UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA

GRADO EN INGENIERÍA INFORMÁTICA EN INGENIERÍA DEL SOFTWARE

TRABAJO FIN DE GRADO

Desarrollo de una versión web de la herramienta educativa LearnBlock

UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA

NOMBRE DE LOS ESTUDIOS

TRABAJO FIN DE GRADO

Desarrollo de una versión web de la herramienta educativa LearnBlock

Autor: Rebeca García Moreno

Tutor: Pilar Bachiller Burgos

Resumen

A lo largo de los años, la tecnología ha ido abriéndose paso poco a poco en la sociedad hasta llegar a formar parte de la vida cotidiana de las personas. El número de avances científicos y tecnológicos aumenta cada vez más, y se prevee que en los próximos años continúe creciendo de la misma manera. Por tanto es recomendable, incluso necesario, tener la capacidad de poder adaptarse con facilidad a lo que está por venir.

La educación es uno de los ámbitos que está evolucionando para otorgar dicha capacidad a las nuevas generaciones. Cada vez es más común que las instituciones de enseñanza se hagan con herramientas de aprendizaje con las que los alumnos puedan establecer un contacto más profundo con las tecnologías que les rodean. Estas herramientas ofrecen distintas maneras de comprender esas tecnologías de una forma sencilla y de experimentar con ellas.

En lo relacionado con la programación, es habitual que la educación recurra a la utilización de robots como medio de enseñanza, pues es el instrumento que mejor muestra los resultados de los programas desarrollados. A su vez, la mayoría de herramientas educativas que hacen uso de robots ofrecen un tipo de programación basado en bloques que resulta bastante intuitivo para el usuario, y que se conecta directamente con el control del robot.

LearnBlock es una herramienta de aprendizaje de programación que se ajusta a estas características. Ofrece al usuario una interfaz de programación por bloques y permite la conexión con un robot que ejecute el resultado de los programas desarrollados. Proporciona un método de aprendizaje de programación progresivo

que va desde un lenguaje visual hasta un lenguaje profesional. Buena parte de los bloques disponibles en LearnBlock están orientados a controlar las acciones del robot conectado, pero también se permite la creación de nuevos bloques que definan comportamientos distintos para el robot a partir de una función en Python desarrollada por el propio usuario. La novedad de LearnBlock es que, a día de hoy, es la única herramienta existente que permite que el mismo código pueda ser ejecutado en distintos robots, independientemente de si se trata del código visual o del código generado.

LearnBlock está en continuo desarrollo. Ha sido creado por investigadores de la Universidad de Extremadura (RoboLab).

El proyecto presentado surge de la idea de hacer que LearnBlock pueda llegar a la mayor cantidad de personas posible de manera sencilla. Se trata de una versión online del mismo que posee una estructura visual similar y comparte su funcionamiento y arquitectura.

LearnBlockWeb abarca las funcionalidades de LearnBlock que están relacionadas con la programación visual mediante bloques y con la generación de código. De esta manera, se le proporciona al usuario todo lo relacionado con el aprendizaje de programación que ofrece LearnBlock.

El objetivo de LearnBlockWeb es dar a conocer el proyecto LearnBlock de primera mano, permitiendo que los usuarios hagan una toma de contacto inicial y descubran de forma fácil y rápida parte de lo que la herramienta puede ofrecerles. Al ser una página web, no existe ningún requisito previo ni necesidad alguna de instalar nada. De lo único que se debe disponer es de conexión a internet y de un navegador desde el que poder acceder a ella.

Overview

Through the years, technology has been making its way into society to the point of being part of people's daily lives. The number of scientific and technological advances is increasing more and more, and it's expected that in the following years it continues to grow in the same way. Therefore it's recommendable, even necessary, being able to easily adjust to what is still to come.

Education is one of the scopes that's evolving to give that skill to the new generations. It's becoming more common that educational institutions get hold of learning tools that allow students to establish a deeper contact with the technology around them. These tools provide different ways to simply understand that technology and to experiment with them.

With regards to programming, it's usual in education to use robots as a teaching way, since it's the tool that better represents the developed program's results. In turn, most of the learning tools that use robots, provide block-based programming, which is highly intuitive to the user, and it's directly connected to the robot's control.

LearnBlock is an educational programming tool that suits these features. It provides a graphical user interface where programs made with blocks can be built, and allows the connection to a robot that executes the results of those programs. It grants a way of progressive programming learning, that starts with a visual language and ends with a professional language. Most of the available blocks in LearnBlock are aimed to control the connected robot's actions, but it's also allowed to create new blocks that define different behaviours for the robot from a Python function that can be developed by the user itself. The novelty of LearnBlock is that, nowadays, it's the only tool that allows

the same code to be executed in different robots: LearnBlock is robot-agnostic, and that property is guaranteed for both visual and generated code.

LearnBlock is under permanent development. It has been created by researches from the University of Extremadura (RoboLab).

The submitted project comes up from the idea of making LearnBlock a more accessible tool. This is an online version that has been made with the same visual structure as LearnBlock and that shares the same properties and architecture.

LearnBlockWeb takes account of all the LearnBlock functionalities related to visual programming using blocks and code generating. That way, it provides to the user everything related to programming learning that also provides LearnBlock.

LearnBlockWeb's goal is to make LearnBlock more known, allowing users to make a first contact and quickly and easily let them discover some aspects that the tool could provide. There aren't previous requirements and there's no need for any kind of software installation. The user only needs an internet connection and a web browser to use this web page.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Las tecnologías avanzan a pasos agigantados. Cada vez son más los progresos y las investigaciones tecnológicas que se llevan a cabo en el mundo, y todas ellas se fundamentan en la búsqueda del bienestar humano, ya sea de forma directa o indirecta. Gran parte de los desarrollos están tan asentados en la sociedad que a las personas les resulta difícil imaginar cómo serían sus vidas sin las facilidades que éstos aportan. Al final, saber utilizar de forma correcta estas tecnologías se convierte en algo clave para asegurar tal bienestar.

Teniendo en cuenta que dichos progresos aumentan de forma significativa con el paso de los años, acabará por resultar fundamental el poder habituarse fácilmente y con rapidez a cualquier cambio y avance que surja. Para ello, es indispensable ser capaz de ver el mundo de una manera lógica y aritmética, tal como lo hace una máquina, y conseguir encontrar soluciones a problemas que puedan ser representadas en el lenguaje de la informática, con instrucciones, algoritmos y programación. Esto es lo que se conoce como *pensamiento computacional*.

Hoy en día, tener tal habilidad es algo que está siendo cada vez más valorado. El pensamiento computacional se puede aplicar para resolver problemas de diversas disciplinas de una manera organizada y eficiente. Esa es la razón por la que, desde hace un tiempo, se esté promoviendo su desarrollo desde edades tempranas. Actualmente, la tecnología está muy unida a la educación, de forma que resulta común la progresiva

inclusión en materia de enseñanza de métodos y herramientas que permitan a los alumnos obtener esa habilidad y desarrollarla de manera fácil, continua y gradual.

Aunque los términos *pensamiento computacional* y *programación* no son lo mismo, sí que están estrechamente relacionados. Aprender los fundamentos de la programación y saber aplicarlos hace que, eventualmente, una persona termine adquiriendo todas las aptitudes que caracterizan al pensamiento computacional. Por lo tanto, es común que la mayoría de herramientas creadas para la adquisición de esta destreza estén basadas en enseñar a programar partiendo de un nivel de abstracción muy alto con el que el proceso de aprendizaje resulte sencillo e intuitivo.

Una de las maneras más fáciles de aprender a programar es hacerlo mediante el lenguaje de programación por bloques. Un bloque es una pieza que se corresponde con un elemento de programación: estructuras de control, funciones, variables... Cada pieza tiene una forma distinta según lo que represente, y puede encajarse con otras piezas como si se tratara de un puzzle. Que se puedan encajar o no depende de lo que represente cada pieza; este tipo de programación limita al usuario a unir solamente las que equivalgan a estructuras que tienen sentido estando juntas. Al final, se termina asociando cada bloque con la estructura que le corresponde y se aprende su semántica y su funcionamiento.

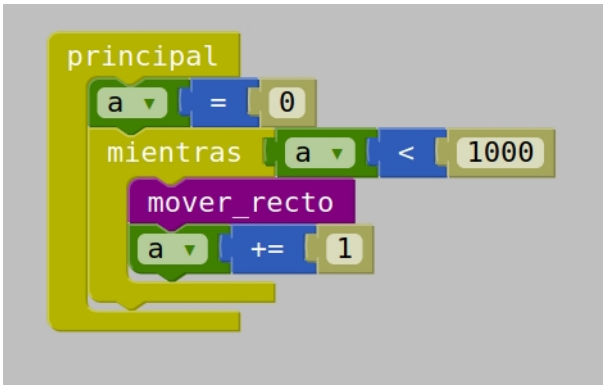


Figura 1.1: Ejemplo de programación por bloques

Muchas de las herramientas que utilizan este tipo de aprendizaje, además, están complementadas por algún instrumento con el que se representa de forma visual el resultado de los programas creados con los bloques, ya sea una imagen interactiva o

un robot físico. Algunos ejemplos de éstas son Blockly, Scratch, Alice o TurtleArt.

El proyecto LearnBlock [1] nació con el objetivo de ir un paso más allá y ofrecer una nueva herramienta educacional de programación que cubriera las limitaciones de todas las herramientas existentes hasta la fecha. Si bien comparte algunas propiedades con todas ellas, otras resultan completamente novedosas.

LearnBlock ha sido desarrollado como una aplicación de escritorio, al contrario que la mayoría de herramientas educativas similares. La razón es que, en algunos casos, es necesaria una conexión local entre la aplicación y una plataforma robótica, y a veces se requiere instalar en el equipo algún tipo de software con el que poder usarlo. Sin embargo, que se haya creado como una aplicación local hace que no goce de los mismos beneficios que el resto de herramientas análogas que, en cambio, se han desarrollado para un entorno web.

LearnBlockWeb se ha desarrollado con el objetivo de otorgar a LearnBlock tales beneficios.

En lo referente a LearnBlock, todas las características que posee se pueden dividir en tres funcionalidades claramente diferenciadas: Programación en bloques, generación de código y ejecución del código. De esas tres, LearnBlockWeb abarca las dos primeras, con las que se cubre la parte que se centra en el aprendizaje de programación progresivo.

LearnBlockWeb le ofrece al usuario una interfaz gráfica similar a la de LearnBlock con la que poder empezar a programar de manera intuitiva con el lenguaje de programación visual basado en bloques. Igual que ocurre en LearnBlock, finalmente terminará programando en un lenguaje de programación profesional como es Python.

Está alojada en un servidor de internet (`https://learnblockweb.pythonanywhere.com/`) y todos los archivos están disponibles en GitHub (`https://github.com/RebeGM19/LearnBlockWeb`).

Capítulo 2

Objetivos

LearnBlock surge dentro del proyecto Emorobotic, que tiene como propósito la creación de una serie de herramientas que permitan la enseñanza y el aprendizaje de los procesos de programación en las etapas de educación primaria y secundaria, centrándose sobre todo en utilizar la programación como forma de desarrollar habilidades de gestión emocional. LearnBlock se ha diseñado para facilitar el aprendizaje de programación de manera progresiva, comenzando por un lenguaje de programación visual por bloques, pasando después por su representación textual, y que finalmente termina en un lenguaje de programación profesional. Esta aplicación, a su vez, posibilita establecer conexiones con múltiples robots, que se encargan de realizar las acciones especificadas en el código desarrollado.

LearnBlock destaca entre el resto de herramientas similares ya existentes por presentar una serie de propiedades de las que todas ellas carecen. Sus características son las siguientes:

- Posee una interfaz gráfica desde la que se le permite al usuario crear un programa utilizando elementos gráficos: bloques. Los bloques están relacionados con el control del programa, acciones que puede realizar el robot conectado, e información sensorial.
- Genera código Python a partir de un código intermedio que representa de forma textual el programa construido con los bloques.

- Posibilita no sólo modificar el código de bloques, sino también el lenguaje intermedio y el lenguaje Python, permitiendo así que el usuario elija en cuál de los tres niveles de abstracción desea programar.
- Soporta dos modelos de programación diferentes: programación secuencial y programación orientada a eventos.
- Admite la creación de bloques individuales cuya acción viene definida por una función Python desarrollada por el usuario desde la propia herramienta o desde herramientas externas.
- Permite que el mismo código, independientemente de si se trata del código visual, textual o Python, pueda ser ejecutado en diferentes robots.
- Actualmente es compatible con varios robots físicos, como EBO, Cozmo, Thymio y EV3, y con robots simulados bajo RCIS y V-REP.
- Se puede añadir compatibilidad con más robots sin necesidad de modificar el código base de LearnBlock ni de redefinir los bloques ya creados.

LearnBlock posee todas estas cualidades precisamente por ser una aplicación de escritorio que debe ser previamente instalada en el equipo. Como ya se ha especificado, esto es así por razones de conexión entre el proyecto y las plataformas robóticas, ya que en ocasiones es necesario disponer de algún otro tipo de software instalado que permita dicha conexión. No obstante, al ser una aplicación local, LearnBlock no se beneficia de las ventajas que otorga una herramienta de este tipo desarrollada como aplicación web.

LearnBlockWeb nace como una propuesta de los investigadores involucrados en el proyecto LearnBlock, cuyo objetivo principal es proporcionar una versión web del mismo que contenga la mayor parte de sus características y goce de los beneficios que supone tener una herramienta educativa online.

El principal de estos beneficios es el alcance. LearnBlock de por sí tiene un alcance bastante limitado, pues es necesario llevar a cabo un proceso de instalación

para poder ser utilizado, y una instalación es algo que no todas las personas están dispuestas a hacer, ya sea por falta de espacio en sus equipos, falta de tiempo, falta de conocimientos o, simplemente, falta de ganas. Por otro lado, la principal característica de LearnBlockWeb, al igual que el resto de herramientas online, es que no requiere de ese proceso de instalación para ser usada.

Otra de las particularidades de LearnBlockWeb es que puede ser utilizada sin importar el hardware del equipo ni el sistema operativo; lo único que se necesita es tener conexión a internet y un navegador instalado en el ordenador (Firefox, Chrome, Opera...). Esto hace que su accesibilidad aumente de manera considerable, pues cualquier tipo de usuario puede llegar a ella siempre que lo desee y puede utilizarla de manera sencilla simplemente con acceder al enlace de la web, sin necesidad de tener amplios conocimientos previos sobre software.

La aplicación web, al estar alojada en un servidor online, está constantemente disponible. Esto significa que los usuarios pueden acceder a ella en todo momento.

Todos estos beneficios que otorga LearnBlockWeb contribuyen a aumentar el alcance de LearnBlock. De esta manera, se consigue llegar a un mayor número de personas que no podrían acceder a él de otra forma que no fuera a través de su versión online.

Este es un modo, también, de lograr que el usuario tenga una primera toma de contacto con LearnBlock y conozca gran parte de lo que la aplicación de escritorio puede ofrecerle. LearnBlockWeb se ha desarrollado con la idea de mantener la mayor similitud posible con LearnBlock en lo referente, entre otros, al entorno visual. Esto facilita la transición por parte del usuario de una interfaz a otra y hace que no suponga un problema aprender a utilizar LearnBlock una vez ya se ha usado su versión online.

Otro de los aspectos en los que se busca esa similitud es en la arquitectura y en el funcionamiento. Tal como se verá en apartados posteriores, la arquitectura seguida es la misma en LearnBlock y en LearnBlockWeb en cuanto a las funcionalidades que ésta última abarca se refiere. Por otra parte, para desarrollar dichas funcionalidades, se ha hecho uso de los archivos y del código que las implementa dentro del proyecto

LearnBlock. Esto es así para facilitar el trabajo de los desarrolladores de ambos proyectos: si se realizan cambios en uno, las modificaciones necesarias para adaptar esos cambios al otro son mínimas.

Por otra parte, y debido a que la funcionalidad de LearnBlock de ejecución del código en distintos robots es algo que la web no podrá abordar por sí misma, se tiene como objetivo crear un paquete software que, una vez instalado en el equipo, permita al usuario ejecutar de forma local el código Python generado en la página sobre uno de los robots compatibles con LearnBlock.

Capítulo 3

Estado del Arte

El proyecto presentado, al ser una página web, presenta dos partes claramente diferenciadas [2]: La capa de presentación (Front-End) y la capa de acceso a datos (Back-End). A su vez, dentro del Front-End, en este caso se ha optado por hacer una diferenciación más: la estructura y el diseño general de la página, y lo referente al editor de programación por bloques. Para cada una de estas tres partes, se especificarán las herramientas y frameworks disponibles a día de hoy con las que se pueden llevar a cabo las funcionalidades correspondientes.

3.1. Front-End

El Front-End es todo lo relacionado con la interfaz de usuario, todo aquello que la persona que utiliza la aplicación puede ver y con lo que puede interactuar. Es el responsable de obtener los datos que introduzca el usuario y se encarga de convertirlos a un formato con el que, posteriormente, el Back-End pueda procesarlos de forma correcta. Dentro del ámbito web, el desarrollo Front-End consiste en la transformación de datos en una interfaz gráfica

3.1.1. Estructura y diseño web

Hay varias herramientas disponibles para desarrollar la interfaz de una página web. Las principales son las tres siguientes:

- **HTML** (HyperText Markup Language [3]). Es un lenguaje de marcas de hipertexto con el que se crea el esqueleto y la estructura principal de la página web mediante una serie de etiquetas. El código HTML otorga una primera visión de cómo será el sitio web. Con él, se pueden establecer vínculos que accedan a otras páginas web, y se pueden crear imágenes, tablas y elementos similares. Para añadir un elemento externo a la página (una imagen, un vídeo...), no se añade en el código de la página, sino que se hace una referencia textual a la ubicación en la que se encuentra ese elemento. Así, web solo contiene texto, y es tarea del navegador web interpretar ese texto y representar los elementos enlazados. La última versión de HTML es HTML5 y contiene nuevas maneras más sencillas y eficaces de manejar elementos de mayor complejidad.

