver esto se usaron scripts personalizados de NPM, que fue necesario testear sobre ambas plataformas. Dejando así scripts estandarizados que son ejecutados antes de crear el instalador correspondiente para la plataforma.
- **Selección de dispositivos físicos**: durante el desarrollo se probaron distintos componentes electrónicos para ejecutarse con la librería Johnny-Five. Se buscaron componentes que tengan poca complejidad en cuanto a la conexión sobre las placas Arduino. De esta manera se logró armar un kit de componentes electrónicos simples para armar los esquemas. Para reducir la complejidad aún más se hicieron tutoriales usando imágenes descriptivas para el armado de los circuitos.

Problemas de software:

- **Versión inestable de Electron**: durante el desarrollo de la plataforma se probaron distintas versiones de Electron compatibles con la librería de serialport. Ésta es importante porque es usada por Johnny-Five. Para lograr la compatibilidad con la versión de Electron se tuvieron que hacer modificaciones en los scripts personalizados de la aplicación.
- **Validación del código fuente generado por los programas Blokino**: como se mencionó anteriormente la validación del código fuente de los programas que se generan con Blokino se hizo de dos maneras: a) los desafíos se validan usando un módulo personalizado adaptado para cada desafío, dado que para cumplir los desafíos se deben realizar todas las pruebas. Estas pruebas deben cumplir con una

descripción específica y con el módulo personalizado de validación se valida esta descripción. Este módulo de validación se repite para todos los desafíos, con distintos criterios de validación. La validación varía por cada uno de los desafíos, debido que en cada uno se buscaba controlar distintos escenarios. Pero la validación del módulo personalizado quedó acotada a cada desafío, dejando pocas opciones de ser extensible. b) el modo “experto” de Blokino usa otro tipo de validación de los programas, el motor v8 de JavaScript. Este tipo de validación es más completa y precisa, dado que esta validación la hace el motor base de JavaScript. En síntesis estos dos tipos de validaciones de código fuente son usadas en distintas secciones de la plataforma, la validación personalizada para los desafíos y la validación del motor V8 de JS para el editor libre.

Dado que el objetivo de esta tesina es programar componentes electrónicos con la plataforma Blokino, durante el desarrollo de los desafíos fue necesario implementar módulos de validación del código fuente JS generado para luego poder ejecutarlo. Como primer acercamiento de solución se implementaron módulos de validación específicos de cada desafío, sin embargo se propone como trabajo futuro extender estos módulos de validación para que la plataforma Blokino cuente con su propio Motor de validación de código fuente genérico al estilo del motor V8 de JS, es decir que pueda adaptarse a cada desafío. Este aporte haría más versátil la manera en cómo se arman los programas de Blokino.

Capítulo 7 - Conclusiones y trabajos futuros

7.1 Conclusiones

Las plataformas educativas de robótica se utilizan para enseñar a niñas, niños y jóvenes programación en escuelas y en otros espacios educativos. Existe cierto consenso que es un material motivador y que facilita el aprendizaje de programación. Actualmente existen múltiples proyectos globales sobre robótica educativa que utilizan el paradigma de programación visual basada en bloques dada la naturalidad con la que es adoptada. En general estos proyectos usan hardware cerrado para ejecutar los programas y muchos de ellos son de código fuente privativo.

El objetivo del trabajo aquí presentado es desarrollar una plataforma de código fuente libre, llamada Blokino, que permite construir y programar objetos electrónicos, favoreciendo el aprendizaje de programación en las aulas de la escuela secundaria. La intención es que las y los estudiantes puedan llevar adelante proyectos sencillos de robótica, usando elementos de hardware libre, de bajo costo y frecuentemente disponibles en las escuelas.

La implementación de Blokino se encuentra alojada en el repositorio público github **https://github.com/georgefarfan/blokino**, junto con la documentación necesaria. Es posible hacer aportes para mejorar la plataforma, quedando a mi cargo la aprobación de los cambios propuestos. El proyecto es de acceso público alentando de esta manera la conformación de una comunidad de desarrolladores de Blokino que aporten mejoras.

