Contribuciones al proyecto Physical Bits

Victoria Agnelli
Universidad Abierta Interamericana, Centro de Altos Estudios en Tecnología Informática, CABA, Argentina

Resumen
En este artículo se detallan las contribuciones realizadas al proyecto Physical Bits. Las mismas se centraron principalmente en ampliar la funcionalidad del entorno de desarrollo, mejorar la interfaz gráfica para hacerla más accesible a usuarios sin experiencia, y adaptar el lenguaje de programación para su futura utilización en el proyecto Mendieta URPE. Todas las contribuciones fueron integradas satisfactoriamente al proyecto y se encuentran disponibles en el repositorio de Github.

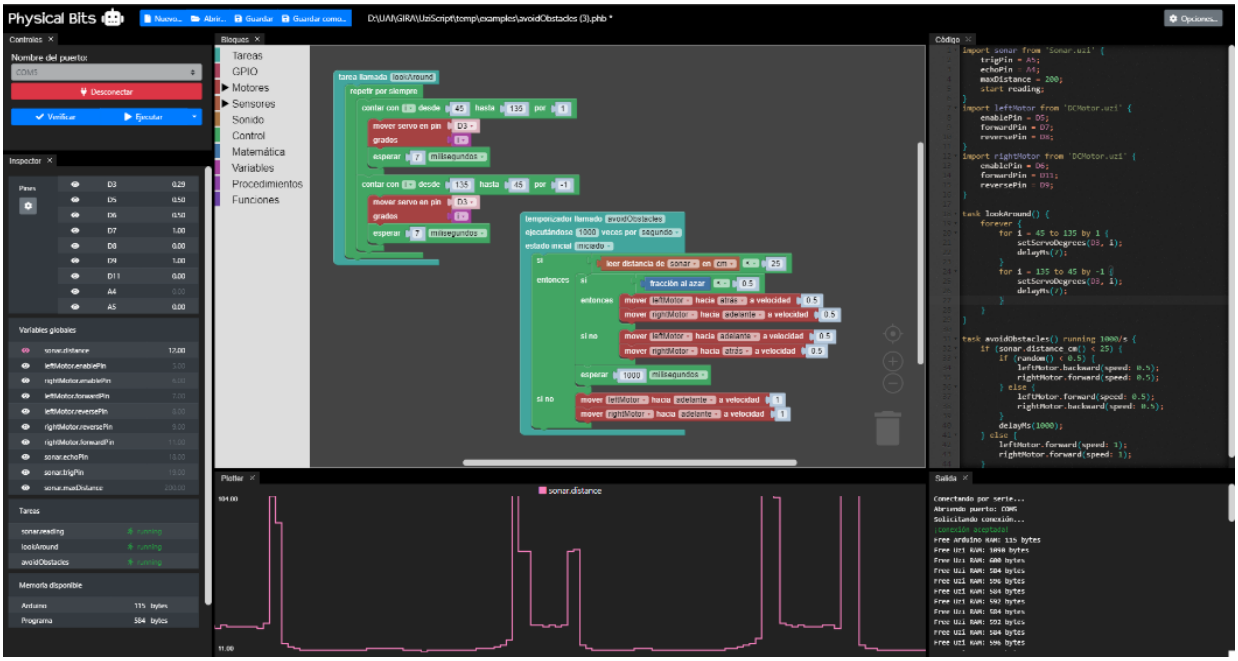
Introducción

En este artículo se describirán en detalle mis contribuciones al proyecto Physical Bits, radicado en el Centro de Altos Estudios en Tecnología Informática (CAETI) dentro de la línea de investigación “Sociedad del Conocimiento y Tecnologías Aplicadas a la Educación”.
Physical Bits es un entorno de desarrollo integrado (IDE) para robótica educativa que se propone como objetivo resolver una serie de problemas comunes que afectan, en mayor o menor medida, a todas las herramientas disponibles en la actualidad para programar robots en un contexto educativo.
El principal problema que se observa es que la interfaz de programación visual que presentan muchas de estas herramientas es efectiva para el aprendizaje en un estadío inicial, pero puede resultar contraproducente una vez que el estudiante deba incursionar en un lenguaje de programación de propósito general. Los estudiantes que pasan de un lenguaje de bloques a aprender un lenguaje de programación basado en texto pueden sufrir un problema denominado “sobrecarga de sintaxis” (Moors et al., 2018). Adicionalmente, los estudiantes que aprenden inicialmente con lenguajes basados en bloques pueden adquirir malos hábitos de programación que dificultan su capacidad para trabajar satisfactoriamente con lenguajes basados en texto (Meerbaum-Salant et al., 2011).
El segundo problema identificado está vinculado a la necesidad de programar dispositivos que se mueven e interactúan con objetos del mundo real. Al mantener separados el entorno de programación (la computadora del estudiante) del entorno de ejecución (el robot), resulta difícil fomentar un estilo de programación interactivo que reduzca el tiempo entre que el usuario realiza un cambio al programa y se observan los efectos del mismo. Acortar este “feedback loop” tiene efectos positivos sobre el aprendizaje ya que fomenta la experimentación y la prueba a la vez que disminuye los efectos de la frustración ante los errores (Lodi et al., 2019) (Weintrop & Wilensky, 2015).
El tercer problema se refiere a la utilización de abstracciones inadecuadas para el dominio del problema o el nivel de aprendizaje de los alumnos. Un ejemplo claro de este problema es el soporte inadecuado para la concurrencia que tienen algunos entornos. A pesar de que la mayoría de las actividades en robótica educativa requieren la programación de un dispositivo capaz de realizar dos o más tareas en forma simultánea, muchos lenguajes no proveen ningún mecanismo para expresar esta concurrencia de forma sencilla, dejando a los usuarios la responsabilidad de organizar y coordinar el trabajo, muchas veces mediante complicadas

máquinas de estado. Si bien el limitado soporte para concurrencia es una de las falencias más visibles en los entornos de programación para robótica educativa, no es el único caso de abstracciones inadecuadas. Dado que la inserción de robots en las aulas es un medio para ayudar a cumplir un objetivo mayor y no un fin en sí mismo (Barrera Lombana, 2015), resulta crucial evitar distraer al alumno con complejidad innecesaria de forma que pueda concentrarse en aquellos conceptos fundamentales para el aprendizaje (Lopez et al., s. f.).