- **CSS** (Cascading Style Sheets [5]). Son hojas de estilo que controlan el diseño y la apariencia visual del entorno web. Está creado para diferenciar entre el contenido de la página web y la presentación de la misma. Con esta separación se consigue mejorar la accesibilidad, la flexibilidad, el control de los elementos presentados y la facilidad de lectura e interpretación de los archivos que forman la web. Además, permite que varias páginas HTML compartan un mismo diseño. CSS se ha creado en varios niveles: cada uno de ellos se construye sobre el anterior y lo completa. La especificación actual es CSS3.

- **JavaScript** [6]. Es un lenguaje de programación interpretado, orientado a eventos y dinámico. Se usa para añadir elementos dinámicos a una página HTML estática. Se usa en el lado del cliente. Cuando se creó, solo se utilizaba con el fin de darle dinamismo a las páginas HTML. Sin embargo a día de hoy, junto a otras librerías y frameworks, es usado para enviar y recibir información del servidor. Se trata de un lenguaje imperativo y estructurado: se diseñó con una sintaxis

similar a la de C y, de hecho, es compatible con la mayoría de las estructuras de programación C.

Los frameworks que aparecerán a continuación están orientados a ser utilizados por alguna de estas tres herramientas.

En lo que a JavaScript se refiere, hay una gran cantidad de frameworks desarrollados para ser usados en el ámbito Front-End. Los más conocidos son los siguientes:

- **AngularJS** [7]: Es un framework de diseño y una plataforma de desarrollo para crear aplicaciones de una sola página usando HTML y TypeScript. Implementa la base y funcionalidades opcionales como un conjunto de librerías TypeScript. Es de código abierto y está mantenido por Google. Su objetivo es crear páginas que sigan el patrón MVC (Modelo-Vista-Controlador). Con este framework se pretende otorgar un mejor contenido dinámico con el uso de un data binding que permite la sincronización entre vistas y modelos. Hace que el desarrollador se centre menos en la manipulación de los documentos para así lograr que el testeo tenga mayor importancia.

- **React** [8]: Es una librería de código abierto que se utiliza para crear interfaces de usuario intuitivas. Se usa sobre todo para el desarrollo de la capa de visualización. React se encarga de actualizar y renderizar los componentes de la interfaz cuando la información presentada cambie. No obstante, también es efectiva para crear aplicaciones de una sola página. Se ha creado para ser adoptado de manera gradual.

- **Vue** [9]: Es un framework progresivo de código abierto, usado para construir interfaces de usuario. Ha sido desarrollado desde el principio para ser adoptable de forma incremental. El framework solo se centra en la capa de visualización, y es sencillo de integrar con otras librerías o proyectos. Es muy liviana y versátil, y supone una alternativa a Angular y React.

- **Meteor** [10]: La diferencia principal de este framework con los anteriores es que éste está orientado a un desarrollo Full-Stack de aplicaciones web y móviles. Facilita la creación de prototipos y está integrado con MongoDB.

- **Node** [11]: Al igual que Meteor, Node es un framework que se puede utilizar tanto en Front-End como en Back-End. Es muy rápido, altamente escalable y de código abierto. Todas las APIs del framework son asíncronas y se controlan por eventos. Los procesos se realizan en un único hilo.

- **Ember** [12]: Framework de código abierto que se centra en el desarrollo de aplicaciones ágiles y productivas, que consten de una única página y que sigan el patrón Modelo-Vista-VistaModelo (MVVM). El objetivo con esto es que puedan ser mantenidas a largo plazo.

Por otra parte, hay una librería para JavaScript bastante conocida cuyo objetivo es facilitar la forma de interactuar con los documentos HTML, así como la manipulación de eventos, estilos, efectos, animaciones y objetos. Este framework es **JQuery** [13], es de software libre y de código abierto. Actualmente tiene soporte para todas las versiones más actualizadas de los navegadores y su versión anterior, y es compatible con Android y con iOS. De entre todo lo que ofrece JQuery, una de las funcionalidades más útiles a la hora de integrar el Front-End con el Back-End de la página es la que proporciona la tecnología **AJAX** [14]. Con AJAX se puede conseguir intercambiar información con el servidor en segundo plano y de manera asíncrona, con lo que se consigue actualizar la página web sin necesidad de recargarla.

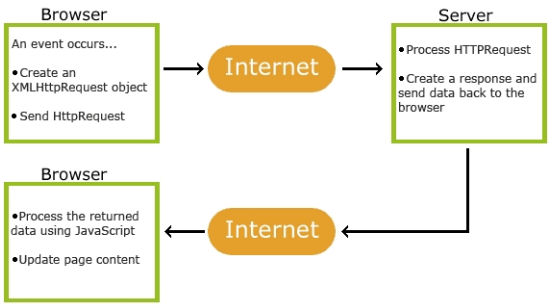


Figura 3.1: Funcionamiento AJAX. Fuente: W3Schools

En cuanto al propósito de conseguir un diseño *responsive*, es decir, que la página se adapte automáticamente al tamaño de pantalla que utilice el usuario, los frameworks considerados han sido los siguientes:

- **Bootstrap** [15] es, probablemente, el framework más conocido actualmente de entre todos los frameworks disponibles para el desarrollo Front-End. Se caracteriza por ser *responsive*. El framework está formado por una colección de hojas de estilo y funciones JavaScript auxiliares, incluyendo la librería **JQuery**. Ofrece una gran variedad de componentes predefinidos con los que se facilita la estructuración e implementación de la web, y es compatible con la mayoría de navegadores. Es de código abierto, disponible en GitHub, y se puede descargar o incluir directamente en el código HTML de la página.

- **Foundation** [16] es otro framework cuyo objetivo principal es proporcionar una interfaz de usuario *responsive*. Se adapta a cualquier dispositivo, medio y forma de accesibilidad. Incluye componentes HTML, CSS, plantillas, fragmentos de código y extensiones JavaScript. Foundation le otorga al desarrollador los patrones más comúnmente utilizados a la hora de crear y estructurar una página web *responsive*. Es un framework de código abierto y está disponible en Github.

- **HTML5 Boilerplate** [17] es otro framework que permite un diseño de aplicaciones *responsive*. Permite al desarrollador crear páginas web con HTML5 de una manera ágil, para lo cual ofrece una serie de plantillas. Ofrece código normalizado para todos los navegadores y optimizado para dispositivos móviles. Es de código abierto y está disponible en GitHub.

3.1.2. Programación por bloques

A día de hoy, es habitual el desarrollo de herramientas que contengan una interfaz con la que se le permita al usuario crear un programa con un lenguaje de programación por bloques. Existe un gran número de librerías que permiten hacer esto, según la plataforma en la que se esté desarrollando dicha herramienta y qué tipo de aplicación

se obtenga como resultado. No obstante, para aplicaciones creadas en un entorno web, solo hay una librería que se puede utilizar para insertar en la página una interfaz de programación por bloques.

Blockly [18] es un proyecto software de código abierto, desarrollado por completo en lenguaje JavaScript, compatible con la mayoría de navegadores web. Es una librería del lado del cliente, sin ningún tipo de dependencias del lado del servidor. Ofrece un editor de lenguaje de programación basado en bloques y un conjunto de generadores de código que traducen el programa construido con los bloques a una serie de lenguajes de programación. Blockly es altamente personalizable y extensible: como es del lado del cliente, el desarrollador tiene acceso a todo el código y puede modificarlo a gusto para cambiar la interfaz, los bloques presentados y todo su funcionamiento.

La interfaz gráfica por defecto de Blockly consta, por una parte, de un listado que contiene todos los bloques disponibles y desde donde el usuario puede seleccionar dichos bloques. Por otra parte, dispone de un editor en el que se pueden arrastrar, colocar y recolocar los bloques que se desee.

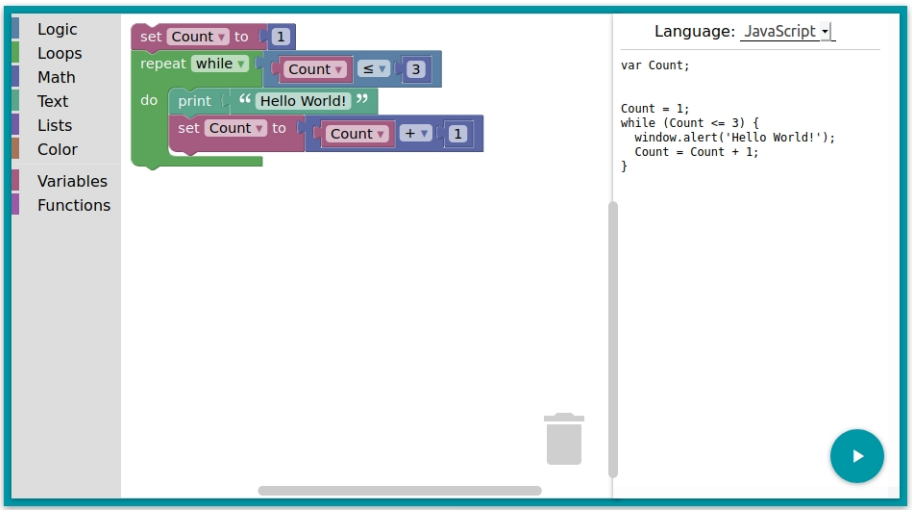


Figura 3.2: Interfaz Blockly. Fuente: Blockly - Google Developers

Blockly ha sido utilizado en numerosos proyectos educativos bastante conocidos. Algunos de ellos son App Inventor [19], aplicaciones de Code Studio [20], Micro:Bit [22] y demás herramientas de Microsoft MakeCode [21], OzoBlockly [23], y CodeBug [24].

3.2. Back-End

Por otra parte, para el desarrollo de la capa de datos de la página web se ha optado por utilizar el lenguaje de programación Python [25], que es a su vez usado para implementar las funcionalidades del proyecto LearnBlock. En cuanto al desarrollo de aplicaciones web con Python, también existen varios frameworks bastante potentes que permiten y facilitan el proceso. Son los siguientes:

- **Django** [26]: Se trata de un framework web de alto nivel y es, probablemente, el mayor de todos los frameworks existentes basados en Python. Se centra en ofrecer una manera de desarrollar páginas complejas y orientadas a contenido de forma limpia, rápida y con un diseño pragmático. Permite una alta seguridad, escalabilidad, flexibilidad y facilidad de mantenimiento. Utiliza ORM (Object Relational Mapper) y es compatible con PostgreSQL, MySQL, SQLite y Oracle. Siguiendo el principio DRY (Don't Repeat Yourself), Django le proporciona al desarrollador todo lo que necesita para crear su sitio web. Es de código abierto.

- **Flask** [27]: Es un microframework que resulta especialmente conveniente para desarrollar de forma ágil y rápida una aplicación o web sencilla que no necesite demasiadas extensiones. Incluye un servidor de desarrollo, lo cual hace que no se necesite una infraestructura con un servidor web para probar el funcionamiento de la página. Posee un depurador de errores, tiene la posibilidad de integrar pruebas unitarias, y soporta de forma nativa la utilización de cookies. Flask se puede aprender a usar muy rápido.

- **Pyramid** [28]: Este framework tiene como objetivo ser rápido y fiable. Se puede escribir una aplicación entera con esta librería que ocupe un único archivo, lo cual hace que sea fácil de entender. A su vez, su despliegue es sencillo. En cierta manera, permite al desarrollador hacer todo lo que consiguen los microframeworks de formas bastante similares. Con Pyramid es posible crear aplicaciones simples sin necesidad de tener conocimientos avanzados,

pero también existe la posibilidad de construir aplicaciones considerablemente grandes.

- **Web2Py** [29]: Con este framework resulta sencillo empezar a programar webs con Python, es fácil de usar y de desplegar. Es una librería orientada a crear aplicaciones web con bases de datos que sean escalables, rápidas y seguras, y busca que el desarrollo de las mismas resulte ágil. Permite generar formularios desde las tablas de las bases de datos, y contiene un gran número de validadores. Es una librería de código abierto y es sumamente completa. No necesita de ningún tipo de instalación o configuración y trae consigo su propio IDE de desarrollo web.

- **CherryPy** [30]: Este es un framework minimalista que hace que el desarrollo de una aplicación web sea similar al de cualquier otro programa orientado a objetos. Se ha creado para ser extensible y lograr una construcción rápida del sitio web. Es completamente personalizable, cubre el manejo de sesiones, autentificación, caché, codificación y contenido estático, y tiene un sistema de configuración potente. Con él es simple ejecutar varios servidores HTTP a la vez.

- **Bottle** [31]: Se trata de un microframework diseñado para ser ligero, fácil y rápido. Fue creado en un principio para construir APIs sencillas de una manera simple y ágil. Todo Bottle está implementado en un único archivo y no tiene ninguna dependencia aparte de la librería estándar de Python. Como características principales destacan el uso del enrutamiento y de plantillas, estar incorporado en un servidor HTTP, y facilita el acceso a datos, cookies, headers y otros metadatos HTTP.

Capítulo 4

Metodología

4.1. Lenguajes y herramientas utilizados

Hay numerosos lenguajes de programación que se pueden utilizar para desarrollar una página web. Para la capa de presentación, se necesita utilizar siempre el lenguaje HTML, pues es el que le da forma a la página. Como lenguajes adicionales y altamente recomendados para proporcionar una interfaz apropiada, conviene utilizar CSS en lo relativo al diseño y JavaScript para contenido dinámico.

De entre las diferentes versiones de HTML que hay, la más reciente y la más completa es HTML5. Contiene nuevos atributos, elementos y comportamientos, y permite la integración de librerías que resulta conveniente integrar para el desarrollo de la web. A su vez, HTML5 se está convirtiendo en un estándar entre las páginas. Estas son las razones por las que se ha optado por usar dicha versión en LearnBlockWeb.

Por otra parte, para crear la capa de manejo de datos, las opciones de lenguajes son bastante más diversas. El Back-End de una web se puede implementar prácticamente en cualquier lenguaje, y cada uno de ellos ofrece una serie de ventajas y desventajas a la hora de hacerlo. En este caso, por razones de simpleza y, sobre todo, de similitud con referencia al proyecto LearnBlock, el funcionamiento de este proyecto se ha desarrollado en Python.

Python es un lenguaje interpretado, multiplataforma y multiparadigma, que se

caracteriza por ser sencillo, flexible, ordenado, legible, versátil y fácil de aprender, y se puede utilizar para desarrollar cualquier tipo de programa. Proporciona una amplia variedad de herramientas y librerías con las que se puede desarrollar prácticamente cualquier tipo de proyecto. Éste es uno de los lenguajes más populares de los últimos años, y ha sido la opción elegida para desarrollar el proyecto LearnBlock.

En el apartado 'Estado del Arte' se han listado una serie de herramientas y librerías que se pueden utilizar para el desarrollo Full-Stack de una página web. A continuación, siguiendo el orden de dicho apartado, se expondrán las razones por las que se han elegido algunas de ellas a la hora de implementar LearnBlockWeb.

En primer lugar, de entre los seis los frameworks considerados para el desarrollo Front-End de la página con JavaScript, se ha optado por no utilizar ninguno de ellos. La razón es que la gran mayoría del código JavaScript utilizado es el correspondiente a la implementación de la librería Blockly, que ya tiene su propia estructura y organización para su correcto funcionamiento. Por lo tanto, no se ha planteado el uso de un framework ya que se aplicaría a una mínima parte del código.

No obstante, sí ha sido necesario considerar la integración de JQuery en el proyecto. Esto es así porque una de las funcionalidades principales, la de generación de código, necesita hacerse tomando datos de la web (los bloques) y mostrar otros datos distintos obtenidos a partir de ellos (el código generado). Para lograr esto de una forma limpia y ágil, se debe hacer de manera asíncrona y sin recargar la página: esto último resultaría contraproducente, porque se perdería la información que ha introducido el usuario. Para lograr esa lectura de datos del servidor y ser capaz de enviar posteriormente otra información de manera asíncrona, es necesario hacer uso de la tecnología AJAX, que viene integrada dentro de la librería JQuery.

En cuanto a los frameworks para conseguir un sitio web *responsive*, la elección ha sido Bootstrap. De los tres presentados, es el más extendido y, probablemente, el más fácil de utilizar. Ofrece una gran variedad de componentes muy sencillos de entender y de añadir al sitio web. Además, la documentación que tiene es la más amplia, y abarca todo lo que la librería ofrece. Por otra parte, integrar Bootstrap es muy simple: no hay

necesidad de descargar nada, lo único que hay que hacer es añadir en el código HTML cuatro sentencias, una que incluya su hoja de estilo y tres que incluyan los scripts necesarios. Hay que tener en cuenta que para utilizar Bootstrap es conveniente usar una plantilla de HTML5; si no se usa, es probable que se vean estilos o funcionalidades incompletas dentro de la web.

Para obtener el entorno de programación por bloques, la opción más sencilla es añadir la librería Blockly al proyecto. La alternativa sería crear un entorno similar desde cero, ya que no hay otras librerías orientadas a web que otorguen esta funcionalidad. Blockly es un proyecto bastante completo y, aún así, se puede modificar y extender como y cuanto se desee. Al ser código del lado del cliente, para incluir la librería Blockly en una web propia lo que se debe hacer añadir al proyecto los archivos *.js* que implementan su estructura, diseño y funcionalidad. Estos ficheros se deben importar dentro del documento *.html* en el que se desee plasmar el entorno de desarrollo como si fueran cualquier otro archivo JavaScript, con la etiqueta *<script>*.