Para el armado del kit de componentes electrónicas de hardware libre que conforman Blokino, durante el desarrollo de la plataforma, se probaron distintos componentes de Arduino y de esta manera se seleccionaron los compatibles con Blokino.

Para el desarrollo de la plataforma de software se evaluaron distintos frameworks dedicados a crear aplicaciones de escritorio usando JavaScript. Se hicieron pruebas con Meteor y Electron. El rendimiento, la extensibilidad y la creación de instaladores para Windows y Linux, fueron los criterios para elegir Electron. Además un factor importante es que funciona en conjunto con la librería de JavaScript Robotics llamada Johnny-Five. La selección de Johnny-Five requirió de la realización de múltiples pruebas con diferentes librerías de JavaScript Robotics. El criterio de elección de Johnny-Five fue el soporte para una amplia variedad de componentes electrónicos y que es posible personalizarlo para adaptarlo a Electron.

Blokino se publicó en una página web que contiene todo lo necesario para descargarlo y aprender a usarlo. Se encuentra disponible en: **www.blokino-platform.com**.

Dentro de la comunidad de JavaScript Robotics se crearon robots que en la comunidad los llaman NodeBots, debido a que son robots que funcionan con JavaScript y NodeJS. A medida que completaba la implementación de los desafíos de cada uno de los componentes electrónicos del kit de Blokino, se llegó a la conclusión que era necesario armar ejemplos que llamen la atención de las y los jóvenes. Para ello se armaron varios modelos de NodeBots con cartón y cinta adhesiva, pero debido a que no tenían consistencia se fueron descartando, sin embargo fueron útiles para ganar experiencia en el ensamblado de partes y en el uso de diferentes materiales. Finalmente se diseñaron dos NodeBots, uno de fibrofacil y otro de plástico impreso en una impresora 3D. Los

NodeBots despertaron la curiosidad e interés de las y los estudiantes por Blokino y esto pudo observarse en los encuentros de prueba.
La realización de las pruebas de campo con potenciales usuarios de Blokino permitió recolectar información útil sobre la experiencia de usar Blokino. Las y los estudiantes hicieron distintos aportes, desde críticas hasta recomendaciones. La plataforma funcionó correctamente en términos generales, las y los estudiantes pudieron completar todas las actividades propuestas y se pudo brindar soporte cuando se requirió.
Ante lo antes expuesto se puede afirmar que la plataforma Blokino cumple con los objetivos planteados en esta primera etapa de desarrollo, dado que este trabajo se focalizó en que las y los estudiantes puedan crear programas sencillos que se ejecuten sobre placas de hardware libre, como así también crear una plataforma que se pueda ejecutar en Windows y Linux usando código fuente libre y hardware libre.

7.2 Trabajos futuros

A continuación de describen los posibles trabajos futuros que surgen de esta tesina de grado a partir de las evaluaciones realizadas, la socialización en los *meetups* y mi propia experiencia durante el desarrollo de Blokino.
Entre las mejoras se incluyen la implementación de módulos personalizados, mejoras de infraestructura para conseguir mejor rendimiento de la plataforma y pruebas de concepto para buscar alternativas de funcionalidades:

- Migrar la plataforma desarrollada con VanillaJS a algún framework escalable como Angular, React o VueJS. Esta migración busca lograr una aplicación escalable, mantenible y poder aplicar tecnologías de JavaScript que garantizan mejor rendimiento.
  Durante el desarrollo de la plataforma se eligió usar VanillaJS debido a los conflictos que se presentaron entre los módulos de NPM de Robotics y el generador de instaladores multiplataforma de Electron.
  Migrar a frameworks como Angular, React o VueJS permitirá alcanzar un mejor rendimiento y poder usar tecnologías como TypeScript o Redux, que aportan un orden de trabajo que se basa en estándares establecidos que aseguran un entorno de trabajo escalable.
- Implementar un módulo personalizado para establecer una conexión bluetooth entre Blokino y las placas Arduino, usando el módulo Bluetooth Hc-05.
  Para poder realizar esta POC, se usó la guía de configuración de módulos de Bluetooth publicada por Rick Waldron (Rick Waldron. J5 & Hc-05, 2014). Se logró conectar tanto en Windows como en Linux, pero a medida que se usaba se encontraron inconsistencias con la conexión, como por ejemplo una pérdida de conexión o retraso en el envío de los programas para que se ejecuten sobre la placa Arduino. El módulo personalizado debe contener estas funcionalidades:
  - Detectar los dispositivos bluetooth: actualmente los módulos de Serialport y Gort que se usan en Blokino funcionan bien pero falta conseguir más información sobre los dispositivos detectados.
  - Configuración del módulo de Bluetooth Hc-05: está configuración se realizó usando Arduino IDE, mediante una secuencia de pasos difíciles de seguir debido al uso de sketch de Arduino y configuración del entorno Arduino IDE. Para solucionar esto y que la configuración del módulo Hc-05 sea automática mediante pasos sencillos,