Physical Bits propone resolver estos problemas mediante la implementación de una serie de principios de diseño orientados a atacar cada uno de los problemas mencionados anteriormente.
Estos principios de diseño son:

1. Autonomía del robot: Los programas se almacenan y ejecutan directamente en el robot sin necesidad de comunicación con una PC.
2. Programación interactiva: Los cambios al programa se envían al robot inmediatamente, acortando el "feedback loop".
3. Monitoreo y adquisición de datos: El entorno provee herramientas para visualizar el estado interno del robot y del programa.
4. Modelo híbrido bloques/código: El diseño del lenguaje de programación facilita una transición gradual desde lenguajes de bloques hacia lenguajes de propósito general.
5. Soporte para concurrencia: El lenguaje permite definir tareas concurrentes sin requerir la coordinación manual de las acciones del robot.
6. Herramientas de depuración: El entorno incluye herramientas de depuración que permiten detener la ejecución del programa, ejecutar instrucciones paso a paso, y observar su impacto en el estado del programa.



Como parte de este proyecto, mis contribuciones se centraron principalmente en ampliar la funcionalidad del entorno, mejorar la interfaz gráfica para hacerla más accesible a usuarios sin experiencia, y adaptar el lenguaje de programación para su futura utilización en el proyecto Mendieta URPE.

Las contribuciones se pueden resumir de la siguiente manera:

1. Posicionamiento de los bloques luego de modificar el código
2. Configuración de la interfaz en modos básico y avanzado
3. Notificaciones “toast” para optimizar el uso del espacio en la pantalla
4. Incorporación de strings en el lenguaje de programación
5. Soporte para displays LCD usando el módulo I2C

A continuación, se describe cada una de estas contribuciones en detalle.

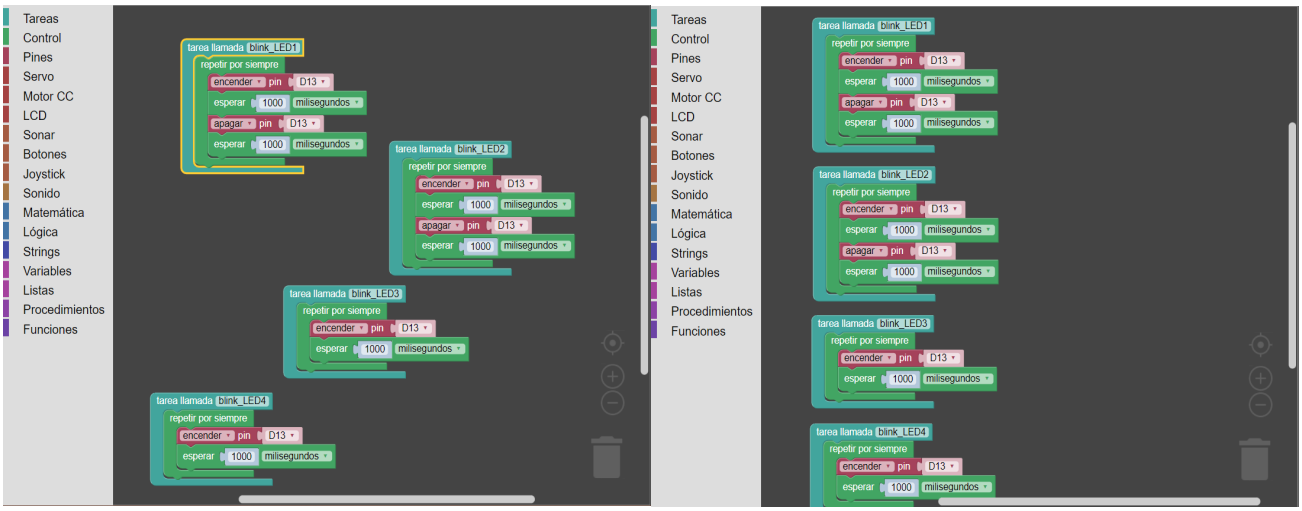
Contribuciones

Posición de Bloques

Physical Bits ofrece la flexibilidad de programar tanto mediante bloques como directamente mediante la escritura de código. En su interfaz de pantalla, se presentan dos paneles representando los bloques y el código.

La singularidad de esta herramienta radica en su capacidad para mantener una sincronización entre la programación por bloques y la escritura de código. Cada vez que se realiza una modificación en uno de estos dos modos de programación, se activa automáticamente la conversión correspondiente.

En versiones anteriores, al modificar el código, los bloques se regeneran sin tener en cuenta ningún dato del estado previo de los mismos, lo que resultaba en que todos los bloques volvieran a su posición predeterminada:



Esta limitación afectaba negativamente a la experiencia del usuario, ya que daba la impresión de que todos los bloques se modificaban en lugar de solo aquellos relacionados con el código alterado.

Para abordar esta problemática, se implementó una solución que implica el almacenamiento de información sobre los bloques padres previamente existentes, incluyendo sus nombres y posiciones. Al conservar esta información, cuando se generan nuevos bloques, se verifica si

corresponden a los bloques anteriores, identificándolos a partir del nombre y, en caso afirmativo, se utiliza la posición almacenada. Se siguió la siguiente lógica:

Dentro del código, se encuentran diversos bloques, pero nuestra atención se centró principalmente en los denominados "Top Blocks", ya que estos definen la posición de los bloques que dependen de ellos. Para preservar esta información esencial, se requirió la asignación de un identificador único a cada bloque. Se optó por emplear el nombre del bloque, ya que es el único identificador persistente a través de las modificaciones del programa. En caso de que el nombre de un "Top Block" cambie, se considera como un nuevo bloque y no se vincula con la información previamente guardada en cuanto a las posiciones.