En lo referente al Back-End de la web, el framework utilizado ha sido Flask. Al ser un microframework, ofrece una serie de funcionalidades ideales para el desarrollo de páginas web que no sean excesivamente complicadas. Es la mejor opción para crear de forma rápida y sencilla el funcionamiento de una aplicación, y resulta muy fácil de comprender para un desarrollador que no tenga experiencia previa con el lenguaje Python. Para instalarlo, no hace falta más que crear un entorno desde la terminal, activarlo y añadir la sentencia de instalación. Para añadir Flask al proyecto, con hacer un *import* en el fichero que corresponda es suficiente.

4.2. Desarrollo. Metodología

A la hora de crear una página web, si la implementación es Full-Stack, el desarrollador puede elegir si comenzar por la parte correspondiente a la capa de presentación o a la parte de la capa de datos. A continuación se explicará el orden que se seguirá en este caso para realizar el proyecto.

El primer paso ha sido plantearse qué es lo que se quiere conseguir con la

aplicación: qué debe hacer, con qué puede interactuar el usuario y qué debe mostrar la web. Tras esto, se ha investigado y se ha decidido cuáles son las herramientas que se deben utilizar para cada una de las partes de la aplicación.

Una vez hecho esto, se empezará creando el esqueleto de la página web en HTML. La idea es hacer una página con dos partes diferenciadas. La primera parte es un menú superior donde se incluyen, entre otras, las opciones de elegir el robot que ejecutará el código y el idioma de la página. La segunda parte es toda la zona del editor de código. Con esta estructura, ya se podrá empezar a trabajar de manera cómoda.

El siguiente paso será añadir el editor del lenguaje de bloques. Esto se consigue integrando la librería Blockly al proyecto. La interfaz por defecto proporcionada por Blockly es bastante distinta a la de LearnBlock tanto en estructuras y funcionalidades como en diseño, por lo que, previendo un trabajo que se tendría que hacer más adelante y que, probablemente, en etapas posteriores resulte más complicado llevar a cabo, en este paso se modificará esa interfaz. Se eliminarán funcionalidades, cambiarán opciones y modificarán las categorías y los bloques disponibles.

En cuanto se tenga el editor de bloques añadido y completamente funcional, se pasará a crear la integración de la parte del Front-End con la parte del Back-End. De esta manera, ya se podrá lanzar la página de una forma dinámica y será posible trabajar con los datos obtenidos de la misma.

Cuando se establezca dicha conexión, el objetivo será procesar los datos obtenidos del editor de bloques para generar el código Block-Text equivalente. A su vez, una vez se tiene el código Block-Text, procesarlo para conseguir el código Python. Los módulos de traducciones a ambos lenguajes están escritos en Python y se han integrado desde el proyecto LearnBlock. Los dos códigos generados serán enviados de vuelta al Front-End de la web para ser mostrados en sus interfaces correspondientes.

Hecho esto, se generará un intérprete que traduzca el código Json con el que se procesan y crean los bloques en LearnBlock a una sintaxis Json equivalente que sea capaz de entender la librería Blockly. De esta manera, los bloques se cargarán de manera dinámica, desde archivos independientes y separados de la funcionalidad

general de la aplicación, y facilitará la incorporación de nuevos bloques al no ser necesario traducir sus atributos de una sintaxis a otra de forma manual.

En cuanto se tenga terminada toda la funcionalidad, se realizarán pruebas de ejecución generales en las que se probará si los bloques obtenidos desde la interfaz son correctos, si el código generado a partir de ellos equivale al programa que representan, y si los bloques son cargados de forma apropiada. En este momento habrá que buscar fallos y depurar los detalles del Back-End.

Una vez todo funcione como debe, será el momento de comenzar con el diseño visual de la aplicación. Si es necesario, se podrá reorganizar mínimamente la estructura HTML, pero lo principal será centrarse en la visualización de la página y de la interfaz. Cambiar colores, tamaños, disposiciones, fuentes de letra... Lo necesario para que a la persona que la vaya a utilizar le resulte lo más agradable y cómodo posible.

Por último, habrá que realizar pruebas de la página en su conjunto. Buscar detalles que puedan ser confusos para el usuario, modificarlos para hacerlos más claros y accesibles, añadir funciones que faciliten el uso general de la página... En definitiva, recopilar ideas, propias y ajenas, y aplicarlas según proceda.

Capítulo 5

Implementación y desarrollo

5.1. Descripción de la página web

La interfaz gráfica principal de la página desarrollada permite que el usuario navegue entre los tres entornos de programación diferentes que LearnBlock ofrece, cada uno de ellos asociado con los tres lenguajes en los que se puede programar: Bloques, Block-Text y Python. Esta navegación se lleva a cabo con las tres pestañas disponibles en la parte superior de la interfaz. Por defecto, cada vez que se accede a la página web, el entorno activo es el correspondiente al lenguaje de bloques.

Desde la pestaña `LearnBlock` se accede a la interfaz de programación por bloques (Figura 5.1). Ésta presenta dos partes diferenciadas. La parte de la derecha es el editor, donde se pueden arrastrar los bloques y conectarlos entre sí para formar un programa. La parte de la izquierda es una lista que contiene todos los bloques disponibles con los que trabajar.

Los bloques están divididos en distintas categorías, agrupados según la estructura o funcionalidad que representen. Son las siguientes:

- Control: Estructuras de control del programa. Bucles, condicionales y funciones de control del tiempo.
- Operadores: Aritméticos: suma, resta, multiplicación, división, igualdad. Lógicos: comparaciones, and, or, not.

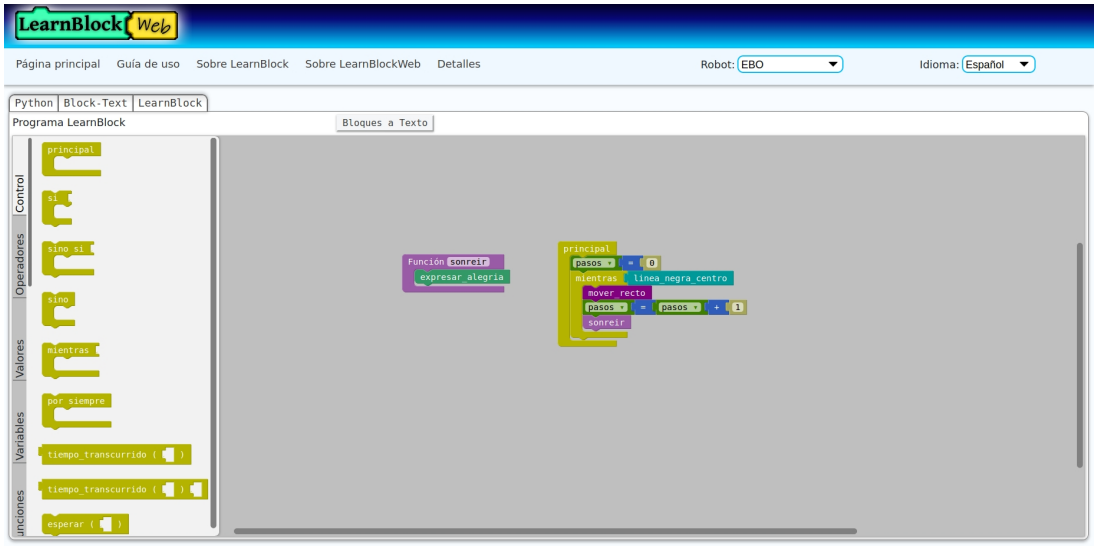


Figura 5.1: Interfaz LearnBlock

- Valores: Numéricos y textuales.
- Variables: Variables creadas por el usuario. No tienen tipo definido, se le puede asignar cualquier valor a cualquier variable.
- Funciones: Funciones creadas por el usuario. Son funciones sencillas, no tienen parámetros y no devuelven ningún valor.
- Emociones: Emociones que puede sentir y expresar el robot.
- Speaker: Frases que puede decir el robot.
- Base: Movimientos que el robot es capaz de realizar sobre una superficie.
- Motor: Movimientos que el robot es capaz de hacer con su propio cuerpo.
- Cámara: Comprobaciones de lo que el robot detecta sobre el entorno que le rodea a través de su cámara.
- Distancias: Comprobaciones de las distancias a las que se encuentra el robot de posibles obstáculos que le rodeen.
- Suelo: Comprobaciones de lo que el robot se puede encontrar en la superficie en la que está.

La forma que tienen los bloques varía según la funcionalidad que representen. Algunos permiten introducir nuevos bloques dentro de ellos (Figura 5.2.1), otros aceptan bloques debajo (Figura 5.2.2), y otros pueden ser conectados a la derecha de un bloque ya existente (Figura 5.2.3).

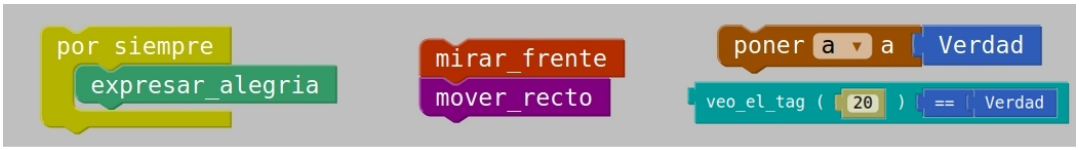


Figura 5.2: Tipos de bloques

Si los bloques se conectan correctamente y forman un programa cuya sintaxis tenga una estructura válida, se podrá realizar la traducción de ese conjunto de bloques a lenguaje Block-Text y a lenguaje Python. Si, por el contrario, hay algún fallo en la estructura creada, el código equivalente en ambos lenguajes no será generado. Para generar los dos códigos a partir de los bloques hay que hacer click en el botón `Bloques a Texto` que hay en la parte superior de la interfaz (Figura 5.1).

Si se selecciona la pestaña `Block-Text`, se puede ver una interfaz diferente en la que aparece el código equivalente a la estructura de bloques previamente creada, en el caso de que ésta sea correcta y se haya podido llevar a cabo la traducción (Figura 5.3). Desde esta interfaz, el usuario puede editar el código generado si lo desea, o incluso comenzar a escribir él mismo un programa desde el principio sin necesidad de pasar por la construcción de bloques, en caso de que haya adquirido los conocimientos suficientes. Con el código Block-Text escrito o modificado se puede, a su vez, generar el código Python correspondiente.

Block-Text es un lenguaje creado por los propios desarrolladores del proyecto LearnBlock. Se trata de un lenguaje intermedio que representa de forma textual, con una estructura similar a la que tiene Python, lo construido con el lenguaje visual de bloques.

Por último, si se selecciona la pestaña `Python`, se le mostrará al usuario una interfaz similar a la anterior en la que se presenta el código Python generado a partir del

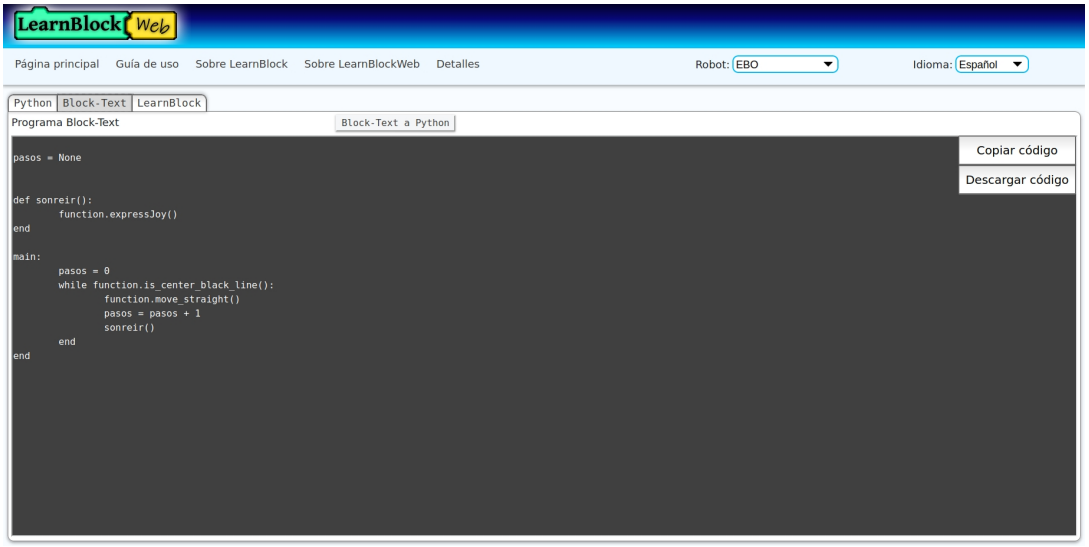


Figura 5.3: Interfaz Block-Text

lenguaje textual (Figura 5.4). De la misma manera, existe la posibilidad de crear o modificar el código Python desde la interfaz.

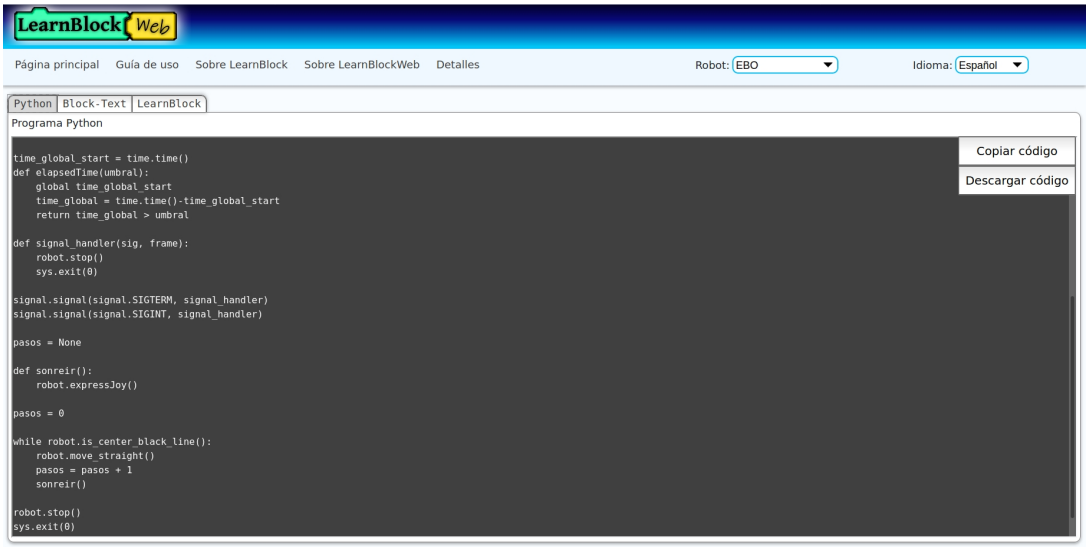


Figura 5.4: Interfaz Python

El código Python presentado por el programa, además de contener la traducción equivalente a lo generado de forma previa, incluye una serie de comandos necesarios para la creación de una instancia de una clase `robot` que se encarga de establecer comunicación con la plataforma robótica seleccionada.

El proyecto LearnBlock es compatible con diferentes robots físicos y simulados.

Éstos son presentados en LearnBlockWeb en un menú desplegable desde el cual el usuario puede elegir el robot que desea que ejecute el código Python generado. Los robots físicos disponibles son EBO, EV3, Cozmo y Thymio. Los robots simulados son EBO_sim, EBO_remote_sim y EV3_sim.

LearnBlock permite crear programas basados en dos estructuras de programación distintas: secuencial y orientada a eventos. LearnBlockWeb cubre la estructura de programación secuencial, en la que el usuario debe incluir todos los bloques que quiere que tenga su programa dentro de un bloque principal (main) o en bloques de funciones que a su vez sean añadidas en el bloque principal.

Al igual que LearnBlock, LearnBlockWeb ofrece la posibilidad de elegir el idioma con el que se presenta la página. Todas las palabras, ya sean las del texto de la web o las de los nombres de los bloques, varían entre el inglés y el español según la elección del usuario (Figura 5.5).

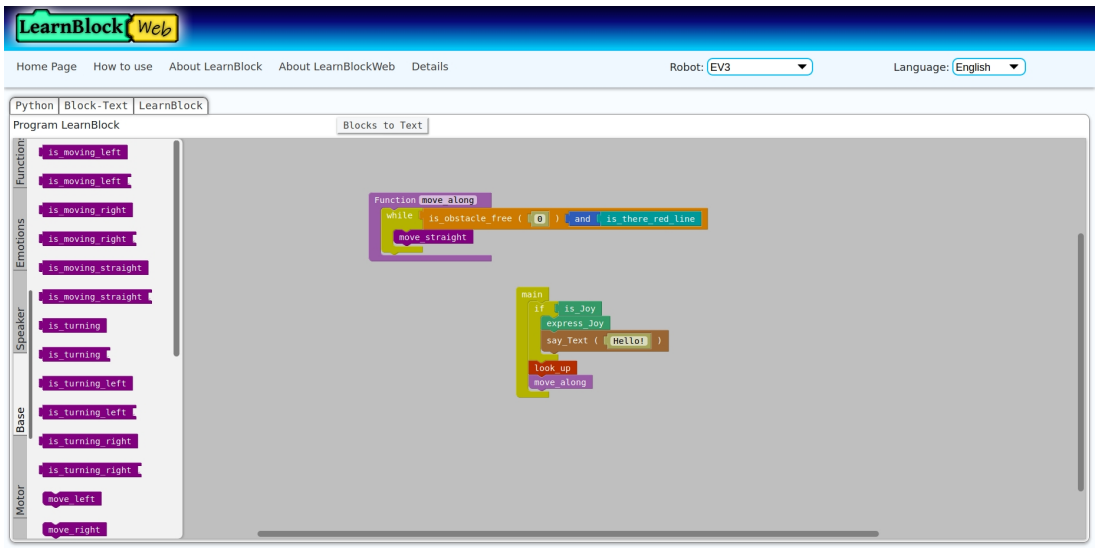


Figura 5.5: LearnBlockWeb en inglés

Por otra parte, y teniendo en cuenta la creciente exigencia de atender a las necesidades de los distintos tipos de usuarios y sus circunstancias, LearnBlockWeb se ha desarrollado de tal manera que cumple los criterios WCAG de conformidad y accesibilidad en entornos web. Esto se explicará mas en profundidad en el apartado 'Accesibilidad Web'.

