AelE: una herramienta para la enseñanza de programación basada en Arduino

Gonzalo Pablo Fernández
Departamento de Ciencia y Tecnología
Universidad Nacional de Quilmes
Bernal, Buenos Aires, Argentina
gpfernandez@dc.uba.ar

Christian Cossio-Mercado
Departamento de Computación
Universidad de Buenos Aires
Buenos Aires, Argentina
ccossio@dc.uba.ar

***Resumen*—La enseñanza de programación aporta al pensamiento abstracto y a la resolución de problemas, entre otros aspectos. Adicionalmente, la utilización de programación de placas *Arduino*, con su diseño de hardware abierto y multiplicidad de recursos disponibles, permite trabajar con programación con salidas físicas, además facilitar el trabajo con proyectos. Existen muchas herramientas para el trabajo con programación en el contexto escolar, como *Scratch* y *PilasBloques*. En particular, para *Arduino* se suelen utilizar *mBlock* y *Scratch*, además de otras herramientas específicas de kits de robótica *Arduino*-compatibles. Las herramientas disponibles para el trabajo de programación con *Arduino* tienen limitaciones para su uso en la enseñanza, en tanto se requiere programar en texto, con la dificultad que implica para una persona que recién se inicia en la programación. Adicionalmente, las herramientas disponibles por bloques, o bien son específicas de ciertos kits, y no son compatibles con sensores y actuadores genéricos, o bien tienen pocos bloques disponibles, generan código poco legible y no son intuitivas en su uso, además de estar en su mayoría sólo en idioma inglés, entre otras cosas. En este trabajo presentamos la herramienta *Arduino en la Escuela*, que fue diseñada para la enseñanza de programación utilizando placas *Arduino* por medio de un lenguaje por bloques. Se destaca su versatilidad en tanto fue utilizada en contextos variados y con públicos de características muy diferentes, no sólo en edad, sino también en contexto sociocultural y formación previa. Las experiencias con la herramienta fueron mayormente con estudiantes de nivel secundario, pero también se realizaron talleres de formación para docentes e, inclusive, cursos para personas en contexto de encierro. En todos los casos se tuvo una buena aceptación de la herramienta, además de que los participantes pudieron completar adecuadamente las tareas propuestas y trabajar exitosamente con proyectos a elección.**

***Index Terms*—Entorno de programación, Enseñanza de Programación, Arduino, Programación por bloques**

I. Introducción

La enseñanza de programación es relevante para la educación obligatoria, lo que se vio reflejado en Argentina en su incorporación en todos los niveles obligatorios en 2018 [1], en tanto aporta al pensamiento abstracto y a la resolución de problemas, entre otros aspectos. Adicionalmente, la utilización de programación de placas *Arduino*, con su diseño de hardware abierto y multiplicidad de recursos disponibles, permite trabajar con programación tangible, además facilitar el trabajo con proyectos. Existen muchas herramientas que se usan para programación en el contexto escolar, como *Scratch* [2] y *PilasBloques* [3]. En particular, para *Arduino* se suelen utilizar herramientas como *mBlock* y el mismo *Scratch*, además de otras herramientas específicas de kits de robótica *Arduino*-compatibles u otras versiones propietarias, como *LEGO Mindstorms*. A nivel nacional, se cuenta con *ArguiBot* [4] que, a pesar de ofrecer varias ventajas, tanto de usabilidad como del punto de vista didáctico, sólo está diseñado para el uso dentro de un curso, y no presenta funcionalidades para uso en forma general. De acuerdo a lo relevado [5], todas las herramientas disponibles para el trabajo de programación con *Arduino* tienen limitaciones para su uso en la enseñanza, en tanto se requiere programar en texto, con la dificultad que implica para una persona que recién se inicia en la programación. Adicionalmente, las herramientas disponibles por bloques, o bien son específicas de ciertos kits, y no son compatibles con sensores y actuadores genéricos, o bien tienen pocos bloques disponibles, generan código poco legible y no son intuitivas en su uso. Además, la mayoría de ellas están sólo en idioma inglés. Así, en este trabajo presentamos la herramienta *Arduino en la Escuela*[1] (*AelE*), que fue diseñada para la enseñanza y el aprendizaje de programación utilizando placas *Arduino* por medio de un lenguaje por bloques.