La estructura de almacenamiento adoptada se presenta de la siguiente manera:

```
topBlocks = {
        'block_1' : {
            'x': x1,
            'y': y1
        },
        'block_2' : {
            'x': x2,
            'y': y2
        }
}
```

Cada vez que se deben generar nuevamente los bloques, la primera tarea consiste en recuperar y almacenar las posiciones actuales de los "Top Blocks", calculando las coordenadas relativas al bloque anterior para cada uno.

Luego se configura la posición inicial en cero, lo que hace referencia a la esquina superior izquierda del panel de bloques, y se definen dos variables de posición, X e Y, ambas establecidas en 0.

La siguiente tarea es recorrer los bloques determinando las nuevas posiciones, por cada uno se pregunta si el nombre de ese bloque se encuentra en los datos almacenados. Si se encuentra el nombre, se utiliza la información guardada junto con la suma acumulada en la variable correspondiente para definir la posición de ese bloque tanto en X como en Y. En caso de que el bloque no se encuentre en los datos almacenados, lo que indica que es un bloque nuevo, se mantiene la misma posición X que el bloque anterior y se agrega la altura predeterminada de espacio entre dos bloques a la variable Y, que es de 24, para utilizar la misma.

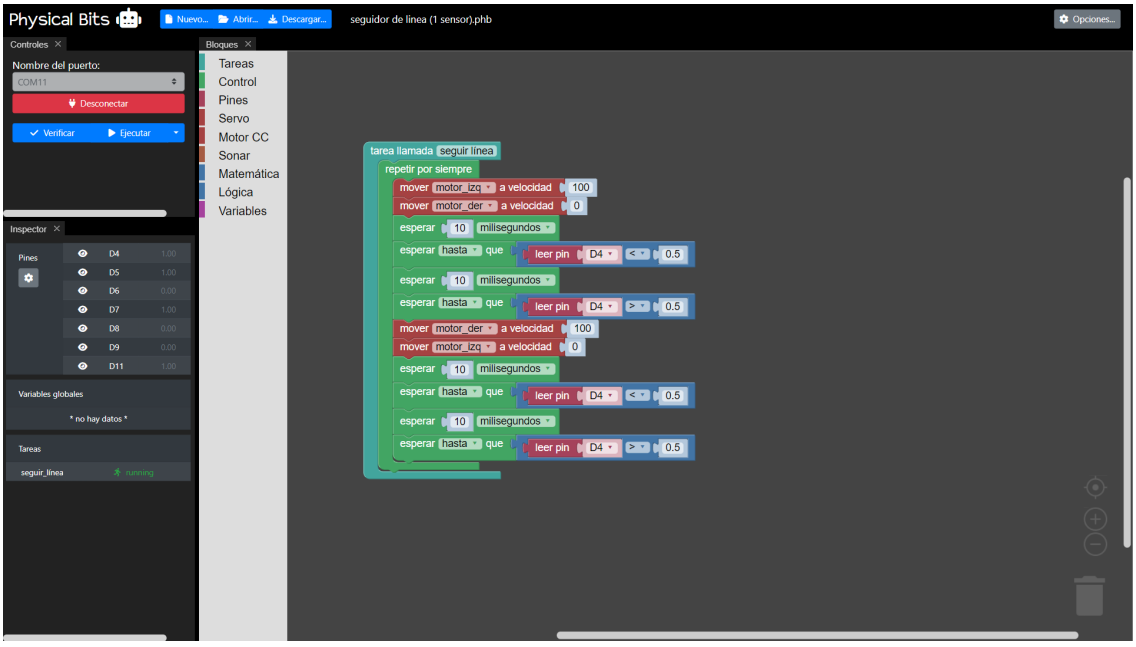
Después de definir la posición de un bloque, ya sea uno previamente almacenado o nuevo, se suma la altura del bloque a la variable Y. De esta manera, se mantiene la altura de todos los bloques en relación con el bloque anterior, representando la posición de la parte inferior del último bloque generado en el eje Y.

Es importante destacar que, si bien no tiene un impacto fundamental en el funcionamiento final del programa, la capacidad de mantener información sobre la posición de los bloques anteriores resulta valiosa tanto para los programadores como para la experiencia de usuario.