5.2. Relación con LearnBlock

LearnBlockWeb se ha diseñado con el objetivo de crear una versión online de LearnBlock que implemente gran parte de sus propiedades. Por lo tanto, no es de extrañar que la mayoría de las funcionalidades de la web estén basadas o se resuelvan de la misma manera que en LearnBlock. Esto hace que los dos proyectos estén estrechamente unidos, lo cual simplifica la labor de los desarrolladores de ambos y facilita que los usuarios se adapten a los cambios entre usar una plataforma u otra.

Como ya se ha explicado en este documento, las características de LearnBlock se pueden dividir en tres grupos diferenciados: Programación, generación de código y ejecución de código. Los detalles de la arquitectura LearnBlock se especifican en la Figura 5.6.

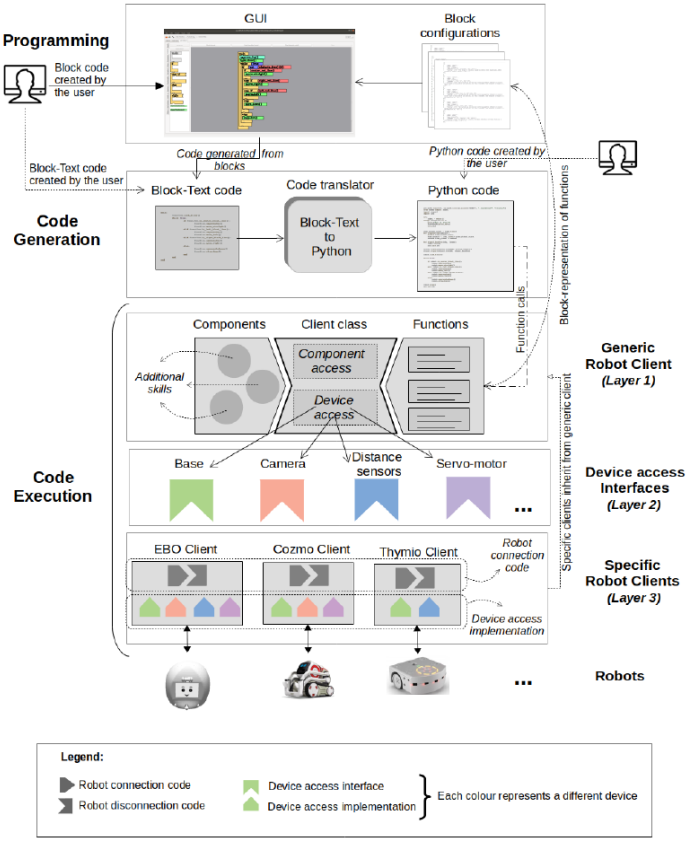


Figura 5.6: Arquitectura LearnBlock. Fuente: ”LearnBlock: A Robot-Agnostic Educational Programming Tool”

LearnBlockWeb comparte la misma arquitectura de LearnBlock en lo referente a la programación y la generación de código. Además de esto, LearnBlockWeb hace uso del mismo código que utiliza LearnBlock para implementar la funcionalidad de traducción a los distintos lenguajes. De esta forma, si se hace algún cambio en el proyecto LearnBlock, se pueden añadir los mismos cambios a su versión web sin necesidad de modificar nada más para que funcione correctamente.

Teniendo en mente la idea de hacer más cómoda la interacción en ambas plataformas por parte del usuario, LearnBlockWeb se ha diseñado para que tenga una estructura visual similar a la de LearnBlock. La distribución del editor, de las listas de bloques y categorías, de las pestañas de los tres lenguajes, del botón de ejecución y del selector de idiomas es semejante en las dos interfaces (Figura 5.7).

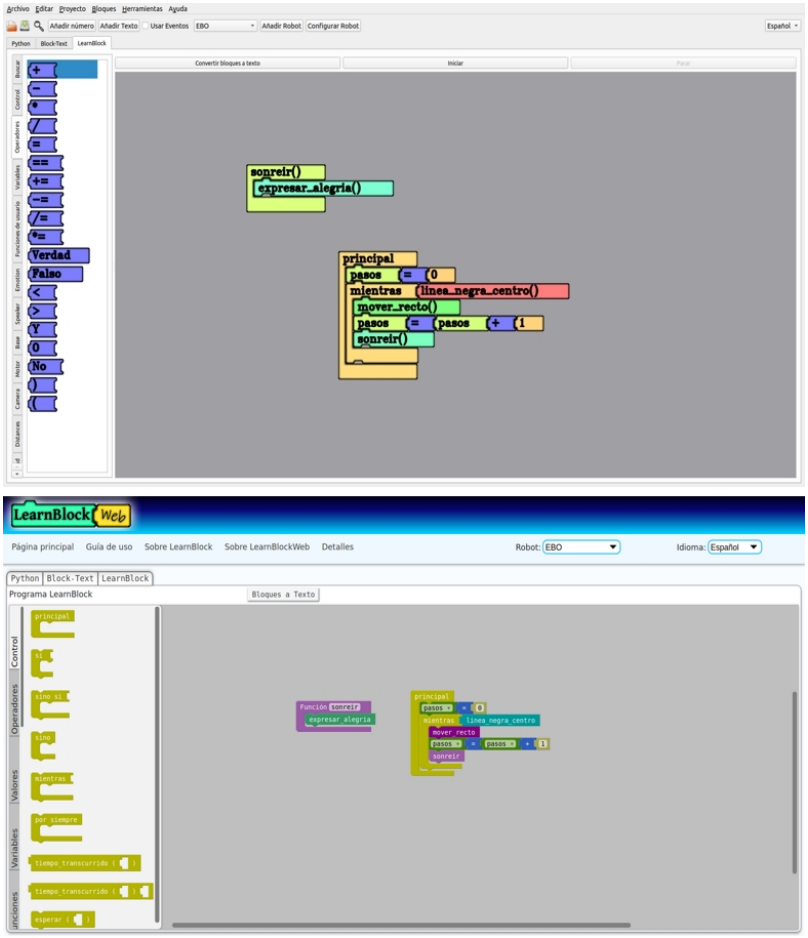


Figura 5.7: Comparación interfaces

En LearnBlock, los bloques disponibles se obtienen a partir de una estructura Json que los define. Con eso se especifica el tipo de bloque que es, a qué categoría pertenece, su nombre y su forma, entre otras cosas. LearnBlockWeb toma esa definición y la adapta para incluir esos mismos bloques en su interfaz. Con esto se consigue que un mismo archivo Json sirva para definir los bloques tanto en LearnBlock como en LearnBlockWeb.

```
"type": "motor",
"category": "Base",
"name": "turn",
"variables": [{"type":"float", "name":"angle", "default":"0"}],
"shape": ["blockVertical"],
"languages": {"ES": "girar", "EN": "turn"},
"tooltip": {"ES": "Gira el robot con una velocidad angular",
           "EN": ""}
```

Listing 5.1: Ejemplo de definición de un bloque en LearnBlock

Ya que LearnBlockWeb no abarca la funcionalidad de LearnBlock de la ejecución del código, se ha creado un paquete que, al instalarlo en el equipo, permite probar el código resultante. Éste contiene los archivos que implementan las acciones de las funciones predefinidas, las funcionalidades de los distintos robots, y las interfaces disponibles. Importante destacar que el paquete no tiene dependencias más allá de las específicas del robot que se vaya a utilizar, por lo que no es necesario añadir ninguna librería o programa al equipo para que funcione correctamente. Este paquete debe ser instalado en un sistema Ubuntu, y se puede descargar desde la pestaña 'Guía de uso' de la página web, donde también se listan los pasos a seguir para su instalación.

5.3. Desarrollo Full-Stack: Front-End

5.3.1. Estructura y diseño de la web

Como se busca que LearnBlockWeb tenga el mayor parecido estructural posible con LearnBlock, se ha optado por desarrollar una web con una única página principal que contenga todas las interfaces con las que el usuario puede interactuar, y que se navegue entre ellas de la misma manera que en la aplicación de escritorio. Como añadido, tiene una serie de pestañas adicionales que muestran una información escueta sobre ambos proyectos y algunos detalles de la web.

Entre los documentos del proyecto LearnBlockWeb se encuentra un único archivo con extensión *.html* dentro del directorio *templates*. Ese archivo sigue las directrices y la sintaxis del lenguaje de marcado HTML5. Además, para complementar el uso de esta estructura, se ha optado por la implementación de la librería Bootstrap.

LearnBlockWeb está dividido en dos partes diferenciadas: la zona superior y la zona central.

- Zona superior: Logo y menú superior. El menú contiene un selector de idioma, el selector del robot que ejecutará el código, y botones de navegación entre la página principal y las páginas de información. Esta navegación se hace a través de los componentes `nav` de Bootstrap. Cada vez que se selecciona un idioma, la página se recarga y todo el texto que hay en ella aparece escrito en el idioma elegido.

- Zona central: Interfaz LearnBlock y párrafos de información. La selección entre la interfaz visual (LearnBlock) y las textuales (Block-Text y Python) también se ha implementado con los componentes `nav` (Figura 5.8). Dentro de la interfaz LearnBlock se han añadido unos mensajes que indican si la traducción del código ha sido correcta o no (Figura 5.9), y unos botones que ejecutan funciones JavaScript al hacer click sobre ellos: dos de esos botones llaman a los procedimientos de generación de código, y los otros dos copian el código

generado (Block-Text o Python, según la interfaz en la que el botón se encuentre) al portapapeles.

```
<ul id="listButtons" class="nav nav-pills">
    <li><button type="button" href="#tab-python" data-toggle="tab" id="bPython">Python</button></li>
    <li><button type="button" href="#tab-blocktext" data-toggle="tab" id="bBT">Block-Text</button></li>
    <li><button type="button" href="#tab-learnblock" data-toggle="tab" id="bLB">LearnBlock</button></li>
</ul>

<div class="tab-content" style="margin-bottom:10px">
    <div class="tab-pane" id="tab-python"> <!-- Content for tab Python -->
        <ul class="nav">
            <li class="insideList"><label for="resultpython" id="programPY"></label></li>
        </ul>
        <div class="textarea-container">
            <textarea id="resultpython"></textarea>
            <button onclick="copyPython()" id="bCopyPY"></button>
        </div>
    </div>
    <div class="tab-pane" id="tab-blocktext"> <!-- Content for tab Block-Text -->
        <ul class="nav">
            <li class="insideList"><label for="resultblocktext" id="programBT"></label></li>
            <li class="insideList" id="sepButtons"></li>
            <li class="insideList"><button class="execButton" type="button" onclick="executeBT2Py()" id="e
            <li class="insideList" id="sepButtons"></li>
            <li class="insideList" id="resultCode2"></li>
        </ul>
        <div class="textarea-container">
            <textarea id="resultblocktext"></textarea>
            <button onclick="copyBT()" id="bCopyBT"></button>
        </div>
    </div>

    <div class="tab-pane active" id="tab-learnblock"> <!-- Content for tab LearnBlock -->
        <ul class="nav">
            <li class="insideList" id="programLB"></li>
```

Figura 5.8: Código HTML de la navegación entre pestañas

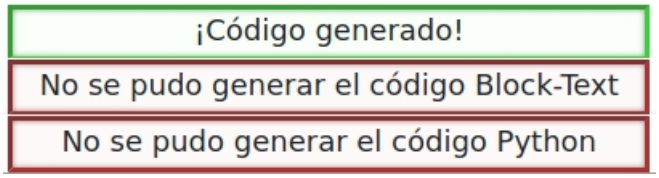


Figura 5.9: Resultados de la generación de código

La interfaz correspondiente al editor visual de bloques viene dada por la librería Blockly de la que se hablará más adelante en este documento. En lo que a la estructura HTML se refiere, esa librería requiere que se especifique con etiquetas *div* el lugar en el que se quiere colocar dicha interfaz, y es también necesario listar con sintaxis HTML todas las categorías disponibles y los bloques que hay dentro de ellas (Figura 5.10).

La zona central de la página, concretamente la interfaz principal, ha sido diseñada para parecerse lo máximo posible a la de la versión de escritorio. La disposición de cada una de sus partes, al igual que sus colores, son similares. Las interfaces

```
<div id="learnblockDiv"></div>
<div id="toolbox" style="display: none">

    <category name="%{BKY_CONTROL}" expanded="true">
        {% for block in allBlocks %}
        {% if block[1] == 'control' or block[1] == 'fcontrol' %}
        <block type="{{ block[0] }}"></block>
        {% endif %}
        {% endfor %}
    </category>

    <category name="%{BKY_OPERATORS}">
        {% for block in allBlocks %}
        {% if block[1] == 'operator' %}
        <block type="{{ block[0] }}"></block>
        {% endif %}
        {% endfor %}
    </category>

    <category name="%{BKY_VALUES}">
        <block type="val_num1"></block>
        <block type="val_num2"></block>
        <block type="val_text1"></block>
        <block type="val_text2"></block>
    </category>
```

Figura 5.10: Código HTML de las categorías y bloques

LearnBlock, Block-Text y Python ocupan la mayor parte de la página: prácticamente toda su anchura, y más de la mitad de su altura.

La interfaz LearnBlock (Figura 5.1) está dividida en dos partes distintas: A la izquierda aparece un panel desplegable que contiene todos los bloques pertenecientes a la categoría elegida, la cual a su vez se selecciona de una lista ubicada a la izquierda de dicho panel. A la derecha se encuentra el editor, donde se pueden arrastrar los bloques desde el panel izquierdo y encajarlos entre sí para formar un programa. Ambas partes se presentan en tonos grises, la derecha con un color más oscuro que la izquierda, al igual que se ha diseñado en el proyecto LearnBlock (Figura 5.7). La forma y el color de los bloques, por el contrario, no son tan parecidos. En LearnBlockWeb, el color de los bloques viene dado por la categoría a la que pertenecen, mientras que en LearnBlock se sigue un patrón distinto. En cuanto a la forma, la librería utilizada por LearnBlockWeb permite cambiar el diseño de los bloques y de sus conexiones, así como la fuente y el color de letra que aparece en ellos; sin embargo, aunque se ha decidido cambiar la forma de la conexión izquierda/derecha (Figura 5.2.3), los diseños de conexión encima/debajo/dentro (Figuras 5.2.1 y 5.2.2), así como la forma del bloque completo, se mantienen inmutables, pues se considera que de esa manera resultan más

agradables visualmente.

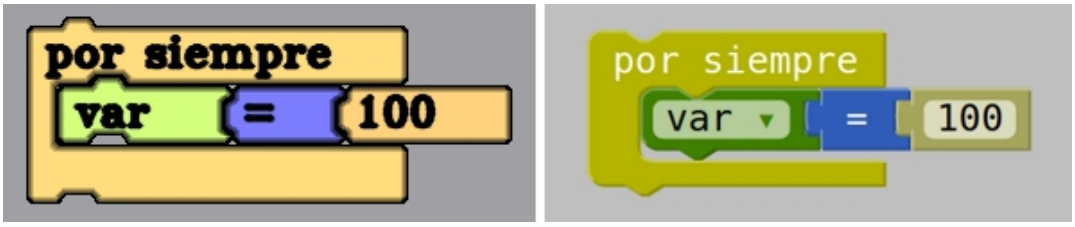


Figura 5.11: Bloques LearnBlock (izq) - Bloques LearnBlockWeb (der)

Las interfaces Block-Text (Figura 5.3) y Python (Figura 5.4) son más simples, pero también tienen una gran similitud con las equivalentes en la versión de escritorio. Se trata de un par de áreas de texto con fondo gris oscuro en el que aparece el código generado en color blanco. En este caso, hay dos diferencias a destacar entre ambas versiones. La primera es que se han diseñado dos botones en cada interfaz: uno de ellos copia al portapapeles el código presentado en la misma, y el otro permite guardar en el equipo un fichero *code.bt* o *code.py* que incluya ese mismo código. La segunda es el color del texto mostrado: mientras que en LearnBlock las palabras reservadas y los valores textuales y numéricos tienen un color diferente, en LearnBlockWeb no se lleva a cabo esa diferenciación y todo el código aparece en color blanco (Figura 5.12).

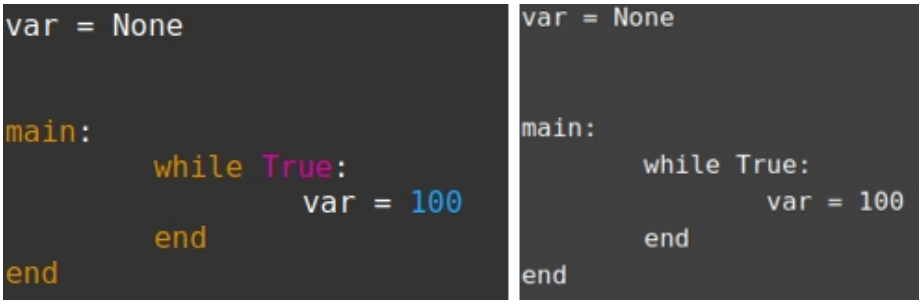


Figura 5.12: Código en LearnBlock (izq) - Código en LearnBlockWeb (der)

En cuanto al resto de la página, se ha optado por utilizar tonos azules que le resulten agradables y no demasiado llamativos al usuario, y una fuente de letra grande, clara y sencilla.