El desarrollo de *AelE* fue motivado por la necesidad de una herramienta que permita programar placas *Arduino*, pero que a su vez esté especialmente diseñada para el ámbito educativo. Ninguno de los entornos de programación por bloques para Arduino cumplía con todos los requerimientos, o eran muy poco robustas en cuanto al código generado a partir de los bloques. Se comenzó haciendo una traducción al español de *Ardublockly* y ampliando su catálogo de bloques para permitir la interacción con más módulos de *Arduino*, aunque, posteriormente, el proyecto tomó entidad propia, diferenciándose lo suficiente como para poder ser una herramienta independiente.

II. Características de la herramienta

Se definieron las siguientes dimensiones para describir las principales características de *Arduino en la Escuela*:

II-A. Preparado para la enseñanza y el aprendizaje

En esta sección se describen las características que hacen de *AelE* una herramienta ideal para ser utilizada en el aula

[1] *Arduino en la Escuela*- Demo: http://aele.dc.uba.ar

ya que facilitan el ingreso al mundo de la programación. Son características que ayudan a generar un ambiente propicio para el aprendizaje asistiendo tanto a estudiantes a ganar autonomía como a docentes para tener más recursos para la enseñanza.

1. **Programación por bloques**: La forma de construir programas es mediante programación por bloques. En contraposición a la construcción de programas mediante la escritura de texto, como en el caso del *IDE de Arduino* , la programación por bloques es mucho más intuitiva para principiantes ya que expone visualmente las reglas de construcción sintáctica, las que suelen dificultar el aprendizaje cuando se empieza en texto.
2. **Bloque principal**: Muchos entornos de programación por bloques generan el código a partir de todos los bloques disponibles en el área de trabajo. Esto puede resultar muy confuso cuando hay una gran cantidad de bloques en diferentes posiciones, ya que no queda claro en qué orden deben ejecutarse o qué es lo que dispara su ejecución. *AelE* incorpora un bloque que define el ciclo principal de ejecución. Al iniciar el programa, sólo lo que esté contenido dentro de este bloque será ejecutado. Los bloques sueltos se ignoran al no haber una clara definición de cómo deberían ejecutarse.
3. **Mensajes de advertencias y errores descriptivos**: En general, los mensajes de error en los entornos de programación son muy poco descriptivos. Las personas con experiencia se van acostumbrando a entender estos mensajes, pero son muy difíciles de interpretar para quienes recién están aprendiendo a programar. Muchas herramientas orientadas a la educación resuelven este problema poniendo esfuerzo en generar mensajes descriptivos, aunque son pocas las que lo hacen para *Arduino*. En muchos casos, los mensajes de error generados por estas herramientas son los mismos que se muestran en el editor de *Arduino*.
4. **Asistencia en corrección de errores**: Además de contar con mensajes descriptivos, *AelE* provee algunas sugerencias para solucionar los errores detectados. Así, se busca que el usuario entienda por qué no funciona lo que está intentando hacer y ver cómo podría solucionarlo.
5. **Evita correcciones automáticas**: Muchas herramientas, orientadas tanto a profesionales como a principiantes, hacen uso de correcciones automáticas. Aunque pareciera una característica deseable, atenta contra que los estudiantes aprendan a solucionar los problemas por sí mismos. En la mayoría de las herramientas basadas en *Blockly* no es posible conectar dos bloques si sus tipos no coinciden (e.g., ver qué sucede si se intenta conectar un bloque numérico como condición de un bloque de alternativa). El problema de utilizar este comportamiento es que no permite que los estudiantes comprendan por qué esa composición es inválida. En *AelE*, en lugar de impedir la conexión, se muestra un mensaje de error explicando claramente por qué no se puede hacer.
6. **Composición de subtareas descendente**: Otra característica que comparten casi todas las herramientas basadas en *Blockly* es el mecanismo mediante el cual se definen subtareas. Este proceso consiste en crear primero un bloque de definición de subtarea, lo cual luego habilita la creación del bloque de invocación de dicha subtarea. En el esquema de trabajo descendente (en inglés, *top-down*) se recomienda escribir primero la estrategia de solución del problema principal, invocando a las subtareas correspondientes sin necesidad de haberlas definido. Siguiendo esta lógica, *AelE* permite crear bloques de invocación a subtareas sin definirlas antes.
7. **Comunicación serial**: En las etapas iniciales de aprendizaje de programación es de suma importancia poder depurar el código. Dado que el código se ejecuta en un hardware distinto al utilizado para escribir el código, sería demasiado complejo implementar un depurador completo. Aún así, es extremadamente útil poder imprimir valores en determinados puntos de la ejecución del programa. Para ello, *AelE* incluye el manejo de la conexión serial entre la placa y la computadora. Esto no sólo permite depurar el código sino que también es útil para explorar nuevos sensores para los cuales se desconoce el rango de valores que pueden describir.
8. **Ejemplos y Tutoriales**: *AelE* cuenta con varios ejemplos, básicos e intermedios, que pueden servir como base para implementar otros proyectos o como guía para entender cómo interactuar con un módulo en particular. También cuenta con algunos tutoriales interactivos para empezar a utilizar la herramienta.