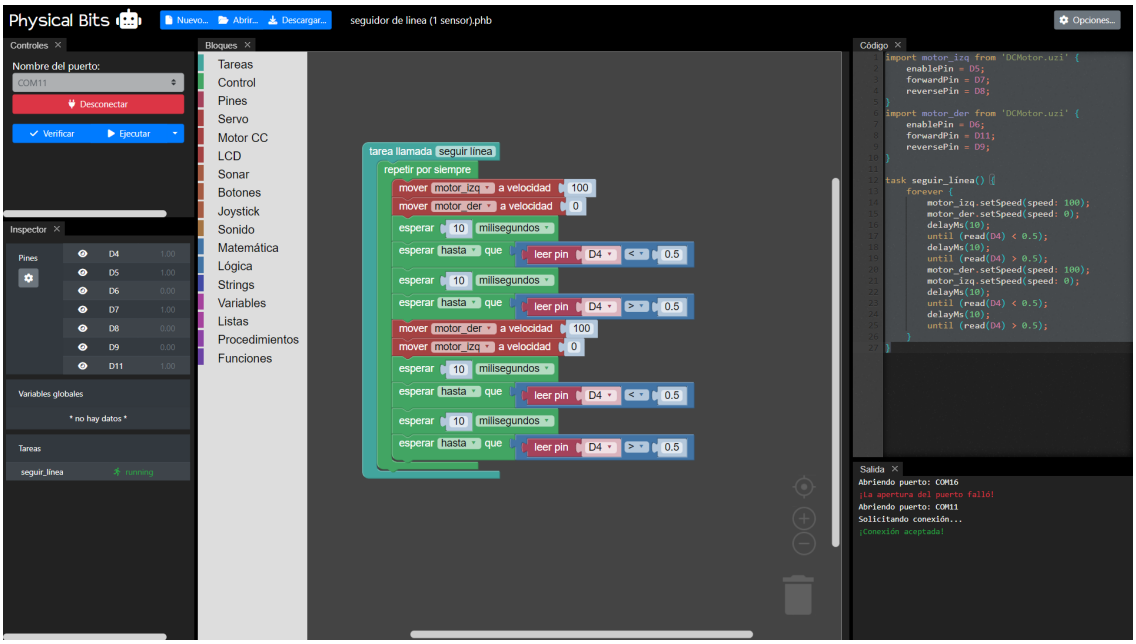
Interfaz Básica/Avanzada

La implementación de una interfaz básica y otra avanzada surgió en respuesta a la creciente cantidad de funciones que presenta en la actualidad Physical Bits. Con la adición de nuevas características y capacidades, la interfaz única previa empezó a verse potencialmente abrumadora para aquellos usuarios que no están familiarizados con el sistema o con la programación en general.
La interfaz básica proporciona simplicidad y accesibilidad, mientras que la avanzada ofrece funciones más completas y detalladas, estando habilitados todos los paneles, para aquellos usuarios con mayor experiencia.

Interfaz Básica:



Interfaz Avanzada:



Los paneles con los que cuenta la plataforma y sus funcionalidades son los siguientes:

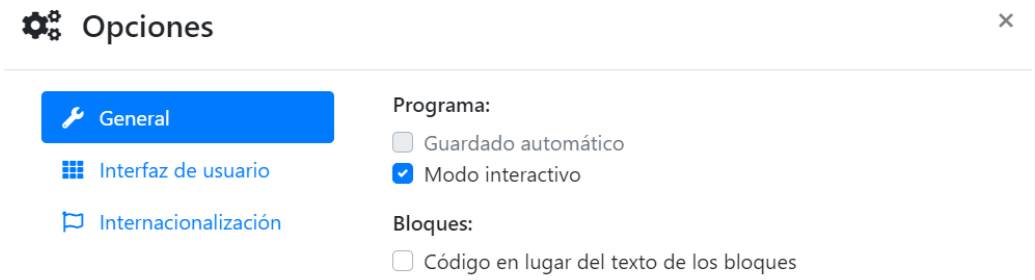
- Controles: Este panel permite la conexión al programa además de la opción de verificar y ejecutar.
- Inspector: El panel del Inspector nos proporciona información detallada de los elementos y propiedades dentro del programa.
- Bloques: En este panel, se encuentra la versión en bloques del programa, siendo uno de los elementos centrales de la plataforma.
- Código: Este panel permite la edición directa del código del programa.
- Graficador: Nos muestra un gráfico de los valores de los pines.
- Depurador: Este panel permite llevar un control de las variables, ejecuciones de tareas, entre otras cosas, que ayudan al usuario a encontrar errores en su código.
- Salida: Este panel muestra los diversos mensajes de compilación del programa. No se planteó para ninguna de las interfaces ya que se reemplazó su funcionalidad con notificaciones.

Inicialmente, se planteó la diferenciación entre modos "básico" y "avanzado" únicamente para el panel de Bloques. Se categorizaron los distintos bloques y se implementó un checkbox que permitía actualizar el ToolBox trayendo todos los bloques o filtrándose. Para ello, se crearon dos archivos que contenían los bloques correspondientes a cada modo.

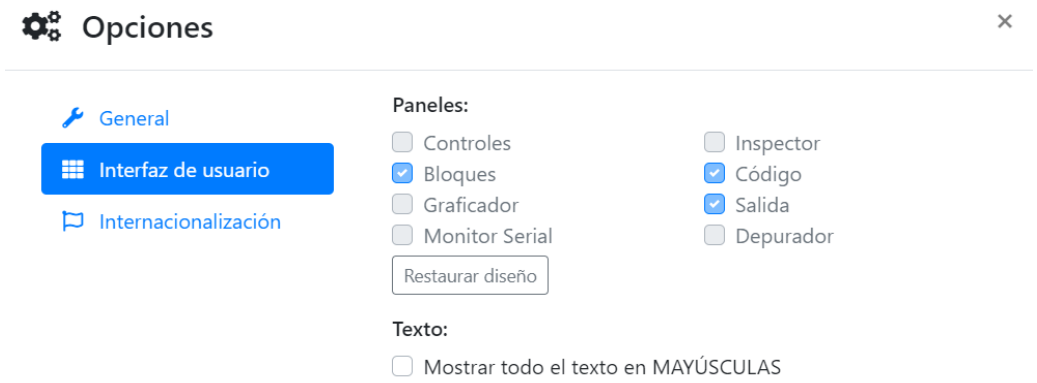
Sin embargo, posteriormente, se decidió extender esta diferenciación a todas las funcionalidades, no solo al ToolBox. Se definió que los paneles Bloques (con el ToolBox básico), Controles e Inspector se considerarían como básicos, mientras que Bloques (con el ToolBox avanzado), Código, Graficador y Depurador se clasificarían como avanzados.

El control de la interfaz se realiza a través del modal de opciones ubicado en la esquina superior derecha de la pantalla. Este modal también sufrió modificaciones significativas. Anteriormente, constaba de tres secciones:

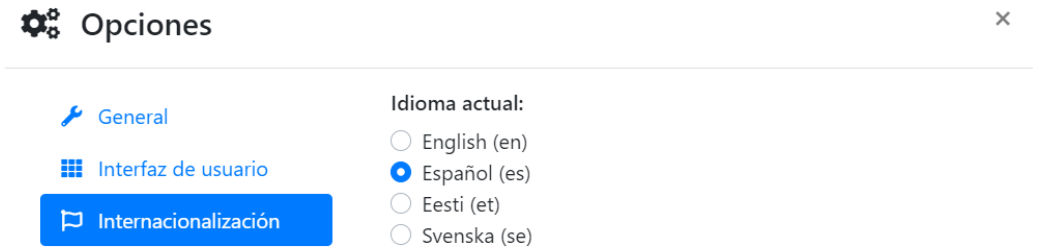
- General: Esta sección contaba con tres opciones. Dos de estas relacionadas al programa, un check que habilitaba el guardado automático, y otro el modo interactivo. Y el último permitía modificar el formato de texto de los bloques.