se debería implementar un módulo de JavaScript que se encargue de hacer este proceso, o generar otra aplicación complementaria de Blokino que haga esta configuración. Debido a que la configuración del módulo Hc-05 debe ser para Windows y Linux, se debe tener en cuenta que el módulo personalizado debe funcionar en ambas plataformas. Con este módulo de JavaScript personalizado, se podría reemplazar a Gort. De esta manera se lograría una aplicación con mejor rendimiento, como así también un control más detallado de los dispositivos que se conectan, ya sea mediante USB o vía un módulo de Bluetooth.

- Implementar una GUI de escritorio liviana: la GUI de Blokino está desarrollado con Electron, este framework fue elegido por el soporte que brinda y los resultados que se tienen con distintas aplicaciones conocidas, como Slack, Skype, VS Code, Visual Code y Whatsapp. Sin embargo una de las desventajas de Electron es el peso de las aplicaciones, sobre todo en Windows. Para reducirlo se analizaron dos posibles POCs:
  - Establecer una mejora de empaquetamiento de la aplicación usando npm, acotando todos los recursos y módulos JavaScript que no son usados dentro de la plataforma.
  - Realizar una PWA Desktop y hacer una comparación de consumo de recursos tanto en Windows como en Linux.
- Implementar un motor de validación para que las pruebas de los desafíos puedan validarse de una manera genérica. En la sección sobre dificultades encontradas se describe detalladamente la problemática y las propuestas de mejora.

Referencias

Aho, A. V. (2012). Computation and computational thinking. The Computer Journal, 55(7), 832-835.

Arduino - entorno de desarrollo integrado multiplataforma, para programar placas de Arduino. Obtenido de https://www.arduino.cc/en/main/software. Fecha de último acceso: 27/2/2020

Backstop and Rick Waldron (2015). JavaScript Robotics: Building NodeBots with Johnny-Five, Raspberry Pi, Arduino, and BeagleBone (Make). Make Community, LLC; 1 edition.

Banchoff Tzancoff C., Martin S., Gómez S. y López F (2019).. Experiencias en robótica educativa. Diez años trabajando con escuelas argentinas. Aceptado para su publicación en el Congreso TEYET 2019. San Luis, Argentina.

Bee-Bot: robot educativo. Obtenido de https://www.bee-bot.us. Fecha de último acceso: 27/2/2020

Benitti, F. B. V. (2012). Exploring the educational potential of robotics in schools: A systematic review. Computers & Education, 58(3), 978-988.

Blockly - A JavaScript library for building visual programming editors. (2012). Obtenido de https://developers.google.com/blockly. Fecha de último acceso: 27/2/2020

Bocoup - Web Platform Consulting. (2012). Obtenido de: https://bocoup.com/about. Fecha de último acceso: 27/2/2020

Code Bug: dispositivo programable y portátil. (2015). Obtenido de: http://www.codebug.org.uk/. Fecha de último acceso: 27/2/2020

Consejo Federal de Educación (2015). Importancia estratégica a la enseñanza y el aprendizaje de la Programación en todas las escuelas, durante la escolaridad obligatoria. Resolución Nº 263/15. Obtenido de: https://cfe.educacion.gob.ar/resoluciones/res15/263-15_01.pdf. Fecha de último acceso: 27/2/2020

Consejo Federal de Educación (2018). Núcleos de Aprendizaje Prioritarios para Educación Digital, Programación y Robótica. Resolución 343/18. https://www.argentina.gob.ar/sites/default/files/res_cfe_343_18_0.pdf. Fecha de último acceso: 27/2/2020

Denning, P. J. (2017). Remaining trouble spots with computational thinking. Communications of the ACM, 60(6), 33-39.