II-B. Interfaz de usuario cómoda e intuitiva

Una herramienta usable debe contar con una interfaz de usuario (IU) fácil de navegar e intuitiva. Las IU muy sobrecargadas abruman al usuario con demasiada información, aunque, en el extremo opuesto, una interfaz demasiado básica dificulta encontrar las características requeridas en cada momento. *Arduino en la Escuela* utiliza un enfoque equilibrado, con una interfaz con poco contenido, pero que permite incorporar nuevos elementos/funcionalidades en la medida que sea necesario.

1. **Máximo aprovechamiento del área de trabajo**: La mayor parte de la interfaz está ocupada por el área de trabajo de bloques. En la barra de título hay un botón para abrir el menú lateral en el extremo izquierdo y algunos botones de acceso rápido en el extremo derecho, como el de abrir el menú de configuración o la ayuda. En el centro de la barra de título se encuentra el nombre del proyecto junto a un botón para modificarlo.
2. **Vistas y modo de visualización de vistas**: Además de la vista de bloques hay una segunda vista que muestra el código *Arduino* generado, y se puede alternar entre ambas vistas con el selector de pestañas. También hay un botón que permite ver ambas vistas a la vez, siendo posible redimensionar el ancho de cada una para darle prioridad a una, pero sin dejar de ver la otra.
3. **Monitor serial incorporado**: En la barra inferior se encuentra la ventana del monitor serial que aparece

minimizada por defecto, pero se puede abrir haciendo clic sobre ella. Nuevamente, es posible redimensionar su altura con respecto al resto de la interfaz.

4. **Íconos**: La gran mayoría de los botones incluyen íconos para describir su comportamiento, como los de acceso rápido y los distintos menúes, así como las opciones en los menúes contextuales. También se encuentran íconos en las categorías del toolbox y en muchos de los bloques.

II-C. Robustez

Una de las características más importantes de *Arduino en la Escuela*, que pocas herramientas comparten, es que el código generado es siempre válido para el *IDE de Arduino* . Así, si *AelE* no muestra errores en los bloques, entonces el *IDE de Arduino* tampoco generará errores al pasar el código a la placa. Esto es algo que se espera en general de las herramientas de programación por bloques. Sin embargo, es difícil manejar errores de tipos o referencias sin definir en el contexto de los bloques en el que se omiten todas esas declaraciones.