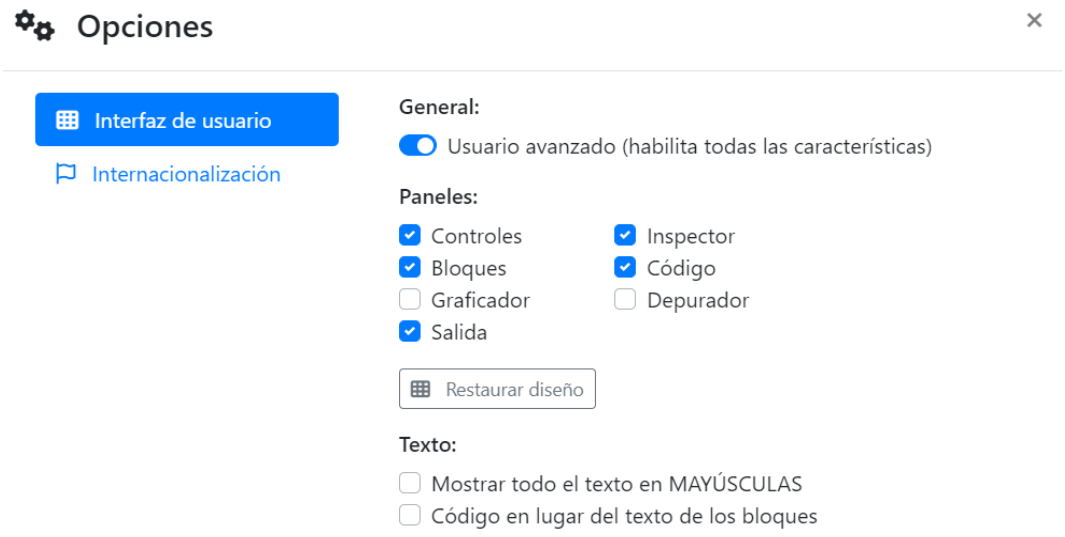
- Interfaz de Usuario: Esta sección mostraba una lista de paneles deshabilitados, utilizados para determinar qué paneles estaban abiertos al momento de consultarlos. También, debajo de estos, se encontraba un botón que restauraba el layout por defecto y una opción para convertir todos los textos de la interfaz a mayúsculas.



- Internacionalización: Esta sección facilita el cambio de idioma.



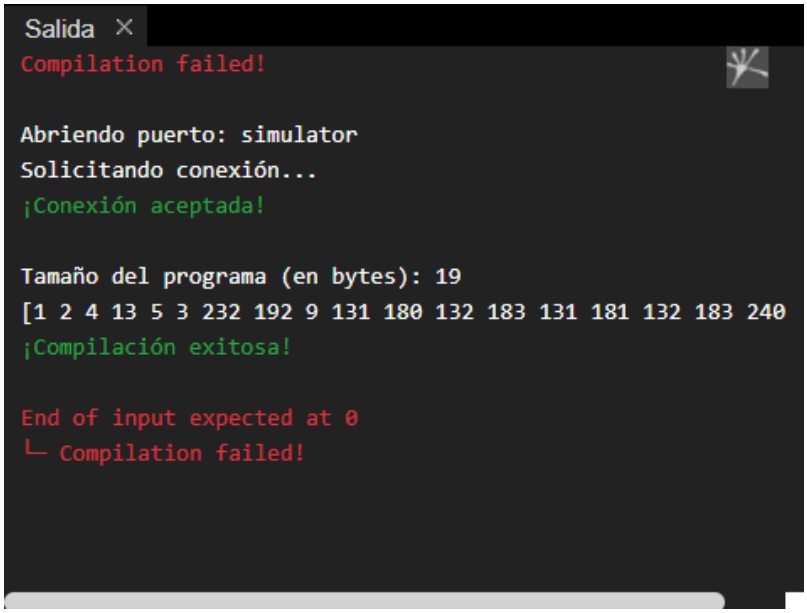
Tras un análisis de cada una de estas funcionalidades, se decidió conservar sólo dos secciones: Interfaz de Usuario e Internalización. La única característica de la sección General que se mantuvo fue la opción para cambiar el formato de texto de los bloques, que se trasladó a la sección Interfaz de Usuario. La sección Internalización se mantuvo sin cambios, mientras que la de Interfaz de Usuario experimentó varias modificaciones. Se agregó un radiobutton antes de la lista de paneles para alternar entre los modos Básico y Avanzado, que incluye la actualización del ToolBox. Además, los checkboxes de los paneles se habilitaron para permitir la apertura y cierre de paneles directamente desde el modal. Por último, se conservó el checkbox de la sección General. La vista final del modal se presenta en la siguiente imagen:



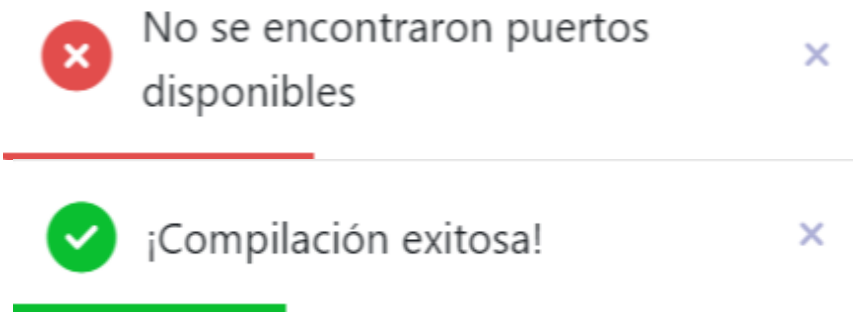
Este enfoque optimizado de la interfaz permite a los usuarios ajustar la plataforma según sus necesidades y preferencias, mejorando así la experiencia de usuario.