Destacar que cada componente de la página que pueda ser seleccionado, ya sea un botón, un enlace, o cualquier otro elemento, cambia sus propiedades de diseño cuando

es resaltado por el puntero del ratón al pasar por encima o por los tabuladores del teclado.

Como ya se ha especificado, se ha implementado una pequeña funcionalidad en la web mediante la cual se le permite al usuario copiar al portapapeles el código Block-Text o Python que se haya escrito en la interfaz correspondiente. Se trata de una función JavaScript que selecciona el contenido textual del elemento HTML que contiene ese código y ejecuta el comando `copy` de la misma manera a como se haría si se hace click derecho sobre el texto resaltado y se selecciona la opción `copiar`. Esta funcionalidad va asociada a los botones `Copiar código`, y se ejecuta cuando se hace click en ellos (Figura 5.13).

```
    <div class="textarea-container">
        <textarea id="resultpython"></textarea>
        <button onclick="copyPython()" id="bCopyPY"></button>
    </div>
    <div class="textarea-container">
        <textarea id="resultblocktext"></textarea>
        <button onclick="copyBT()" id="bCopyBT"></button>
    </div>
```

```
//Copies the Block-Text code
function copyBT() {
    var copyText = document.getElementById("resultblocktext");
    copyText.select();
    document.execCommand("copy");
}

//Copies the Python code
function copyPython() {
    var copyText = document.getElementById("resultpython");
    copyText.select();
    document.execCommand("copy");
}
```

Figura 5.13: Código JavaScript: Copia de código

Además de éstos, también hay otro botón en cada interfaz que permite descargar un fichero y guardarlo en el equipo. Estos ficheros contienen el código Block-Text o Python presentados en las interfaces. Para el código Block-Text se crea un archivo con extensión *.bt*, y para el código Python el archivo tiene extensión *.py*. Dicha funcionalidad se consigue con otras dos funciones JavaScript (Figura 5.14) que, al igual que las funciones que copian el código, se ejecutan cada vez que se hace click en los botones `Descargar código` de las interfaces.

```
//Saves a Python file with the generated Python code
function savePYFile(){
    var textToSave = document.getElementById("resultpython").value;
    var textToSaveAsBlob = new Blob([textToSave], {type:"text/plain;charset=utf-8"});
    var textToSaveAsURL = window.URL.createObjectURL(textToSaveAsBlob);
    var fileNameToSaveAs = "code.py";
    var downloadLink = document.createElement("a");
    downloadLink.download = fileNameToSaveAs;
    downloadLink.innerHTML = "Download File";
    downloadLink.href = textToSaveAsURL;
    downloadLink.style.display = "none";
    document.body.appendChild(downloadLink);
    downloadLink.click();
}
```

Figura 5.14: Código JavaScript: Descarga del fichero con el código Python

Por otra parte, se han creado otros dos procedimientos JavaScript cuyo objetivo es ejecutar las funciones correspondientes a la generación del código Block-Text y Python. Estos dos procedimientos también son ejecutados al hacer click sobre un botón. La diferencia con estas funciones es que deben hacer uso de la tecnología AJAX. En este caso, se leen los bloques que hay en el editor o el texto que hay en la interfaz Block-Text, y se devuelven los códigos Block-Text y Python (en el primer caso) o Python (en el segundo caso) correspondientes. Al ser funciones que conectan el Front-End con el Back-End de la página, se explicarán con profundidad más adelante en este mismo documento.

5.3.2. Librería Blockly. Adaptación a LearnBlockWeb

Para añadir la interfaz correspondiente al lenguaje visual, se ha hecho uso de una librería creada por Google con el objetivo de proporcionar editores de lenguajes de bloques aptos para incluir en sitios web: Blockly. La funcionalidad base de Blockly viene dada por dos archivos: uno de ellos contiene la definición de los bloques y las sentencias de sus inserciones en el editor, y en el otro está escrito todo el código que da forma y funcionamiento a la interfaz. Para que esta interfaz aparezca en la página *.html* deseada, hay que añadir además las líneas de código que aparecen en la Figura 5.15 y crear una estructura con etiquetas HTML similar a la de la Figura 5.10.

Ya que el desarrollador tiene acceso directo al código de la librería Blockly, posee la libertad de personalizarla a gusto. Aunque lo más común y sencillo a la hora de

```
<script>

    //Creates the workspace
    var workspace = LearnBlock.inject('learnblockDiv', {
        toolbox: document.getElementById('toolbox'),
        horizontalLayout: false,
        scrollbars: true
    });

</script>
```

Figura 5.15: Configuración inicial del editor Blockly

insertar Blockly en una web podría ser modificar los bloques que aparecen disponibles y el tamaño que ocupa el editor en la misma, todo lo que la librería ofrece puede ser alterado:

- Se pueden modificar todas las funcionalidades, añadir otras nuevas y eliminar las existentes que no quieran incluirse.
- Permite cambiar el diseño de todo lo que se presenta: tamaño, forma, color, transparencia, sombreado, orientación, resaltado...
- Los bloques que se insertan se pueden crear como se desee. Es posible cambiar su forma y su diseño visual, configurar el nombre que se presenta, y los tipos de conexiones que tiene, así como otros añadidos (parámetros del bloque, nombre o valor editable...)

Otra de las características que ofrece Blockly es la internacionalización: permite el cambio del idioma no solo de los bloques de la interfaz, sino de toda la página. Blockly está disponible en más de 40 idiomas, incluyendo aquellos que se leen de derecha a izquierda. Para integrar esta funcionalidad, se debe añadir un fichero *.js* extra que contenga la configuración del cambio de idioma, y otros tantos archivos como idiomas disponibles se desee que tenga la web, en los cuales vendrá definido un diccionario clave-valor con las traducciones correspondientes.

En resumen, se podría decir que Blockly presenta tres propiedades principales: interfaz de lenguaje de bloques, generador de código e internacionalización. Para el

desarrollo de LearnBlockWeb se ha hecho uso de la primera y de la tercera, las cuales se han sometido a una serie de cambios. Estos cambios se indican a continuación.

En lo que respecta a la interfaz, para la integración de Blockly en LearnBlockWeb se han eliminado varias funcionalidades y modificado otras tantas. La diferencia que más puede resaltar entre ambas interfaces es que, en LearnBlockWeb, el listado de bloques que aparece cuando se selecciona una categoría no desaparece cuando se selecciona un bloque o cuando se hace click en cualquier otra parte del editor, al contrario de lo que ocurre en Blockly, sino que lo hace cuando se vuelve a hacer click sobre la categoría ya seleccionada: esto hace que el usuario pueda saber en todo momento qué tiene disponible sin necesidad de realizar un gran número de clicks. Por otra parte, se ha eliminado la propiedad de que la interfaz emitiera sonidos, ya fuera a la hora de seleccionar un bloque, eliminarlo o conectarlo con otro. Se ha descartado la posibilidad de que un bloque se colapse y se expanda, de inhabilitarlo y volverlo a habilitar, y no se permite añadir comentarios en los mismos. Lo que en Blockly se conocen como bloques *shadow*, en LearnBlockWeb no existen.

Sin embargo, lo que más destaca en un primer vistazo general es la diferencia entre los diseños de las interfaces Blockly y LearnBlockWeb. Mientras que en la primera el listado de categorías tiene una orientación horizontal y se presenta un icono con un color distinto por cada una de ellas, en LearnBlockWeb esa misma lista tiene orientación vertical y las categorías no tienen ningún color asociado. Por otra parte, independientemente de su tamaño, el editor presenta ciertas diferencias en ambas versiones, sobre todo en lo referente al color de la zona principal y el listado de categorías y de bloques. A este último, además, se le ha añadido un ligero sombreado para aumentar su contraste con respecto a la zona del editor. Por otra parte, la fuente y el tamaño de letra de la interfaz y de los bloques también se ha modificado.

Ya se habló anteriormente en este documento sobre las formas y los colores de los bloques en LearnBlockWeb: la forma de las conexiones izquierda/derecha varían, mientras que la de las conexiones arriba/abajo se mantienen igual que en Blockly, y los colores dependen de la categoría a la que pertenezca el bloque.

Algo que llama la atención dentro de LearnBlockWeb es que algunos bloques presentan un espacio para introducir otros bloques en él. Esto es porque LearnBlock permite que, en los bloques de funciones que tienen parámetros, esos parámetros puedan ser variables. Para que LearnBlockWeb también pueda implementar ese detalle, la forma de hacerlo es permitiendo que se conecte el bloque que represente a una variable dentro del espacio del parámetro de la función. No obstante, esto trae consigo una modificación visual de todo el bloque.

Blockly tiene dos maneras de presentar visualmente los bloques que están conectados a otros dentro de una misma línea (Figura 5.2.3). La primera es de forma 'externa', en la que se unen ambos bloques uno al lado del otro, por separado (Figura 5.2.3, bloque superior). La segunda es la forma 'interna', en la que el bloque se introduce dentro de éste (Figura 5.2.3, bloque inferior). La forma 'externa' es la más intuitiva para las conexiones a la derecha, mientras que la forma 'interna' es la ideal para la inserción de parámetros (bloques de valores o variables) en una función. Ambas, tanto las conexiones a la derecha como las conexiones de los parámetros, se definen con el mismo atributo (`input_value`) dentro de la sintaxis Json del bloque. Por otra parte, cuál de estas dos formas adquiera el bloque depende del valor que tome otro de los atributos (`inputsInline`) dentro de su definición Json.

La librería Blockly fuerza a que, a la hora de crear un bloque, la forma de conexión se defina de una manera o de otra; no permite mezclar ambas visualizaciones en un mismo bloque. Esta es la razón por la que, en ocasiones, algunos bloques que permiten conexiones a su derecha incluyan un hueco para insertar otros bloques detrás del espacio correspondiente a los parámetros (Figura 5.16).

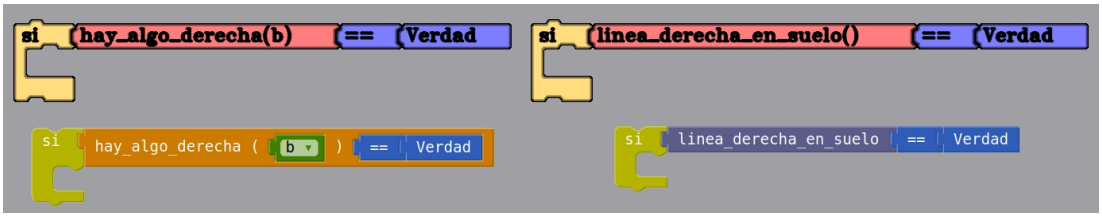


Figura 5.16: Conexiones a la derecha LearnBlock (arriba) y LearnBlockWeb (abajo)

Blockly ofrece la opción de añadir categorías que permiten generar bloques de forma dinámica. Estas categorías se utilizan para la creación de bloques que representen variables y funciones creadas por el propio usuario. Cuántos bloques se generan cada vez que se crea una variable o una función, y la forma de éstos, depende de lo especificado en el código de la librería. En LearnBlockWeb se ha cambiado la forma de los bloques generados cuando se crea una variable (Figura 5.17), y se ha eliminado la posibilidad de crear funciones que devuelvan algún valor (Figura 5.18).

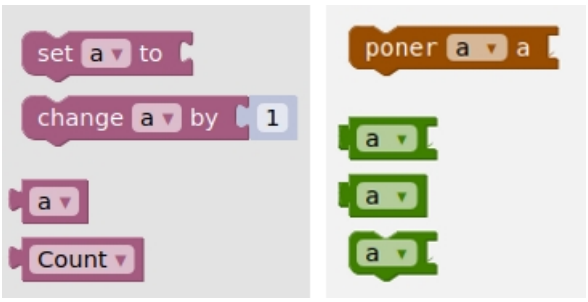


Figura 5.17: Variables Blockly (izq) - Variables LearnBlockWeb (der)

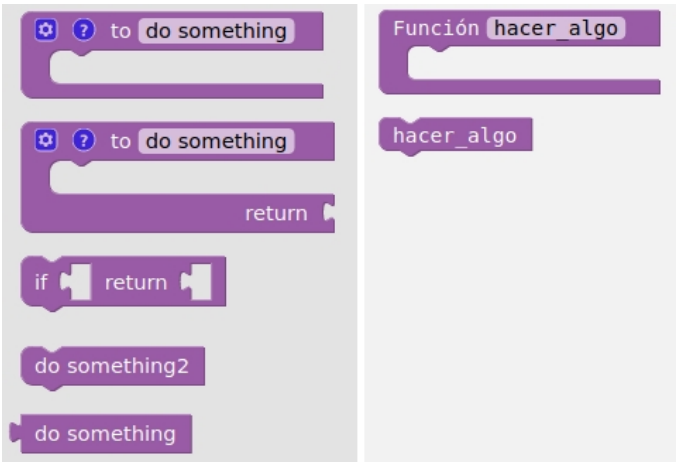


Figura 5.18: Funciones Blockly (izq) - Funciones LearnBlockWeb (der)

En Blockly existe la posibilidad de restringir qué bloques se conectan con otros mediante el tipo que tiene el bloque o el resultado de los bloques conectados. Así, si una expresión que equivale a un resultado numérico intenta conectarse a la condición de un bloque if, Blockly no lo permitirá, pues solo se podrían conectar resultados o expresiones booleanas. Esta restricción no se da en la mayoría de bloques de

LearnBlockWeb. Solamente se ha implementado para restringir qué tipo de bloques se insertan como parámetros en los bloques que representan funciones predefinidas.

Para que los bloques puedan ser leídos y añadidos al editor Blockly, deben ser definidos con una estructura similar a la que viene dada a continuación:

```
"type": "string_length",
"message0": 'length of %1',
"args0": [{"type": "input_value", "name": "VALUE"}],
"previousStatement": null,
"nextStatement": null,
"colour": 160
```

Listing 5.2: Ejemplo de definición de un bloque en Blockly

Una de las fortalezas de Blockly es su internacionalización. Actualmente, está traducido a más de cuarenta idiomas, entre los que se incluyen idiomas con distinta simbología a la del alfabeto romano y con sentido de lectura de derecha a izquierda. Puesto que la versión de escritorio de LearnBlock se ofrece tanto en inglés como en español, se ha querido aprovechar esta característica de Blockly para que LearnBlockWeb también sea accesible en esos dos idiomas. Para ello, se han incluido un fichero con el código necesario para reconocer y recargar la página con el idioma seleccionado, y otros dos que contienen el diccionario de palabras en ambos idiomas: uno que define los textos en inglés, y otro en español.

Hay diferencia a la hora de declarar los distintos textos de la página. El texto correspondiente a los bloques y demás palabras pertenecientes de forma directa a la interfaz del editor, se declaran y se llaman como se especifica en el Listado 5.3. El texto del resto de la página, se declara y se llama como aparece en el Listado 5.4.

```
Declaracion: LearnBlock.Msg["EMOTIONS"] = "Emotions";


Llamada: <category name="%{BKY_EMOTIONS}">
```

Listing 5.3: Definición del idioma del texto de la interfaz

```
Declaracion: var MSG = {programPY: "Program Python"}

Llamada: document.getElementById('programPY').textContent =
       MSG['programPY'];
*La sentencia de llamada debe ser escrita en el metodo de
 recarga de la pagina con el idioma seleccionado.
```

Listing 5.4: Definición del idioma del texto de la página

Todos los cambios realizados en la librería Blockly se han llevado a cabo para conseguir una mayor similitud entre LearnBlockWeb y lo ofrecido por el proyecto LearnBlock.

En los archivos que definen la librería Blockly añadidos a los ficheros del proyecto LearnBlockWeb se ha escrito una descripción escueta para cada uno de los métodos implementados que define de manera sencilla cuál es su funcionalidad. Además, todo el código de diseño que había implementado en JavaScript se ha traducido a CSS y añadido al fichero *styles.css*. Por otra parte, la definición de los bloques, también dada en JavaScript, se ha eliminado y se ha adaptado para que se carguen directamente desde los archivos con estructura Json del proyecto LearnBlock (apartado ”Lectura de Bloques”).

5.3.3. Accesibilidad Web

La accesibilidad es el grado en el que se mide si un objeto puede ser utilizado por todo el público, independientemente de sus capacidades técnicas o físicas, o de los conocimientos de los que disponga. Los documentos denominados *Pautas de Accesibilidad al Contenido en la Web* (WCAG) explican los aspectos a tener en cuenta para que el contenido de una página sea accesible y utilizable por el máximo número de personas. Ese contenido va desde texto hasta imágenes, formularios o sonido. La WCAG 2.0 está compuesta por los siguientes cuatro principios:

- Principio 1: Perceptibilidad.
- Principio 2: Operabilidad.
- Principio 3: Comprensibilidad.
- Principio 4: Robustez.