1. **Manejo de tipos**: *Blockly* fue pensado originalmente para lenguajes no tipados o débilmente tipados. Al usarlo para generar el código de un lenguaje tipado surgen dificultades para declarar el tipo de las variables, las funciones y los parámetros. *AelE* cuenta con un sistema de tipado propio que permite inferir los tipos de todas las variables, funciones y parámetros y generar errores cuando estos tipos no coinciden en distintas apariciones.
2. **Parámetros, variables locales y variables globales**: Una de las principales falencias de los entornos de programación por bloques para *Arduino* es su incapacidad para manejar parámetros, así como para distinguir el variables locales y globales. En *Ardublockly*, por ejemplo, se optó por anular el uso de parámetros y forzar el alcance global a cualquier variable que se declare. Gracias a su sistema de tipado, *AelE* puede permitir el uso tanto de parámetros como de variables locales.
3. **Errores y advertencias estáticas**: Como la mayoría de los entornos de desarrollo modernos, *AelE* analiza el código mientras este está siendo construido y genera notificaciones tanto de errores como de advertencias en el mismo. Estas notificaciones se muestran a través de la característica de *Blockly* que muestra burbujas sobre los bloques. *Blockly* tiende cada vez más a dejar de usar estos mensajes flotantes y restringir las construcciones de código erróneas de otras formas y así también lo están haciendo los entornos basados en *Blockly*. Como se mencionó en la sección II-A, preferimos que se puedan generar construcciones erróneas, pero que se muestre un mensaje que ayude a entender por qué esa construcción es inválida, en lugar de simplemente impedirla. Además, de los errores de tipado ya mencionados, el análisis realizado permite detectar e informar una gran cantidad de errores, como usos inválidos de los pines de la placa o la falta de bloques de configuración para módulos que lo requieren, así como generar advertencias, como elementos que están definidos más de una vez o la asignación de un valor a un parámetro. Las advertencias pueden ser ignoradas (lo que significa que el código puede generarse aunque tenga advertencias, pero el resultado podría no ser el esperado) mientras que los errores impiden la generación del código *Arduino*.
4. **Errores en tiempo de ejecución**: Además de los errores que se pueden detectar haciendo análisis estático de código, *AelE* también contempla errores en tiempo de ejecución, como un acceso fuera de rango a una lista o intentar acceder a un archivo de la tarjeta SD que no existe. En el funcionamiento normal de *Arduino*, la mayoría de estos errores detienen la ejecución sin notificar de ninguna forma que ocurrió un error. El código generado por *AelE* captura estos errores y puede mostrarlos a través del monitor serial.
5. **Control de acceso a placas**: Más allá de los errores en la ejecución del código, el *IDE de Arduino* también genera errores cuando hay problemas de comunicación con la placa. Estos también se evitan con *AelE*, ya que se asegura que haya una placa conectada y que esta sea aquella para la cual el proyecto está configurado.

II-D. Herramientas de productividad

Aquí se describen características de los entornos de desarrollo para facilitar y acelerar la programación. Tienen el doble objetivo de, por un lado, que sea más fácil aprender a programar con la herramienta y, por el otro, que usuarios expertos puedan crear proyectos más complejos. Algunas de estas funcionalidades están pensadas para quienes ya dominan entornos de desarrollo textuales y encuentran dificultades con la forma de trabajar que proponen los entornos visuales basados en bloques, como la interacción a través del mouse.

1. **Comunicación con el IDE de Arduino incorporada**: La mayoría de los entornos de desarrollo modernos permiten ejecutar y depurar el código realizado desde la misma interfaz del editor. De la misma forma, *AelE* permite ejecutar el código a la placa, utilizando por detrás el *IDE de Arduino* , sin abandonar la aplicación.
2. **Autocompletado**: Por medio del sistema de tipado mencionado en la sección II-C, *AelE* provee la funcionalidad de completar automáticamente los bloques que tienen huecos, los cuales se rellenan con bloques por defecto, en función del tipo correspondiente.
3. **Buscador de bloques**: Es muy común que en los entornos de programación por bloques sea difícil encontrar un bloque particular para su uso, sobre todo cuando el entorno incluye una gran variedad de bloques distintos, y los entornos para *Arduino* suelen caracterizarse por ello). Para aliviar este problema, *AelE* cuenta con un buscador por texto de bloques.
4. **Multiselección de bloques**: Otro inconveniente que suelen tener los entornos de programación por bloques es que sólo permiten seleccionar un bloque a la vez. En cualquier entorno de desarrollo es posible seleccionar varios elementos a la vez y por eso a la interfaz de *Blockly* se le agregó la posibilidad de seleccionar varios

bloques en simultáneo, y poder realizar acciones sobre ellos, como arrastrar y soltar.