Notificaciones 'Toast'

El layout de la página incluye varios paneles, entre los cuales se encuentra el panel de Salida (Output). Una de sus funciones principales consistía en mostrar al usuario diversos mensajes para indicar tanto operaciones exitosas como errores de compilación, junto con las excepciones correspondientes a dichos errores.



Debido a la distinción actual entre la interfaz básica y la avanzada, se identificó que las excepciones no deberían formar parte de la versión básica. Esto llevó a sacar las excepciones e información de compilación fuera de los mensajes del output.
Este enfoque resulta más claro para el usuario lo que llevó a implementarlo también en la versión avanzada.
Al reducir las notificaciones en el panel de Salida a mensajes simples, se ha permitido la implementación de elementos de tipo Toast para agregar dinamismo y mejorar la experiencia del usuario, haciéndola más amigable e interactiva.



Para estos se definió un tiempo de persistencia por default pero puede varias dependiendo del mensaje en caso de ser necesario. También se especificaron 4 tipos de mensaje:

- Éxito (verde)
- Error (rojo)
- Información (azul)
- Advertencia (naranja)

Implementación de strings en el lenguaje de programación

En sus primeras versiones, Physical Bits carecía de un lenguaje de programación capaz de representar y manipular cadenas de texto (strings). Aunque pueda parecer sorprendente que un lenguaje no admitiera el uso de cadenas de texto, en el contexto para el cual se diseñó Physical Bits, la ausencia de esta funcionalidad, aunque limitante, no impedía la realización de numerosas experiencias de robótica.
Normalmente, las actividades en robótica educativa consisten en la programación de un robot autónomo para que realice alguna tarea específica. Esta tarea usualmente requiere el sensado del ambiente, la toma de decisiones, y una acción correspondiente. A diferencia de los lenguajes tradicionales, que requieren comunicar resultados al usuario muchas veces mediante texto, la forma en que los robots comunican resultados es mediante acciones en el ambiente que los rodea. Por esta razón, y para evitar el costo adicional en almacenamiento y complejidad, las primeras versiones de Physical Bits se publicaron sin soporte para strings.

Sin embargo, la falta de strings entra en contradicción evidente con uno de los objetivos explícitos de Physical Bits: facilitar la transición gradual en el aprendizaje de los alumnos a lenguajes de programación de propósito general. Por esta razón, se tomó la decisión de implementar strings en el lenguaje.
Debido a la limitada capacidad de la memoria RAM de las placas Arduino UNO, se decidió restringir la funcionalidad de los strings. El lenguaje no permite la creación dinámica de strings. Todos los strings deben estar definidos estáticamente de manera que el compilador pueda precalcular el espacio de almacenamiento requerido. Esto significa que operaciones como concatenación o conversión de números a strings no están soportadas.

En lo que respecta a la implementación, se decidió utilizar NULL-terminated strings por la simplicidad y compatibilidad con el lenguaje C. Se modificó la máquina virtual de Physical Bits para almacenar los strings del programa en un array de caracteres. Luego, todas las referencias a strings dentro del programa se representan en forma de índices dentro de este array. Es decir, que si una variable apunta a un string el valor almacenado dentro de esta variable es en realidad el índice correspondiente al primer caracter del string referenciado. Esto tiene como consecuencia que la máquina virtual no distingue entre variables que referencian strings de variables que contienen valores numéricos. Esta misma limitación ya existía previamente en el lenguaje en lo que respecta a valores que apuntan a estructuras de datos como arrays, listas, u otros tipos de objetos. En un futuro sería deseable ampliar la implementación del lenguaje para que los objetos incluyan un tag que permita identificar el tipo de dato dinámicamente, como hacen muchos otros lenguajes de programación.
Además de modificar la máquina virtual para que pueda almacenar los strings, fue necesario modificar el parser para que pueda entender la sintaxis nueva de los strings y el compilador para que pueda emitir las instrucciones correctas que permitan crear y manipular los strings. Asimismo, se modificó el formato del programa compilado (los bytecodes) para que incluyan los strings declarados en el programa.
Finalmente, fue necesario también modificar el entorno de programación para que el editor incluya bloques nuevos que representen strings. Afortunadamente, dado que la sintaxis elegida para los strings es muy popular en lenguajes de programación, el editor de código utilizado por

el entorno de Physical Bits ya incluía syntax highlighting para los strings y no hubo necesidad de modificar esa parte del código.

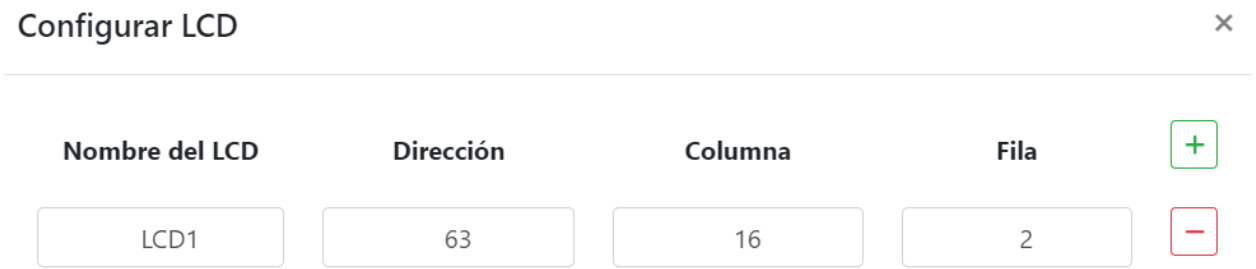
Soporte para control de display LCD usando I2C

El display LCD tenía funcionalidad previa en el programa, pero estaba restringido a la visualización de números debido a la falta de interpretación de strings en el lenguaje. Además de que se limitaba a utilizarse desde el código, ya que no se disponía de bloques ni de una interfaz gráfica para configurar los displays.
Inicialmente, esta limitación se mantuvo para permitir la prueba de su funcionamiento, con la intención de habilitar su uso con strings y bloques una vez que esto estuviera disponible.