Las directrices básicas presentadas por WCAG para el cumplimiento de los criterios de accesibilidad y su aplicación en LearnBlockWeb se listan a continuación:

- Alternativas de texto para todo elemento no textual. Todas las imágenes en LearnBlockWeb tienen una alternativa textual, excepto las que se utilizan como elementos decorativos. La librería Blockly, por su parte, no ofrece una alternativa textual a las imágenes que forman los bloques.
- Contenido alternativo a la información presentada en audio o vídeo. En LearnBlockWeb no hay ninguno de estos dos elementos.
- Distinguible, de forma que sea más fácil para los usuarios que capten el contenido. El uso de los colores del fondo y de la información principal de manera que haya contraste, y el tamaño y fuente del texto de la web, son clave para lograr esto. El diseño de LearnBlockWeb se ha orientado para conseguir esto.
- Accesible mediante teclado. En LearnBlockWeb, se puede acceder a cada una de las pestañas saltando de una a otra con el tabulador, y se cambia el diseño de éstas cuando son resaltadas.
- Dejar bastante tiempo para que los usuarios puedan leer o usar adecuadamente el contenido. En LearnBlockWeb no hay nada que limite el tiempo de uso.
- No diseñar el contenido de tal forma que pueda ocasionar ataques, especialmente por el uso de ciertos contrastes de color en asociación con efectos de flash o similares. El uso del color en LearnBlockWeb busca la mayor comodidad visual para el usuario, y no hay nada que produzca ningún tipo de flash o parpadeo.

- Navegabilidad sencilla, facilidad para encontrar el contenido y determinar dónde se está dentro del sitio web. LearnBlockWeb es una página sencilla en la que no se presentan dificultades ante el usuario para saber dónde se encuentra.
- Entrada de datos utilizando otros dispositivos de entrada además del teclado. LearnBlockWeb puede ser usado con un ratón.
- El contenido textual es legible y fácilmente comprensible. El tamaño de letra, su fuente y su color se ha establecido para que el texto de LearnBlockWeb sea lo más cómodo de leer posible.
- La página web debe funcionar de una manera predecible por los usuarios. Todas las partes en LearnBlockWeb se han estructurado de una forma similar, por lo que esa repetición provoca que a la persona que utiliza la página le resulte predecible hacerlo.
- La página web debe ser lo más compatible posible con los navegadores actuales y futuros. Se busca cumplir con este criterio utilizando tecnologías como HTML5 y Bootstrap para desarrollar la página, que son compatibles con la mayoría de navegadores.

Se puede comprobar el cumplimiento de los criterios de accesibilidad de LearnBlockWeb accediendo al inspector de accesibilidad del navegador dentro de la página web.

5.4. Desarrollo Full-Stack: Back-End

5.4.1. Desarrollo en Python: Flask

Como ya se ha explicado, LearnBlockWeb se ha creado con el objetivo de proporcionar una alternativa a la versión de escritorio de LearnBlock, y se ha desarrollado con la idea de ofrecer la mayor similitud posible entre ambos proyectos, tanto en la parte visual e interactiva (Front-End) como en el funcionamiento que la página tiene detrás (Back-End). Esta es la razón por la que la parte lógica de LearnBlockWeb también se ha escrito en Python: es la manera más sencilla de integrar directamente las funcionalidades de LearnBlock, aumentando así dicha similitud. Además, con esto se pretende facilitar todo lo posible la labor del desarrollador a la hora de integrar los cambios que se hayan podido realizar en LearnBlock en un futuro.

LearnBlockWeb hace uso de Flask para integrar la parte del Front-End de la web con la parte del Back-End escrita en Python 3. Para esto, basta con importar la librería y crear una instancia del objeto Flask en el fichero desde el que se realiza la ejecución.

La forma de manejar Flask es mediante el concepto de `rutas`. El objetivo es asociar a cada ruta una funcionalidad diferente. Para desplegar LearnBlockWeb, la única ruta de la que se hace uso es de la que contiene la interfaz principal. Teniendo esto en mente, se ha creado un procedimiento que se encargue de cargar la página HTML que contiene dicha interfaz, la cual debe estar en el directorio *templates*, cada vez que se acceda a la ruta raíz. La ruta raíz (”\”) es la primera a la que se entra cuando la aplicación se ejecuta. Por lo tanto, cuando se despliega la página web, se carga y se muestra la interfaz.

Por defecto, el servidor de desarrollo proporcionado por Flask se encuentra en la dirección 127.0.0.1:5000. Cada vez que se lanza la página aparecerá en esa dirección el resultado de la ejecución de la misma.

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route("/")
def init():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(debug=False)
```

Figura 5.19: Código base de la ejecución de LearnBlockWeb con Flask

5.4.2. Lectura de Bloques

Los bloques disponibles en LearnBlock son cargados en la interfaz a partir de una definición estructurada con formato Json. En ella se definen el tipo del bloque, la categoría a la que pertenece, su nombre, la forma, el texto presentado en inglés y en español, y las variables que pueda llegar a tener. A partir de esa definición, LearnBlock crea un bloque con las características especificadas y lo añade a la lista de bloques de la interfaz.

Las definiciones de todos los bloques que hay en LearnBlock están agrupadas en seis ficheros con extensión *.conf* según la funcionalidad de los bloques que integran. Los archivos, junto con los criterios de agrupación de los bloques que contienen, son los siguientes:

- Control.conf: Estructuras de control. Bloque principal, bucles, condicionales y de control del tiempo.
- Operators.conf: Operadores aritméticos y lógicos.
- Motor.conf: Funciones de movimiento del robot con su cuerpo o sobre una superficie.
- Expressions.conf: Funciones con las que el robot expresa distintas emociones.
- Perceptual.conf: Funciones con las que el robot percibe detalles de su entorno, líneas, imágenes u obstáculos.

- Proprioceptive.conf: Funciones con las que el robot percibe qué está haciendo, si está expresando alguna emoción o si está realizando algún movimiento.

La forma que adopta un bloque en LearnBlock viene dada por el atributo `shape` dentro de la definición Json del mismo. La relación entre el valor de ese atributo y su forma se especifica en la Tabla 5.1.

Tabla 5.1: Formas y definición de los bloques en LearnBlock

	block1	blockVertical
	block2	
	block3	blockBoth
	block4	blockLeft
	block5	
	block6	
	block8	

La estructura Json que debe tener la definición de un bloque para ser leído por LearnBlock es la que aparece en el Listado 5.1.

La librería Blockly, sin embargo, utiliza una estructura diferente a la hora de definir los bloques que se van a insertar en su interfaz. Esta estructura, aunque sea distinta, define los mismos atributos que LearnBlock con la suya: tipo del bloque, texto que aparece en él, forma y variables. La estructura Blockly no especifica la categoría a la que pertenece el bloque, aunque sí indica el color que va a tomar.

La forma que adopta un bloque en Blockly viene dada por una serie de atributos que aparecerán o no definidos dentro de la estructura, según cómo se quiera generar el

mismo. La relación entre los atributos que se pueden añadir y la forma que adopta el bloque en consecuencia se especifica en la Figura 5.20.

	Conexión dentro	"message0": "name" "message1": "%1" "args1:[{"type":"inputstatement", "name":"DO"}]
	Conexión arriba y abajo	"nextStatement": null "previousStatement": null
	Conexión a la derecha	"message0": "%1" "args0":[{"type":"inputvalue" "name":"VALUE"}]
	Fin de las conexiones a la derecha	"output": null
	Parámetros de una función	"message0": "(%1)" "args0":[{"type":"inputvalue", "name":"PARAM","check":"Number"}] "inputsInline": true"
	Parámetros de una función + Conexión a la derecha	"message0": "(%1) %2" "args0":[{"type":"inputvalue", "name":"PARAM","check":"Number"}, {"type":"inputvalue","name":"DO"}] "inputsInline": true"

Figura 5.20: Formas y definición de los bloques en Blockly

La estructura Json que debe tener la definición de un bloque para ser leído por Blockly es la que aparece en el Listado 5.2.

Para que Blockly pueda insertar los bloques de forma correcta en su editor y dejarlos a disposición del usuario, la carga y lectura de las estructuras Json deben ser realizadas **antes** de crear la instancia de la interfaz.

Ya se ha visto que ambas definiciones tienen una estructura distinta. Para que los bloques disponibles en LearnBlock también lo estén en LearnBlockWeb, es necesario crear una configuración acorde a la de Blockly con cada uno de ellos. No es complicado hacerlo de forma manual, pero resulta tedioso cada vez que se quiera añadir, cambiar o eliminar algún bloque del proyecto LearnBlock. Por lo tanto, se ha decidido crear una serie de funciones que, dada la definición Json de un bloque en LearnBlock, genere otra estructura Json con la definición de ese mismo bloque en Blockly. Así, se consigue crear ese cambio de configuración de manera automática, sin necesidad de redefinir la estructura ni de modificar ninguna sentencia del código de la web.

Destacar que se han seguido unas pautas personalizadas a la hora de definir

los bloques Blockly para facilitar su procesamiento e interpretación en el resto de funcionalidades desarrolladas para LearnBlockWeb:

- Añadido un campo extra: `blocktextname`. Este campo contiene el valor `name` de la definición de bloque de LearnBlock, y es con el que después se representa su equivalencia en el lenguaje Block-Text.

- El valor del campo `message0` no viene dado por un texto plano, sino por una sentencia del tipo `%{BKY_NAME}`, con la que se muestra el nombre del bloque según el idioma en el que se esté presentando la página en el momento.

- La categoría viene dada en el tipo del bloque. El tipo se define con la estructura `categoría_nombre`.

El resumen de la relación entre los atributos de las estructuras LearnBlock y LearnBlockWeb aparece en la Tabla 5.2.

Tabla 5.2: Relación entre estructuras de bloques LearnBlock y LearnBlockWeb

LearnBlock	**LearnBlockWeb**
type	–
category	type
name	blocktextname
languages (eng, esp)	message0 (clave BKY)
variables	args0
–	colour

Hay una serie de bloques que no vienen definidos en LearnBlock con una estructura escrita en Json. Estos son los bloques correspondientes a los valores (textuales y numéricos), a las variables y a las funciones de usuario. Por lo tanto, para que puedan ser cargados por LearnBlockWeb, se ha definido su estructura directamente con la sintaxis de Blockly. Son los únicos tipos de bloques que no se obtienen a partir de una traducción, pues no se leen desde los ficheros LearnBlock.

Los bloques, pues, se cargan en LearnBlockWeb a partir de ocho ficheros de texto que contienen sus definiciones: seis de ellos, los especificados anteriormente,

contienen la sintaxis de los bloques de LearnBlock; los otros dos están escritos con la sintaxis de los bloques de Blockly.

Ya se ha explicado que, al lanzar la ejecución en el servidor de desarrollo Flask, lo que se hace es cargar la página *.html* que contiene la interfaz. Para que se cargue correctamente, es necesario que antes de eso se lean y se interpreten las definiciones de los bloques (Figura 5.21).

```
# This procedure is executed when the web page is opened
@app.route("/")
def init():
    # Loads blocks from json definition
    control = loadLBBlocks("blocks/Control.conf")
    operators = loadLBBlocks("blocks/Operators.conf")
    values = loadBlocks("blocks/values.json")
    variables = loadBlocks("blocks/variables.json")
    expressions = loadLBBlocks("blocks/Expressions.conf")
    proprioceptive = loadLBBlocks("blocks/Proprioceptive.conf")
    motor = loadLBBlocks("blocks/Motor.conf")
    perceptual = loadLBBlocks("blocks/Perceptual.conf")
```

```
def loadBlocks(route):
    f = open(route, "r")
    if f.mode == 'r':
        result = f.read()
    return result

# Loads a file to process blocks in Json
def loadLBBlocks(route):
    f = open(route, "r")
    if f.mode == 'r':
        blocks = f.read()
    result = json.loads(blocks)
    return convertLB2Blockly(result)
```

Figura 5.21: Código para la lectura de ficheros de bloques

El método `convertLB2Blockly` lee uno a uno todos los bloques de la estructura Json con sintaxis LearnBlock conseguida y los procesa para crear otra estructura Json que defina el bloque equivalente en sintaxis Blockly. Añadir que, si en la definición de un bloque LearnBlock hay más de un tipo de forma, significa que se desea crear dos bloques exactamente iguales que solamente se diferencien en su forma.

Por cada bloque conseguido, se deben obtener todos los atributos de la definición Blockly a partir de la definición LearnBlock. La manera de generar de forma correcta dichos atributos, según las pautas establecidas para LearnBlockWeb, es la siguiente:

- Atributo `type`: Atributo `category` en LearnBlock + `_` + atributo `name` en LearnBlock.
- Atributo `blocktextname`: Atributo `name` en LearnBlock.
- Atributo `message0`: Clave `BKY` generada a partir del nombre del bloque.
- Atributo `args0`: Dado según los tipos de los parámetros. Especificado en la Figura 5.22.

- Atributo `colour`: Dado en código hexadecimal. Depende de la categoría a la que pertenezca el bloque.
- Forma: Se añaden atributos según la forma que se desee. Ejemplos en la Figura 5.20, especificado en detalle en la Figura 5.23.
- Idioma: Se genera una lista de tres elementos que contiene el nombre del bloque, su traducción al español y su traducción al inglés. Se utilizará a la hora de crear entradas en los diccionarios de idiomas de forma automática.

Float, Int	{"type": "input_value", "name": "PARAM", "check": ["**Number**", "var"]}
String, Apriltext	{"type": "input_value", "name": "PARAM", "check": ["**Text**", "var"]}
Boolean	{"type": "input_value", "name": "PARAM", "check": ["**Bool**", "var"]}

Figura 5.22: Sintaxis de parámetros en LearnBlockWeb según su tipo

	block1, blockVertical	"previousStatement": null, "nextStatement": null
	block2	"args0": [{"type": "input_value","name": "VALUE"}], "previousStatement": null, "nextStatement": null
	block3, blockBoth	"args0": [{"type": "input_value", "name": "NUM"}], "output": null
	block4, blockLeft	"output": null
	block5	"message1": "%1", "args1": [{ "type": "input_statement", "name": "DO"}], "previousStatement": null, "nextStatement": null
	block6	"args0": [{"type": "input_value", "name": "IF"}] "message1": "%1","args1": [{ "type": "input_statement", "name": "DO"}], "previousStatement": null, "nextStatement": null
	block8	"message1": "%1", "args1": [{ "type": "input_statement", "name": "DO"}]

Figura 5.23: Relación forma - definición de bloques

Una vez realizado todo el proceso de conversión, en el método `init`, que es el que se ejecuta al lanzar la página, se asigna el resultado a una serie de variables.

Dentro de cada variable se encuentra una estructura de dos elementos: el primero es la definición con sintaxis LearnBlockWeb (Blockly) correspondiente a los bloques procesados, y el segundo contiene el objeto en el que se guardan el nombre del bloque y sus traducciones al inglés y al español. Además de esto, se ha creado un nueva objeto a partir de estos elementos en el que se añade una lista con los tipos (atributo `type`) de todos los bloques y las categorías a las que pertenecen, con el objetivo de usarla para cargar los bloques en el fichero *.html* de la interfaz de forma dinámica. Todo esto: las traducciones de los dos idiomas, la definición de los bloques, y la lista de tipos de bloques y sus categorías, deben insertarse en el `return render_template` del método init para que la página pueda hacer uso de toda esa información y procesarla como es debido.

Dentro del fichero *.html*, antes de instanciar la interfaz del editor de bloques, la librería Blockly debe cargar la definición de todos los bloques que se desean insertar. Para ello, se llama al método que se encarga de realizar esto una vez por cada variable que contenga definiciones de bloques. A ese método, la primera vez que es llamado, se le debe pasar por parámetro el objeto que guarda las traducciones a los distintos idiomas (Figura 5.24).

```
<script>
    //Loads blocks from json definition
    loadBlocks({{control | tojson}}, {{language | tojson}});
    loadBlocks({{operators | tojson}}, null);
    loadBlocks({{values | tojson}}, null);
    loadBlocks({{variables | tojson}}, null);
    loadBlocks({{expressions | tojson}}, null);
    loadBlocks({{proprioceptive | tojson}}, null);
    loadBlocks({{motor | tojson}}, null);
    loadBlocks({{perceptual | tojson}}, null);

    //Creates the workspace
    var workspace = LearnBlock.inject('learnblockDiv', {
        toolbox: document.getElementById('toolbox'),
        horizontalLayout: false,
        scrollbars: true
    });

</script>
```

Figura 5.24: Configuración completa del editor

Este procedimiento primero se encarga de generar nuevas entradas en el diccionario del idioma seleccionado a la hora de cargar la página. Cada una de estas entradas

contiene la clave BKY y el nombre del bloque en el idioma correspondiente. La definición de dichas entradas está escrita en el Listado 5.3. Una vez creadas, el método introduce las definiciones de los bloques en la librería Blockly (Figura 5.25).

```
function loadSpanish(languagesBlocks){
    if (languagesBlocks != null){
        for (var i=0; i<languagesBlocks.length; i++){
            LearnBlock.Msg[languagesBlocks[i][0]] = languagesBlocks[i][1];
        }
    }
}
```

```
function loadBlocks(blocks, idioma){
    if(Code.LANG == "es"){
        loadSpanish(idioma)
    }
    if(Code.LANG == "en"){
        loadEnglish(idioma)
    }
    LearnBlock.defineBlocksWithJsonArray(JSON.parse(blocks));
}
```

Figura 5.25: Carga de bloques y entradas del diccionario

Como ya se ha dicho anteriormente en la descripción del Front-End de la web, los bloques que se quieran insertar en la interfaz Blockly deben ser definidos de manera estática en el documento *.html* que vaya a contener el editor, dentro de la etiqueta de la categoría a la que se quieran asignar. La sentencia HTML para añadir un bloque es la definida en el Listado 5.5.

```
<block type="atributo_type_del_bloque"></block>
```

Listing 5.5: Etiqueta HTML de un bloque

Si se realiza de forma estática, hay que ir añadiendo una a una todas las etiquetas que correspondan a todos los bloques que se quieran cargar. Para evitar esto, se utiliza la lista que contiene objetos con los tipos de los bloques y las categorías a las que pertenecen, creada en el procedimiento de carga de la web. De esta manera, por cada categoría que exista, se crea un bucle que recorre la lista y genera una etiqueta HTML para cada bloque que pertenezca a dicha categoría (Figura 5.26).