5. **Pegar a continuación**: Se busca que *AelE* se asimile a las herramientas de programación textual al momento de editar código. Así como en texto se cuenta con un cursor, y al realizar alguna operación (e.g., copiar y pegar) el resultado se realiza a continuación de él, en la mayoría de las herramientas basadas en *Blockly* el comportamiento de pegar un bloque crea uno nuevo, suelto en una posición cercana al bloque copiado. En *AelE* la posición del nuevo bloque depende de lo que esté seleccionado en ese momento. Si hay un bloque seleccionado, primero se intenta conectar el nuevo bloque al bloque seleccionado. Si este tiene huecos y el bloque pegado es una expresión, entonces se usa para rellenar el hueco. Si ambos son comandos, el pegado se conecta como bloque siguiente de la secuencia. Si no es posible conectarlos (e.g., si el seleccionado es una expresión y el pegado un comando) el comportamiento es el de *Blockly*, pero no respecto al bloque copiado sino al seleccionado. Si no hay ningún bloque seleccionado el comportamiento es el mismo, pero respecto al bloque copiado, si es que este sigue existiendo.
6. **Edición rápida en los menúes contextuales**: Además de la opción para rellenar huecos automáticamente, se permite la edición rápida desde los menúes contextuales de los bloques, siendo una de las más relevantes el convertir a un bloque de comando en un procedimiento y convertir a un bloque de expresión en una función, una variable, un parámetro o un valor constante. También se incluyen opciones adicionales en el menú contextual del área de trabajo, como abrir o cerrar todas las burbujas, los comentarios, los errores o las advertencias.

II-E. Diseño del lenguaje

En esta sección se encuentran las características que hacen que el lenguaje de programación por bloques de *Arduino en la Escuela* sea completo y versátil.

1. **Gran variedad de sensores y actuadores**: Encontrar un equilibrio en la cantidad de bloques disponibles es una tarea muy compleja para cualquier entorno de programación por bloques, pero más aún para uno hecho para programar placas *Arduino*, debido a la enorme cantidad de sensores y actuadores disponibles. Muchos entornos optan por los extremos, ya sea incluyendo un conjunto de bloques mínimo, lo que hace que construir programas complejos sea más dificultoso, o incluyen una gran variedad de bloques, lo que hace que sea difícil encontrar un bloque particular cuando se lo necesita. *AelE* tiene bloques para los sensores y actuadores más comunes, pero asegurándose de no abrumar al usuario con demasiada información a la vez. Además de contar con el buscador de bloques mencionado en la sección II-D, los bloques se organizan en categorías y subcategorías que facilitan la navegación.
2. **Gran variedad de elementos de programación**: Al crear una herramienta por bloques se define un nuevo lenguaje, el cual traducirá al lenguaje de base, que generalmente es textual y de más bajo nivel. Muchos entornos utilizan el mismo lenguaje que aquel al que se traducen, pero reemplazando los elementos textuales por bloques. Esto tiene la ventaja de que la conversión de bloques a texto es sencilla y muchas veces hasta puede ser bidireccional, pero, a su vez, tiene la desventaja de que el lenguaje de bloques queda restringido por el lenguaje al que se traduce. En el caso de *Arduino*, al ser un lenguaje de bajo nivel, esto significa un problema para el ámbito educativo. Es por ello que el lenguaje de *AelE* es de más alto nivel que el de la mayoría de los demás entornos visuales para *Arduino* y por eso permite trabajar con elementos del lenguaje que no están presentes en *Arduino* como listas y eventos.
3. **Variantes y extensiones**: Otra característica que ayuda a evitar el exceso de bloques es que algunos bloques tiene variantes. Es frecuente que los entornos de programación por bloques para *Arduino* cuenten con un bloque que active un zumbador y otro que lo active durante una cantidad de tiempo, mientras que en *AelE*, el segundo bloque es una variante del primero, por lo que ocupan un único espacio en el menú. Además, el bloque principal permite configurarse para que ejecute una única vez, que es el comportamiento habitual en la mayoría de los demás lenguajes de programación, además de hacerlo indefinidamente, que es el comportamiento por defecto en *Arduino*. Así, se accede a estas variantes desde cada bloque, donde se muestran las configuraciones adicionales que tiene disponibles.
4. **Bloques para describir pines**: Cuando se configura un sensor o un actuador, en *Arduino* se debe especificar en qué pines de la placa está conectado. Esta información puede ser suministrada en un bloque como un argumento o a través de un campo selector. Mientras que la mayoría de los entornos optan por la alternativa del argumento, tienen el problema de que no pueden validar que lo ingresado sea un pin válido, ya que se debe verificar que sea un número entero y que esté en el rango de los pines que la placa posee. Además, esta decisión de diseño permite que el argumento sea una variable o, incluso, el resultado de una función, lo que hace que no se pueda resolver antes de comenzar la ejecución (o, peor aún, que vaya cambiando a lo largo del programa). La alternativa del campo selector es más segura, ya que permite definir qué pines serán utilizados para sensores y actuadores antes de comenzar el programa, pero restringe la libertad del programador impidiendo, por ejemplo, tener definido un pin a través de un nombre y poder cambiarlo en un único lugar para todo el programa. *AelE* combina varias características para lograr una correcta implementación de la alternativa por argumento, pero sin riesgo de generar programas inválidos: por un lado incorpora bloques específicos para describir a los pines, y usando el sistema