Tras la implementación de strings en el lenguaje y la creación del bloque correspondiente, se volvió posible realizar las modificaciones necesarias para permitir la visualización de texto en el display LCD.

Se comenzó por agregar la sección "LCD" al Toolbox, junto con un botón destinado a su configuración. Al presionar este botón, se abre un modal en el cual se debe ingresar la información correspondiente al display.
La información que podemos configurar es la siguiente:

- Nombre: Nos permite identificar al LCD, más que nada en los casos que haya más de uno.
- Dirección: Se refiere a la dirección de comunicación utilizada para establecer una conexión con el módulo de pantalla LCD.
- Columna: Este parámetro se utiliza para especificar el ancho del display en cantidad de caracteres (16 por defecto).
- Fila: Este parámetro determina el alto del display en cantidad de caracteres (2 por defecto).

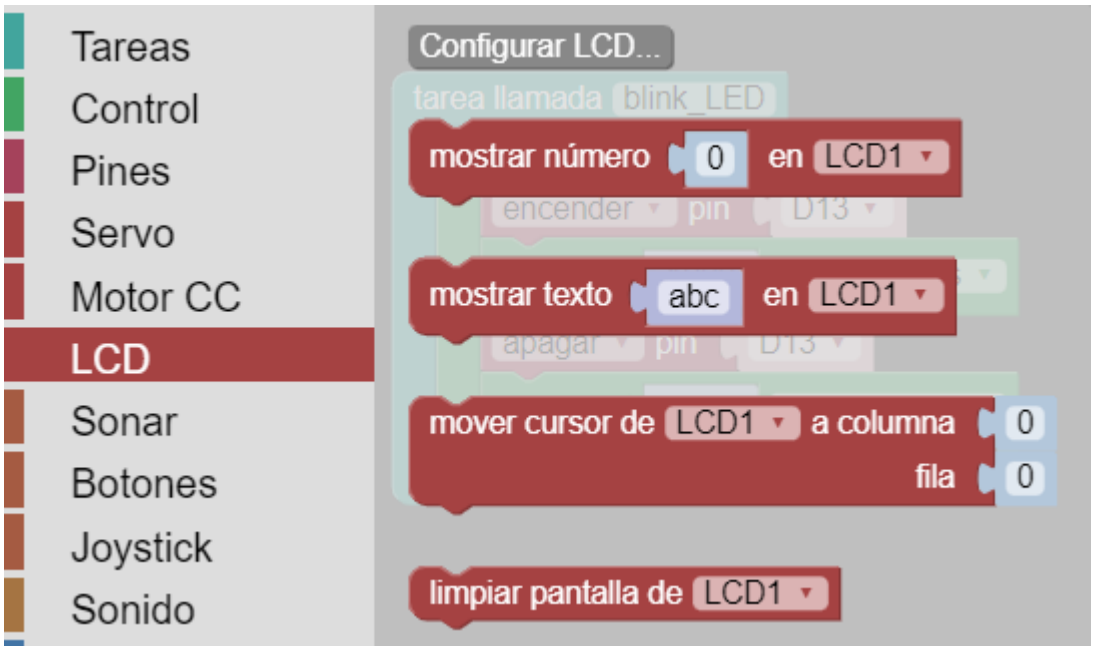


Cuando el usuario ingresa al menos la información de un display, se hacen visibles cuatro bloques específicos: "Mostrar número", "Mostrar texto", "Mover cursor" y "Limpiar pantalla". Estos bloques han sido diseñados exclusivamente para esta sección.

Los primeros dos bloques, "Mostrar número" y "Mostrar texto", permiten utilizar el display LCD para mostrar números y texto respectivamente, ya que como se mencionó anteriormente, el programa no puede distinguir entre variables con valor string y valor numérico. Ambos bloques requieren dos datos para funcionar adecuadamente. El primer dato corresponde al LCD que se desea utilizar, especialmente útil cuando hay múltiples disponibles. El segundo dato se utiliza para especificar el número o el texto que se desea mostrar en el display.

A continuación, se encuentran los siguientes dos bloques. Uno de ellos, "Mover cursor", proporciona control sobre la posición del cursor en el LCD y requiere tres datos: el LCD que se está modificando, así como las nuevas coordenadas de columna y fila.

Por último, tenemos el bloque "Limpiar pantalla", que tiene la función de borrar el contenido de la pantalla del LCD. Este bloque solo requiere la especificación del LCD correspondiente.



Cuando se emplea uno de estos bloques, no solo se genera el llamado a la función correspondiente en el código, sino que también se incluye automáticamente la configuración del LCD en la parte superior. Esto facilita al usuario la capacidad de modificar dicha configuración directamente desde el mismo código, como se ejemplifica a continuación:

```
import LCD1 from 'LCD_I2C.uzi' {
        address = 63;
        cols = 16;
        rows = 2;
}

task default() {
        forever {
                LCD1.printNumber(3);
                LCD1.printString('abc');
        }
}
```

No sólo se agregaron los bloques sino que hubo que agregar primitivas a la máquina virtual para interactuar con el display LCD. Se agregó también una librería en el lenguaje de Physical