```
<category name="%{BKY_MOTOR}">
    {% for block in allBlocks %}
    {% if block[1] == 'motor' %}
    <block type="{{ block[0] }}"></block>
    {% endif %}
    {% endfor %}
</category>
```

Figura 5.26: Código de la definición de los bloques en HTML

5.4.3. Traducción de Bloques a Block-Text

La funcionalidad principal de LearnBlockWeb es, junto con la programación por bloques, la generación de código. LearnBlock, de la misma manera que LearnBlockWeb, traduce a dos lenguajes a partir de la estructura construida con los bloques: Block-Text y Python. Para cada uno de esos dos lenguajes se ha creado un intérprete que toma la información del código inmediatamente anterior y genera el equivalente. Así, el intérprete de Block-Text usa la información que le proporcionan los bloques, y el intérprete de Python la obtiene del código Block-Text.

La información de los bloques que recibe el generador de código Block-Text debe tener una estructura concreta para que éste pueda procesarla. Por cada bloque que se vaya a tratar, se debe generar un objeto que cumpla con la siguiente disposición:

- Nombre: Nombre del bloque. Se utilizará para escribir su equivalencia en el código Block-Text.
- Diccionario: Objeto que contiene los detalles del bloque.
    - RIGHT: Bloque que se encuentra a la derecha del mismo. Toma valor `None` si no hay ninguno.
    - BOTTOMIN: Bloque que se encuentra dentro del mismo. Toma valor `None` si no hay ninguno.
    - BOTTOM: Bloque que se encuentra debajo del mismo. Toma valor `None` si no hay ninguno.
    - VARIABLES: Parámetros del bloque, en el caso de que sea una función.

Valor que toma el bloque, en el caso de que sea el setter de una variable. Toma valor `None` si no hay parámetros o si no es un setter.

- TYPE: Tipo del bloque. Según LearnBlock, los tipos son los siguientes:
  - OPERATOR (0): Bloques de operadores.
  - CONTROL (1): Bloques de estructuras de control.
  - FUNTION (2): Funciones predefinidas por LearnBlock. Bloques que pertenecen a las categorías Base, Cámara, Distancias, Speaker, Emociones, Motor y Suelo. La función `esperar` de la categoría Control también tiene este tipo asignado.
  - VARIABLE (3): Variables creadas por el usuario.
  - USERFUNCTION (4): Funciones creadas por el usuario.
  - Si el bloque no se corresponde con ninguna de estas descripciones, TYPE toma el valor `None`.

Importante destacar que lo que se inserta en los atributos RIGHT, BOTTOMIN y BOTTOM no es el nombre del bloque correspondiente, sino el objeto completo de dicho bloque. El resultado final es un objeto que a su vez tiene objetos dentro. Un ejemplo del objeto resultante viene dado en el Listado 5.6, donde aparece la estructura correspondiente a la conexión de bloques de la Figura 5.2.2: un bloque debajo de otro bloque.

```
('look_front', {'RIGHT': None, 'BOTTOMIN': None, 'BOTTOM':
       ('move_straight', {'RIGHT': None, 'BOTTOMIN': None,
        'BOTTOM': None, 'VARIABLES': None, 'TYPE': 2}),
 'VARIABLES': None, 'TYPE': 2})
```

Listing 5.6: Estructura de datos "Bloque"

Por lo tanto, a la hora de realizar la traducción a Block-Text, es necesario generar un objeto con esa disposición en el que se definan todos los bloques que el usuario ha creado y conectado entre sí en el editor.

La librería Blockly ofrece un método que devuelve una lista con todas las variables creadas por el usuario, y otro que devuelve una estructura XML en la que se especifican los detalles de todos los bloques que hay en el editor de texto en ese momento. Se ha creado una función JavaScript que obtenga ambos resultados en el momento en el que el usuario hace click sobre el botón `Bloques a Texto` de la interfaz (Figura 5.27).

```
function execute() {
    var loadBlocks;
    var vars;
    var result;
    if (window.sessionStorage) {
        var xml = LearnBlock.Xml.workspaceToDom(Code.workspace);
        var text = LearnBlock.Xml.domToText(xml);
        window.sessionStorage.loadOnceBlocks = text;
    }
    try {
        loadBlocks = window.sessionStorage.loadOnceBlocks;
        vars = Code.workspace.getAllVariables();
        result = getVarsNames(vars) + '\n' + loadBlocks;
    } catch (e) {
        loadBlocks = null;
        vars = null;
    }
```

Figura 5.27: Código con el que se obtienen los bloques del editor

Ya se ha hablado de las funciones AJAX en un apartado anterior. AJAX se usa para obtener datos desde el servidor y devolver esa misma información u otra distinta de forma asíncrona, haciendo así que la página se actualice con esos nuevos datos sin necesidad de recargarla. En este caso, el objetivo es obtener la definición de los bloques que hay en el editor en el momento en el que se pulsa el botón, procesarlos para crear la estructura de datos equivalente con la que generar las traducciones, y mostrar en la web los códigos generados. Para conseguir esto, a la función de la Figura 5.27 se le ha añadido el código AJAX que permite pasar la información de los bloques obtenida desde el Front-End de la web hasta una función del Back-End que se encargue de procesarla, y que a su vez le devuelve el resultado de ese procesamiento para poder mostrarlo en la página (Figura 5.28).

```
$.ajax({
    type: "POST",
    contentType: "application/json;charset=utf-8",
    url: "/result",
    traditional: "true",
    data: JSON.stringify(result),
    dataType: "text",
    success: function (data) {
```

Figura 5.28: Código AJAX

- Type: Toma el valor `Get` si se quieren recibir datos. Toma el valor `Post` si se quieren enviar.
- ContentType: Tipo del dato que se envía.
- URL: Dirección a la que se envían los datos. En el Back-End, se crea una función Flask que se ejecute cada vez que se acceda a la url especificada en este parámetro y que devuelva un valor concreto, que es el que recibe la función AJAX.
- Data: Se introducen los datos a enviar, y en el propio parámetro `data` se guarda el resultado obtenido de la función ejecutada. En este caso, se envían la lista de variables y la estructura XML de los bloques, y se obtienen los códigos Block-Text y Python equivalentes.
- DataType: Tipo del dato obtenido después de la ejecución.
- Success: Especifica qué lleva a cabo la función AJAX si el intercambio de datos ha tenido éxito. En este caso, se crea una función con la que se asocian los códigos Block-Text y Python generados a distintos elementos de la página *.html*.

La función del Back-End ejecutada al hacer click en el botón `Bloques a Texto` es la que tiene asignada la ruta `/result`. En ella, se obtiene el XML de los bloques que hay en el editor a través del request, se procesa para obtener el objeto `Bloque` equivalente, a partir del cual se ejecuta la generación del código Block-Text y Python, y devuelve dichos códigos (Figura 5.29).

```
@app.route("/result", methods=['GET', 'POST'])
def getBlocks():
    if request.method == 'POST':
        blocks = request.get_json()
        processVars(getVars(blocks))
        toParser = formatBlocks(blocks)
        blocktext = parserBlockText(toParser)
```

Figura 5.29: Función ejecutada de forma asíncrona

Dentro de la sintaxis XML de un bloque se especifican el tipo y el nombre del mismo, al igual que los bloques que tiene conectados a su derecha, dentro de él y debajo de él. También se especifican los parámetros que pueda tener cada bloque. En la Figura 5.30 aparece una comparación entre una estructura de bloques sencilla y el código obtenido que la representa.

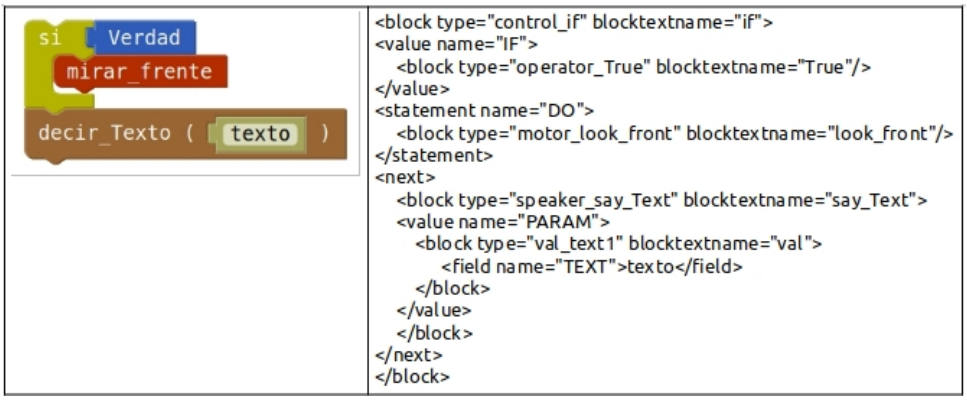


Figura 5.30: Programa con bloques y sintaxis XML correspondiente

En la Tabla 5.3 se lista la relación que hay entre la cada atributo de la sintaxis XML y su equivalencia con los atributos de la estructura de datos `Bloque`.

Tabla 5.3: Relación Objeto Bloque - XML

Bloque	XML
Nombre	BlockTextName
Dic['TYPE']	Type (hasta llegar al primer '_')
Dic['RIGHT']	Value (`name` distinto a PARAM)
Dic['BOTTOMIN']	Statement
Dic['BOTTOM']	Next
Dic['VARIABLES']	Field de Value (`name` igual a PARAM)

La manera de procesar el resultado XML obtenido es recorrer en profundidad la estructura generada hasta haber alcanzado todos y cada uno de los bloques que la conforman. Por cada bloque, se debe generar un objeto `Bloque` que lo represente, y se debe comprobar si está conectado con algún otro bloque a su derecha (value), dentro de él (statement), o debajo (next). Si tiene alguna de esas conexiones, se crea el objeto del bloque con el que está conectado y se inserta en su atributo RIGHT, BOTTOMIN o BOTTOM, respectivamente. Si no tiene ninguna conexión, el objeto `Bloque` creado tendrá asignado el valor `None` a esos tres atributos.

Este procesamiento se consigue de forma sencilla con un método recursivo que crea el bloque, realiza dichas comprobaciones e inserciones, y devuelve el objeto creado. El pseudocódigo de este método está escrito en el Listado 5.7.

```
procesarBloque(bloqueXML)
    objetoBloque = crearBloque(bloqueXML)
    if bloqueXML.value != None and bloqueXML.value.name != "PARAM"
        bloqueValue = procesarBloque(bloqueXML.value)
        insertarBloque(objetoBloque, bloqueValue, RIGHT)
    if bloqueXML.statement != None
        bloqueStatement = procesarBloque(bloqueXML.statement)
        insertarBloque(objetoBloque, bloqueStatement, BOTTOMIN)
    if bloqueXML.next != None
        bloqueNext = procesarBloque(bloqueXML.next)
        insertarBloque(objetoBloque, bloqueNext, BOTTOM)
    return objetoBloque
```

Listing 5.7: Pseudocódigo del procesamiento de bloques

La creación de un objeto `Bloque` es tan sencilla como asignar los valores correspondientes según lo especificado en la Tabla 5.3. Sin embargo, hay algunos casos concretos en los que el objeto a generar no se puede obtener siguiendo esa relación con la sintaxis XML obtenida. Cuando se dan esas situaciones, se deben llevar a cabo un conjunto de ajustes que aparecen listados a continuación:

- Si el bloque pertenece a la categoría `Valores`, se debe insertar en el nombre del bloque lo que aparece entre los campos `field` del XML. En el caso en el que el valor sea textual, ese nombre debe aparecer entre comillas dobles ().
- Si el bloque es la definición de una función de usuario, se debe insertar en el nombre del bloque lo que aparece entre los campos `field` del XML.
- Si el bloque es la llamada a una función de usuario, se debe insertar en el nombre del bloque lo que aparece entre los campos `mutation` del XML.
- Si el bloque es una variable creada por el usuario, se debe insertar en el nombre del bloque lo que aparece entre los campos `field` del XML.
- Si el bloque es el setter de una variable creada por el usuario, se debe insertar en el nombre del bloque lo que aparece entre los campos `field` del XML y en el atributo VARIABLES del bloque se debe añadir el nombre del bloque inmediatamente a su derecha.

Con todo este proceso, ya se puede obtener el objeto `Bloque` con la estructura adecuada para que sea interpretado y procesado por el generador de código Block-Text de manera correcta.

5.4.4. Traductores Block-Text y Python

Como ya se ha especificado, LearnBlock dispone de dos traductores que se encargan de generar código en dos lenguajes de programación distintos, Block-Text y Python, a partir de la información que reciben. LearnBlockWeb utiliza esos mismos generadores de código a la hora de realizar la traducción entre los distintos lenguajes. El proyecto LearnBlockWeb contiene los ficheros del proyecto LearnBlock en los que se ha implementado el código de dichos traductores.

El traductor Block-Text debe recibir una estructura de datos como la definida en el apartado anterior, que contenga una representación de todos los bloques del programa creado con el lenguaje visual. A partir de ésta, el traductor se encarga de generar un

texto que contenga los nombres de todos los bloques, además de una serie de símbolos, saltos de líneas y tabulaciones, según de qué tipo de bloque se trate y cuál sea su posición.

El código que genera el lenguaje Block-Text se encuentra en dos archivos distintos en el proyecto LearnBlock. La primera parte del intérprete, que procesa las variables y los bloques de procedimientos creados por el usuario, así como el bloque principal, se ha obtenido del archivo *learnbot_dsl/learnbotCode/**LearnBlock.py*** y se encuentra dentro de LearnBlockWeb en el fichero *btParser.py*. Se han creado algunos cambios en el código a la hora de realizar las llamadas y procesar el contenido que recibe por parámetro, pero la estructura principal es la misma que la de LearnBlock.

La segunda parte del intérprete procesa uno a uno los bloques que hay dentro del bloque principal y de los bloques correspondientes a las funciones creadas por el usuario. Este código viene dado en el proyecto LearnBlock dentro del archivo *learnbot_dsl/learnbotCode/**VisualBlock.py***. Este ha sido insertado directamente en el proyecto LearnBlockWeb, en la ruta *learnblockCode/**VisualBlock.py*** tras haber sido sometido a una serie de cambios con los que se ha eliminado todo el código referente a la parte de la interfaz visual de LearnBlock.

El funcionamiento de la primera parte de la traducción es el siguiente:

- Si hay variables creadas por el usuario, éstas se han debido introducir previamente en una lista, de la cual el generador de código saca sus nombres. Por cada una de ellas, genera un texto en el que iguala su nombre a None: `var = None`. Si no hay variables, no genera ningún texto.

- Si hay funciones creadas por el usuario, por cada una de ellas se genera un texto que empieza con la palabra `def`, continúa con lo obtenido de la segunda parte del intérprete, y termina con la palabra `end`. Si no hay funciones, no genera ningún texto.

- Tras lo anterior, se comienza el procesamiento del bloque principal. Se genera un texto que comienza con la palabra `main:`, continúa con lo obtenido de la segunda

parte de la generación de código, y termina con la palabra `end`. La llamada a la segunda parte de la generación de código se realiza en el caso de que haya algún bloque dentro de él; en caso contrario, se escribe la palabra `pass` en el lugar correspondiente dentro del texto generado.

El funcionamiento de la segunda parte de la generación de código es el siguiente:

- Se escribe el nombre del bloque.
- Si se trata de una función creada por el usuario, ya sea definición o llamada de la misma, se escribe su nombre en el texto seguido de '()'.
- Si es una función de tipo `control` y tiene parámetros asignados, se escribe su nombre junto a '(' + parámetros separados por una coma + ')'.
- Si es un bloque de tipo `function`, se añade `function.` antes del nombre, seguido por '()'. Entre los paréntesis aparecen parámetros separados por comas en caso de que la función los tuviera.
- Si es un bloque de tipo `variable` y tiene un valor asignado, se iguala su nombre a dicho valor: `var = 2`.
- Si hay un bloque a su derecha, se llama de nuevo a este intérprete para procesarlo tras introducir en el texto un carácter en blanco.
- Si hay un bloque dentro, se escribe ':' y se añaden un salto de línea y un tabulador antes de llamar otra vez al intérprete para procesarlo.
- Si el bloque es una estructura de control (while, if...) escribe `end` en el texto después de haber procesado todos los bloques que tuviera dentro.
- Si hay un bloque debajo, se añade un salto de línea antes de llamar de nuevo al generador de código para procesarlo.

El orden en el que aparecen todos los elementos en estas dos listas es el orden que siguen ambos generadores para realizar el procesamiento.

El generador Block-Text se llama dentro de la función `getBlocks` del fichero *learnblock.py*, que es la que se ejecuta cada vez que se hace click en el botón `Bloques a Texto` de la interfaz. Antes de realizar la llamada se ha debido recuperar la sintaxis XML que representa a los bloques que hay en el editor en el momento de hacer click en el botón y se han debido generar los objetos `Bloque` equivalentes. El generador devuelve todo el código Block-Text obtenido en forma de *string*.

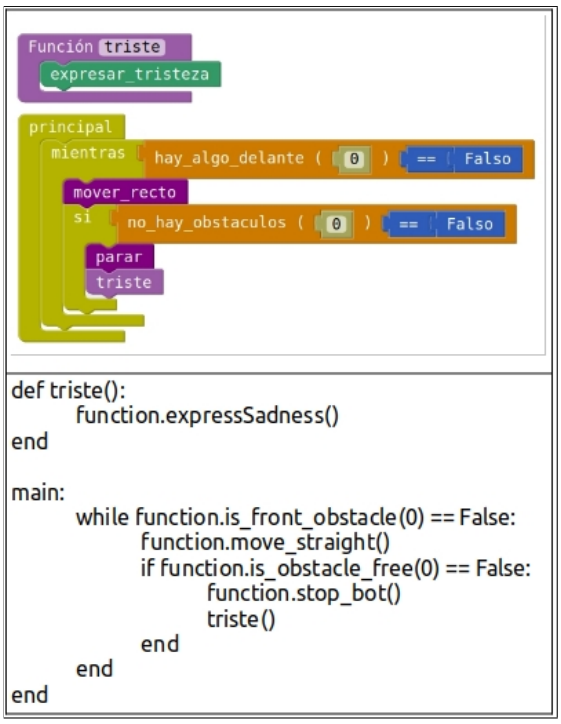


Figura 5.31: Código Block-Text generado

El generador de código Python necesita las sentencias Block-Text para usarlas como referencia a la hora de generar el código. Por cada una de esas sentencias leídas por el intérprete, éste genera la sentencia equivalente en lenguaje Python. Recordar que el código Python generado por LearnBlock no se corresponde únicamente con el programa creado por el usuario, sino que también incluye una serie de sentencias necesarias para crear una instancia de la clase `robot`, la cual se encarga de establecer una comunicación con la plataforma robótica que se vaya a utilizar.

El código que genera el lenguaje Python se encuentra en el fichero *learnbot_dsl/learnbotCode/**Parser.py*** del proyecto LearnBlock. Ese archivo se ha

introducido en LearnBlockWeb en la ruta *learnblockCode/**Parser.py***.

Para llevar a cabo la traducción, se hace uso de la librería *PyParsing*. Esta librería ofrece una manera sencilla de crear y ejecutar gramáticas simples directamente en código Python, sin necesidad de utilizar expresiones regulares o los programas *Lex* y *Yacc*.

Para generar una gramática, se debe crear una serie de símbolos que tendrán asignados o bien valores concretos (caracteres), o bien algún otro símbolo o un conjunto de ellos. En el caso de este generador Python, la primera parte de la gramática asigna a cada símbolo un carácter concreto: operadores, símbolos de escritura, tabulaciones... (Figura 5.32). En la segunda parte de la gramática se declaran símbolos creados con otros símbolos anteriores, que representan estructuras más complejas como comparaciones, condiciones o bucles (Figura 5.33).