de tipado impide que pasen argumentos que no son pines cuando se espera algo de este tipo, además de que se verifica que los bloques no sean variables ni funciones (además de que las variables no pueden recordar pines ni las funciones pueden devolverlos), y, finalmente, se incorporan bloques de definición de valores constantes para poder asignarle nombre a los pines.

II-F. Personalización

Se espera que *AelE* pueda adaptarse a distintos contextos de uso, por lo que puede personalizarse a través del menú de configuración, más allá de las configuraciones iniciales.

1. **Editor de paletas**: Como ya se mencionó en la sección II-E, es difícil diseñar un menú de bloques (paleta) sin llegar a los extremos en los que hay tantos que es difícil encontrar lo buscado o que hay tan pocos que cualquier programa, por más simple que parezca, requiere una gran cantidad de bloques para ser construido. Más allá de que la paleta por defecto está balanceada en ese sentido, también podrían darse situaciones particulares en las que se requiera tener otros sets de bloques. Así, se implementó un editor avanzado de paletas que habilita a que docentes puedan diseñar sus propios set de bloques en función de las necesidades de sus clases particulares. El editor no sólo permite seleccionar qué bloques aparecen en la paleta, sino que también permite agruparlos en categorías personalizables.
2. **Herramientas habilitadas**: Además de personalizar los bloques disponibles en el menú, *AelE* permite controlar qué herramientas del lenguaje se pueden usar. Así, es posible planificar distintas actividades restringiendo el acceso a determinadas herramientas que no se espera que sean utilizadas para la resolución. Adicionalmente, se pueden configurar algunos bloques para que sólo aparezcan en la paleta en ciertas ocasiones (e.g., que los bloques de configuración de módulos sólo aparezcan si en el área de trabajo hay un bloque para tal módulo).
3. **Manejo de errores y advertencias**: Es posible configurar la forma en que *AelE* notifica los errores y las advertencias en el código. Las advertencias pueden ser tratadas como errores (i.e., impiden generar el código), ser tratadas sólo como advertencias (lo cual es comportamiento por defecto) o directamente ser ignoradas. En lo respectivo a los errores, estos no se pueden ignorar, pero se puede decidir si se los muestra apenas son detectados o recién cuando se intenta ejecutar el código. Por defecto, los errores sobre procedimientos o funciones sin definir se muestran al intentar ejecutar el código, mientras que el resto se muestra inmediatamente.

III. Resultados

AelE se pudo implementar en algunos cursos y talleres de programación, con públicos de diferentes edades y características. Se destacó por su versatilidad, ya que fue utilizada en contextos variados y con públicos de características muy diferentes, no sólo en edad, sino también en contexto sociocultural y formación previa. Aunque la mayoría fueron estudiantes de nivel secundario en contexto extracurricular, también se realizaron talleres de formación docente y cursos de educación en contexto de encierro. En todos los casos la herramienta tuvo buena aceptación de los participantes, quienes pudieron realizar todas las tareas propuestas. En particular se destaca el éxito logrado en el trabajo con proyectos, instancia de evaluación final en la mayoría de los cursos en los que se utilizó la herramienta durante varios encuentros.

Cuadro I
Preguntas y puntaje promedio de las respuestas a cada una.