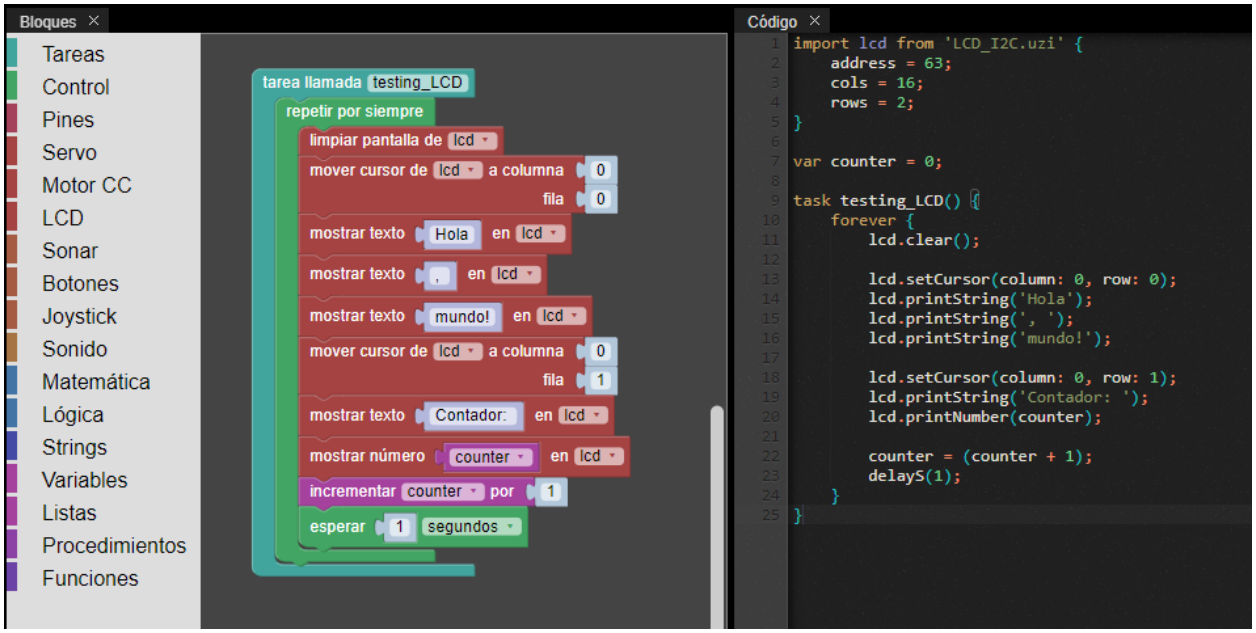
Bits, la cual encapsula las primitivas y se asegura que el display esté correctamente inicializado antes de mostrar texto.

Como se puede ver en el código, se está importando la librería LCD_I2C.uzi, y pasándole los parámetros correspondientes, configurados anteriormente, para definir el LCD. Este posee los distintos métodos previamente mencionados en referencia a los bloques.
El código completo de la librería es el siguiente:

uzi > libraries > LCD_I2C.uzi

```
var lcd = 0;
var address = 63; "NOTE(Richo): 39 is a common address too"
var cols = 16;
var rows = 2;

prim lcd_init0;
prim lcd_init1;
prim lcd_print_num;
prim lcd_print_str;
prim lcd_clear;
prim lcd_set_cursor;

task setup() {
  lcd = lcd_init1(lcd_init0(address, cols, rows));
}

func ready() {
  return lcd > 0;
}

proc printNumber(value) {
  until lcd;
  lcd_print_num(lcd, value);
}

proc printString(value) {
  until lcd;
  lcd_print_str(lcd, value);
}

proc setCursor(column, row) {
  until lcd;
  lcd_set_cursor(lcd, column, row);
}

proc clear() {
  until lcd;
  lcd_clear(lcd);
}
```

Un ejemplo de la ejecución exitosa de un programa utilizando esta librería y sus distintas funciones:



Conclusiones y Trabajo Futuro

Las contribuciones previamente analizadas fueron integradas satisfactoriamente al proyecto y se encuentran disponibles en el repositorio del proyecto[1]. Todavía no se ha publicado un release que contenga la última versión del código pero se espera poder hacerlo en los próximos meses, una vez hayan concluido las pruebas exhaustivas que se están llevando a cabo.

Se considera que los cambios realizados al proyecto son de vital importancia, sobre todo los últimos 2 (implementación de strings y soporte para display LCD), dado que estos son un requerimiento clave para Mendieta URPE, otro proyecto que está actualmente en desarrollo en el CAETI por el mismo grupo de investigación que diseñó Physical Bits.

Mendieta es un robot autónomo de bajo costo controlado mediante un Arduino Nano, con un servidor web incorporado en una computadora Orange Pi, que permite la programación de todo un curso al mismo tiempo, encolando los pedidos de ejecución de manera tal que con un único dispositivo se pueda llevar adelante la clase en forma dinámica. El entorno que utilizará el proyecto Mendieta para la programación del robot será una versión modificada de Physical Bits. Por esta razón, los cambios detallados en este artículo son cruciales para poder aprovechar las características de Mendieta.

Como trabajo futuro, además del proyecto Mendieta URPE, se espera poder continuar la implementación de Physical Bits adaptando su código para poder soportar otros tipos de hardware (como son las placas ESP32 y las nuevas versiones de Arduino).

[1] Repositorio del proyecto: https://github.com/GIRA/PhysicalBits

Bibliografía

Barrera Lombana, N. (2015). USO DE LA ROBÓTICA EDUCATIVA COMO ESTRATEGIA DIDÁCTICA EN EL AULA. *Praxis & Saber*, *6*(11), 215-234.

Lopes Guedes, A., Lopes Guedes, F., & Guedes Laimer, A. C. (2015). Experiencias de robótica educativa / Experiences with Educational Robot. *Revista Internacional de Tecnología, Ciencia y Sociedad*, *4*(2). https://doi.org/10.37467/gka-revtechno.v4.887

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of programming in scratch. *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education - ITiCSE '11*, 168. https://doi.org/10.1145/1999747.1999796

Moors, L., Luxton-Reilly, A., & Denny, P. (2018). Transitioning from Block-Based to Text-Based Programming Languages. *2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*, 57-64. https://doi.org/10.1109/LaTICE.2018.000-5

Moran, R., Teragni, M., & Zabala, G. (2021). Physical Bits: A Live Programming Environment for Educational Robotics. En W. Lepuschitz, M. Merdan, G. Koppensteiner, R. Balogh, & D. Obdržálek (Eds.), *Robotics in Education* (pp. 291-303). Springer International Publishing. https://doi.org/10.1007/978-3-030-67411-3_26

Weintrop, D., & Wilensky, U. (2015). To block or not to block, that is the question: Students' perceptions of blocks-based programming. *Proceedings of the 14th International Conference on Interaction Design and Children*, 199-208. https://doi.org/10.1145/2771839.2771860