Díaz J., Queiruga C., Banchoff Tzancoff C., Fava L. and Harari V. (2015). Educational robotics and videogames in the classroom. 10th Iberian Conference on Information Systems and Technologies (CISTI), Aveiro, 2015, pp. 1-6. doi: 10.1109/CISTI.2015.7170616.

Digital Ocean - proveedor de servidores virtuales privado. (2019). Obtenido de https://www.digitalocean.com/docs/. Fecha de último acceso: 27/2/2020.

Electron - Build cross platform desktop apps with JavaScript, HTML, and CSS. (2013). Obtenido de https://electronjs.org/docs. Fecha de último acceso: 27/2/2020

Johnny-Five: plataforma JavaScript para controlar robots y componentes eletronicos. (2012). Obtenido de http://johnny-five.io/. Fecha de último acceso: 27/2/2020

Homebrew Computer Club. (2018). Obtenido de: https://www.neoteo.com/homebrew-computer-club-1975/. Fecha de último acceso: 27/2/2020.

Introduction to DiY/counterculture. (2013, febrero). Obtenido de: https://www.permanentculturenow.com/introduction-to-diy-counterculture/. Fecha de último acceso: 27/2/2020

Lyza Danger Gardner. JavaScript on Things - Hardware for web developers - United States of América 2018.

Maker Movement, una nueva cultura de invención e innovación. (2017). Obtenido de: http://www.youngmarketing.co/la-cultura-del-maker-movement-y-como-esta-cambiando-el-mundo/. Fecha de último acceso: 27/2/2020

Martinez, C., Gomez, M. J. & Benotti, L. (2015). A comparison of preschool and elementary school children learning computer science concepts through a multilanguage robot programming platform. Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, 159-164.

Mblock: entorno gráfico de programación basado en el editor Scratch 2.0. (2011). Obtenido de https://www.mblock.cc/en-us/. Fecha de último acceso: 27/2/2020

Micro Bit: microcomputadora de educación maker (2012). Obtenido de https://microbit.org. Fecha de último acceso: 27/2/2020

Ministerio de Educación de la Nación (2017). Competencias de Educación Digital. - 1a ed. Ciudad Autónoma de Buenos Aires. Archivo Digital: descarga y online. ISBN 978-950-00-1198-3

Perch, K. (2015). Learning JavaScript Robotics. Editorial: Packt Publishing, Reino Unido.

Prototyping a Smart Device w/ Arduino & Node.js using Johnny-Five. (2016). Obtenido de: https://www.pubnub.com/blog/howcreate-a-smart-device-with-arduino-and-node-js-using-johnny-five/. Fecha de último acceso: 27/2/2020

Queiruga C., Banchoff Tzancoff C., Venosa P., Martin S., Aybar Rosales V., Gómez S. & Kimura I. EscuelasTIC. El pensamiento computacional en la escuela. Rodríguez N., Murazzo M., Ortega M., Lund M. (eds) XXI Workshop de Investigadores en Ciencias de la Computación 2019 (XXI WICC). ISBN 978-987-3984-85-3. pp 590-595.

Rick Waldron. (2012). Obtenido de: https://bocoup.com/about/bocouper/rick-waldron. Fecha de último acceso: 27/2/2020

Rick Waldron. (2014). Johnny Five and HC 05 Bluetooth Serial Port Module. Obtenido de: https://github.com/rwaldron/johnny-five/wiki/Getting-Started-with-Johnny-Five-and-HC-05-Bluetooth-Serial-Port-Module. Fecha de último acceso: 27/2/2020

Scratch: lenguaje de programación visual. (2003). Obtenido de: https://scratch.mit.edu/

Vue.js: The Progressive JavaScript Framework. (2014). Obtenido de: https://vuejs.org/. Fecha de último acceso: 27/2/2020

Wing J. (2006). Computational Thinking. In Communications of the ACM, vol. 49, 33-35.