Pregunta	**Puntaje**
¿Qué tan difícil creías que era programar Arduino ANTES?	3.68
¿Qué tan difícil creés que AHORA?	2.00
¿Cómo se ve la herramienta?	4.74
¿Cuánto cuesta usarla?	2.27
Cuántas cosas se pueden hacer con ella	4.61
¿Te gustaría seguir usándola?	4.70
¿Cómo fue tu experiencia general usando Arduino?	4.41

En 2019 se realizó un taller sobre robótica y electrónica de cinco encuentros para 60 estudiantes de entre 11 y 17 años. Desde el segundo encuentro se trabajó exclusivamente con *AelE*, y como cierre del taller los participantes tuvieron que presentar un proyecto grupal. Al finalizar el taller 23 participantes completaron una encuesta de satisfacción acerca del taller y de la herramienta usada. Cada pregunta se respondía con un número del 1 al 5. En el cuadro I se muestran las preguntas y el puntaje promedio para cada una. Como puede verse, tras finalizar la cursada la percepción de la dificultad de programar con *Arduino* se redujo a casi la mitad.

En 2023 se realizó un taller de programación de seis encuentros para 44 estudiantes de entre 16 y 18 años utilizando únicamente *AelE*, cuya planificación y temario se pueden ver en [6], en el cual los participantes tuvieron que presentar un proyecto final grupal. Luego se realizó una encuesta de satisfacción acerca del taller y de la herramienta usada, que fue respondida por 32 participantes. En el cuadro II se muestran las preguntas y el puntaje promedio para cada una (cada pregunta se respondía con un número del 1 al 5).

Durante 2023 se desarrollaron 2 cursos cuatrimestrales de robótica y pensamiento computacional de 2 horas semanales en los penales de Devoto y Ezeiza. Participaron 92 personas el primer cuatrimestre y 47 en el segundo (muchas de las personas que se inscribieron en el segundo habían cursado el primero), y como cierre de cada curso los participantes tuvieron que presentar un proyecto grupal. En ambos cursos hubo una amplia variedad de edades, con una importante proporción de participantes mayores de 60 años, y al nivel educativo alcanzado, ya que se tuvo participantes con educación primaria hasta profesionales universitarios. El primer curso fue aprobado por 50 personas y el segundo por 29. Al finalizar cada curso se realizó una encuesta anónima en la que se pedía calificar la herramienta *AelE*. En el primero, 27 personas la dieron como “Excelente” y 22 como “Buena”. En el segundo, 19 la dieron como “Excelente” y 3 como “Buena”.

Cuadro II
PREGUNTAS Y PUNTAJE PROMEDIO DE LAS RESPUESTAS A CADA UNA.

Pregunta	Puntaje
¿Qué te pareció en general la herramienta AelE?	4.28
Fácil de usar (no presenta complejidades...)	3.59
Intuitiva (es fácil encontrar las cosas que necesito...)	3.69
Entretenida (me divierte hacer actividades y me dan ganas de hacer más actividades similares)	4.13
Potente (me permite realizar las cosas que quiero realizar)	3.69
Educativa (mientras lo uso voy entendiendo las cosas que hago y cuando algo falla me resulta fácil entender por qué)	3.88
Frustrante (me topo con problemas que no puedo resolver y la herramienta no me ayuda a entender cómo solucionarlos)	2.78
Compleja (me cuesta entender qué hace cada cosa y cómo hacer lo que quiero hacer)	2.59
Motivadora (me dan ganas de seguir usándola, de seguir explorando y de seguir pensando qué más podría hacer)	3.78
Visualmente atractiva (la interfaz es agradable a la vista)	3.94
Aburrida (resolver ejercicios es tedioso...)	2.09
En general, ¿cómo fue tu experiencia usando AelE?	4.25

En ningún caso fue evaluada como "Regular" o "Mala".

La docente a cargo de los cursos también evaluó el uso de *AelE* como herramienta. En el primer curso, introductorio, funcionó muy bien ya que no se realizaron proyectos complejos y sólo se requirieron los bloques más estándar. En el segundo, un poco más avanzado, surgieron algunas dificultades al utilizar bloques o estructuras de código más complejos. Por un lado, dejó de ser fácil e intuitivo ubicar los bloques necesarios en la paleta. Algunos bloques sólo aparecen en determinadas circunstancias (e.g., los bloques de configuración de algunos módulos sólo aparecen cuando hay en el área de trabajo un bloque que utiliza tal módulo) y, si bien se puede configurar la herramienta para cambiar ese comportamiento (como se mencionó en la sección II-F), la docente no tenía conocimiento de ello. Tampoco sabía la docente cómo habilitar las variables globales, que por defecto están deshabilitadas.

Por otro lado, los proyectos más complejos requieren muchos más bloques en el área de trabajo a la vez, lo que lleva a que el código sea mucho menos legible (aun haciendo división en subtareas) y difícil de navegar, sobre todo en monitores pequeños o de baja resolución. También se notó una gran falta de autonomía en la mayoría de los grupos en el segundo curso con respecto al primero. Adicionalmente, resultó poco intuitivo el hecho de que los bloques de configuración aparezcan "sueltos" en el área de trabajo y no conectados a otro bloque.

IV. DISCUSIÓN

La amplia experiencia con talleres introductorios de programación nos permite afirmar que *AelE* es una excelente herramienta para iniciarse en el mundo de la programación con *Arduino*. Nuestros objetivos iniciales eran que también sirviera para proyectos avanzados y para gente con más experiencia. Sin embargo, no tenemos suficientes experiencias con cursos ni con usuarios avanzados. También es importante mencionar que la mayoría de las experiencias utilizando *AelE* fueron supervisadas por integrantes del equipo de desarrollo, y que, en las pocas instancias en las que la herramienta fue utilizada sin su supervisión, surgieron complicaciones que los docentes a cargo no supieron cómo resolver.

V. CONCLUSIONES

Arduino en la Escuela fue diseñada y construida para facilitar la enseñanza de la programación de placas *Arduino*, superando las limitaciones de las herramientas existentes, con un enfoque didáctico-pedagógico. Adicionalmente, incluye muchas funcionalidades que no están disponibles en la mayoría de los demás entornos de programación por bloques para *Arduino*. En particular, permite trabajar con una gran variedad de elementos de programación que no son habitualmente soportados, como parámetros, listas y eventos, y cuenta con características que la hacen más cercana a los entornos de desarrollo textuales, como la selección múltiple, pegar a continuación, o el manejo de errores y advertencias. También admite un alto nivel de personalización, que se adecua a las necesidades de cada equipo docente. *AelE* se pudo usar con éxito en distintos contextos, tanto como parte de la educación obligatoria de todos los niveles, en cursos extracurriculares, como parte de formación docente y en otros contextos de enseñanza (e.g., en contextos de encierro).

VI. TRABAJO FUTURO

Se está trabajando en varias mejoras para la herramienta. La principal es la incorporación de una versión textual del lenguaje y la capacidad para hacer la conversión bidireccional entre ambas versiones. Para mejorar la usabilidad en proyectos grandes se está planificando una nueva interfaz que permite manejar varias pestañas a la vez, de forma de poder organizar mejor el código. Además, se planean mejoras de accesibilidad, entre las que están contar con una versión apta para educación inicial o estudiantes con dificultades para la lectoescritura, y se espera desarrollar adaptaciones para su uso por parte de estudiantes con problemas motrices o cognitivos.

REFERENCIAS

[1] C. F. de Educación, "Núcleos de aprendizajes prioritarios para educación digital, programación y robótica," *Anexo I de la Resolución CFE Nro. 343/18*, 2018. [Online]. Available: http://www.bnm.me.gov.ar/giga1/normas/RCFE_343-18.pdf

[2] M. Resnick, J. H. Maloney, A. Monroy Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[3] A. Sanzo, F. Schapachnik, P. Factorovich, and F. Sawady O'Connor, "Pilas bloques: A scenario-based children learning platform," *2017 Twelfth Latin American Conference on Learning Technologies*, pp. 1–6, 2017.

[4] N. Streri, M. Daniele, and M. Uva, "Arguibot ide: Un entorno de desarrollo basado en la programación en bloque para la enseñanza de la robótica usando la plataforma arduino," *Jornadas Argentinas de Didáctica de las Ciencias de la Computación 2023*, 2023.

[5] G. P. Fernández and C. Cossio-Mercado, "Assessment of arduino block-based programming environments for teaching and learning," *19th WiPSCE Conference on Primary and Secondary Computing Education Research*, 2024. [Online]. Available: https://www.researchgate.net/publication/380568592_Assessment_of_Arduino_block-based_programming_environments_for_teaching_and_learning

[6] ——, "Enseñanza de fundamentos conceptuales de programación usando arduino," *Jornadas Argentinas de Didáctica de las Ciencias de la Computación 2023*, 2023. [Online]. Available: https://www.researchgate.net/publication/380174232_Ensenanza_de_fundamentos_conceptuales_de_programacion_usando_Arduino