```
L = Literal("<")
NL = Literal(">")
LE = Literal("<=")
NLE = Literal(">=")
E = Literal("==")
NE = Literal("!=")
END = Literal("end")
SECTAB = ZeroOrMore("\t")
CHAINBETTENQUOTE = Group(QuotedString('"')).setResultsName("STRING")
```

Figura 5.32: Ejemplo de símbolos simples

```
"""----------------COMPARACIONES-----------------"""
COMP = Group(L | NL | LE | NLE | E | NE).setResultsName("COMP")
ORAND = Group(OR | AND).setResultsName('ORAND')
```

Figura 5.33: Ejemplo de símbolos compuestos

Una vez declarado un símbolo por cada una de las estructuras que pueden existir en Python, dentro del generador se ha creado un conjunto de funciones que, dado el código Block-Text, procesan su contenido utilizando la gramática definida para comprobar que su disposición es correcta y, de ser así, escriben el código Python equivalente. Si hay algún fallo en la sintaxis o en la semántica del código Block-Text a procesar, las reglas de la gramática no se cumplen y esas funciones no generan ningún código.

A la hora de generar el código Python completo, se utiliza la traducción obtenida del procesamiento de la gramática junto con una serie de sentencias que hacen que, si se ejecuta ese código, se produzca la conexión con el robot a utilizar, previamente seleccionado. Esas sentencias son introducidas antes y después del código generado.

El generador de código Python se llama dentro de las funciones `getBlocks` y `getPyFromBT` del fichero *learnblock.py*. La primera función se ejecuta cada vez que se hace click en el botón `Bloques a Texto` de la interfaz, y en este caso necesita que el generador Block-Text haya generado previamente el código equivalente a los bloques que hubiera en el editor. La segunda función se ejecuta cuando se hace click en el botón `Block-Text a Python`, y en este caso no es necesario que se haya ejecutado anteriormente el generador Block-Text, pues toma como entrada directamente el código que hay escrito en la interfaz Block-Text. El traductor Python devuelve todo el código generado en forma de *string*.

Tomando como referencia el código Block-Text mostrado en la Figura 5.31, en la Figura 5.34 aparece el código Python generado a partir de él, orientado para su ejecución en el robot EBO.

```
def triste():
        function.expressSadness()
end

main:
        while function.is_front_obstacle(0) == False:
                function.move_straight()
                if function.is_obstacle_free(0) == False:
                        function.stop_bot()
                        triste()
                end
        end
end
```

```
#EXECUTION: python code.py
from __future__ import print_function, absolute_import
import sys, os, time, traceback
sys.path.insert(0, os.path.join(os.getenv('HOME'),
".learnblock", "clients"))
from EBO import Robot
import signal
import sys

usedFunctions = ['expressSadness', 'is_front_obstacle',
'move_straight', 'is_obstacle_free', 'stop_bot']

try:
  robot = Robot(availableFunctions = usedFunctions)
except Exception as e:
  print("Problems creating a robot instance")
  traceback.print_exc()
  raise(e)


time_global_start = time.time()
def elapsedTime(umbral):
  global time_global_start
  time_global = time.time()-time_global_start
  return time_global > umbral

def signal_handler(sig, frame):
  robot.stop()
  sys.exit(0)

signal.signal(signal.SIGTERM, signal_handler)
signal.signal(signal.SIGINT, signal_handler)

def triste():
  robot.expressSadness()

while robot.is_front_obstacle(0) == False:
  robot.move_straight()

  if robot.is_obstacle_free(0) == False:
    robot.stop_bot()
    triste()

robot.stop()
sys.exit(0)
```

Figura 5.34: Código Python generado

Capítulo 6

Resultados

En esta sección se presenta una serie de ejemplos de programas sencillos que se pueden construir en LearnBlockWeb. Cada uno de ellos se mostrará en sus tres versiones distintas, correspondientes a los tres lenguajes de programación que ofrece la web.

Los dos primeros ejemplos muestran funciones sencillas que no están orientadas a ser ejecutadas en ningún robot, sino a probar la correcta traducción del lenguaje de bloques al lenguaje textual y a Python. Con el resto sí se pretende que el código resultante se ejecute en alguno de los robots disponibles.

- Ejemplo 1: Función que compara el valor de dos variables. Sin robot especificado (opción por defecto: Client).

LearnBlock

Block-Text

```
numero1 = None
numero2 = None
resultado = None


main:
        numero1 = 1758
        numero2 = 5551
        if numero1 == numero2:
                resultado = "Son iguales"
        elif numero1 < numero2:
                resultado = "Es menor"
        else:
                resultado = "Es mayor"
        end
end
```

Python

```
#EXECUTION: python code.py
from __future__ import print_function, absolute_import
import sys, os, time, traceback
sys.path.insert(0, os.path.join(os.getenv('HOME'), ".learnblock", "clients"))
from Client import Robot
import signal
import sys

usedFunctions = []

try:
    robot = Robot(availableFunctions = usedFunctions)
except Exception as e:
    print("Problems creating a robot instance")
    traceback.print_exc()
    raise(e)


time_global_start = time.time()
def elapsedTime(umbral):
    global time_global_start
    time_global = time.time()-time_global_start
    return time_global > umbral

def signal_handler(sig, frame):
    robot.stop()
    sys.exit(0)
signal.signal(signal.SIGTERM, signal_handler)
signal.signal(signal.SIGINT, signal_handler)

numero1 = None
numero2 = None
resultado = None
numero1 = 1758
numero2 = 5551

if numero1 == numero2:
    resultado = "Son iguales"
elif numero1 < numero2:
    resultado = "Es menor"
else:
    resultado = "Es mayor"

robot.stop()
```

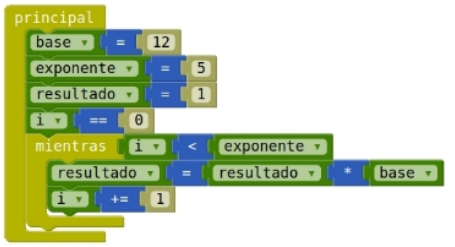
Figura 6.1: Ejemplo 1

Una función sencilla que determina si el valor asignado a la primera variable es mayor, igual o menor al valor que tiene la segunda variable. El resultado de esa comparación se almacena como una cadena de texto en una tercera variable.

Con este ejemplo se muestra el uso del bloque principal y de los bloques condicionales, así como los operadores de comparaciones y la definición y utilización de variables creadas por el usuario.

- Ejemplo 2: Función que calcula la potencia de un número. Sin robot especificado (opción por defecto: Client).

LearnBlock



Block-Text

```
base = None
exponente = None
resultado = None
i = None


main:
        base = 12
        exponente = 5
        resultado = 1
        i == 0
        while i < exponente:
                resultado = resultado * base
                i += 1
        end
end
```

Python

```
#EXECUTION: python code.py
from __future__ import print_function, absolute_import
import sys, os, time, traceback
sys.path.insert(0, os.path.join(os.getenv('HOME'), ".learnblock", "clients"))
from Client import Robot
import signal
import sys

usedFunctions = []

try:
    robot = Robot(availableFunctions = usedFunctions)
except Exception as e:
    print("Problems creating a robot instance")
    traceback.print_exc()
    raise(e)


time_global_start = time.time()
def elapsedTime(umbral):
    global time_global_start
    time_global = time.time()-time_global_start
    return time_global > umbral

def signal_handler(sig, frame):
    robot.stop()
    sys.exit(0)
signal.signal(signal.SIGTERM, signal_handler)
signal.signal(signal.SIGINT, signal_handler)

base = None
exponente = None
resultado = None
i = None
base = 12
exponente = 5
resultado = 1
i = 0

while i < exponente:
    resultado = resultado * 1
    i += 1

robot.stop()
sys.exit(0)
```

Figura 6.2: Ejemplo 2

Un procedimiento que calcula la potencia de un número a partir de su base y su exponente. El resultado de dicho cálculo se almacena en una variable resultado.

Con este ejemplo, al igual que con el anterior, se muestra el uso de las variables creadas por el usuario. Además, se utiliza la estructura de control de bucles, especificando cómo se emplearía junto con un sistema de iteraciones.

- Ejemplo 3: El robot se mueve a lo largo de una línea roja. Código generado para el robot EBO simulado.

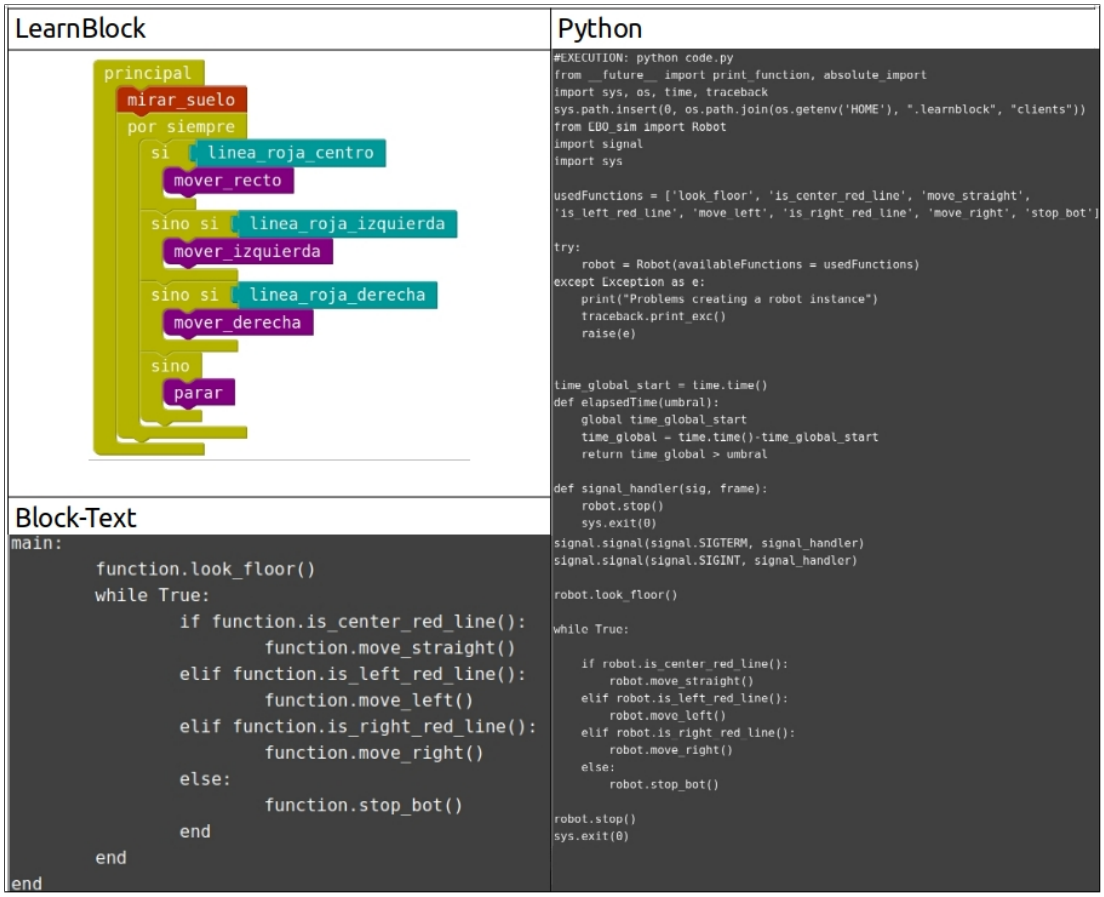


Figura 6.3: Ejemplo 3

La idea es lograr que el robot EBO se mueva siguiendo una línea roja pintada en una superficie. Para ello comprueba de manera continua en primer lugar si la línea roja está delante de él; en caso de que no haya, comprueba a su derecha y después a su izquierda. Si no hay ninguna línea, se detiene.

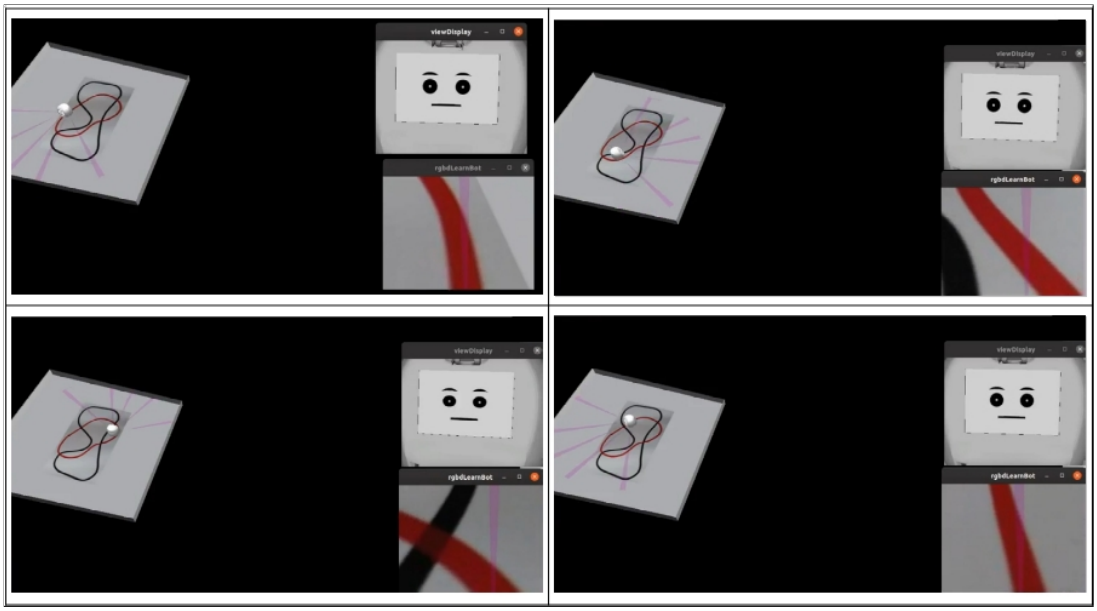


Figura 6.4: Ejecución del ejemplo 3

Vídeo de la ejecución completa: `https://drive.google.com/file/d/1oFreW8QNLIZxfRhLZzaTQ1FM8FZKRhR-/view?usp=sharing`

- Ejemplo 4: El robot repite diez veces el siguiente procedimiento: mirar hacia arriba y expresar sorpresa, mirar al frente y expresar alegría, mirar hacia abajo y expresar tristeza. Cada vez que realiza una de estas acciones, espera durante un segundo antes de llevar a cabo la siguiente. Código generado para el robot Cozmo.

LearnBlock

Block-Text

```
i = None


main:
        i = 10
        while i > 0:
                function.look_up()
                function.expressSurprise()
                function.sleep(1)
                function.look_front()
                function.expressJoy()
                function.sleep(1)
                function.look_floor()
                function.expressSadness()
                function.sleep(1)
                i -= 1
        end
end
```

Python

```
#EXECUTION: python code.py
from __future__ import print_function, absolute_import
import sys, os, time, traceback
sys.path.insert(0, os.path.join(os.getenv('HOME'), ".learnblock", "clients"))
from Cozmo import Robot
import signal
import sys

usedFunctions = ['look_up', 'expressSurprise', 'sleep', 'look_front',
'expressJoy', 'look_floor', 'expressSadness']

try:
    robot = Robot(availableFunctions = usedFunctions)
except Exception as e:
    print("Problems creating a robot instance")
    traceback.print_exc()
    raise(e)


time_global_start = time.time()
def elapsedTime(umbral):
    global time_global_start
    time_global = time.time()-time_global_start
    return time_global > umbral

def signal_handler(sig, frame):
    robot.stop()
    sys.exit(0)
signal.signal(signal.SIGTERM, signal_handler)
signal.signal(signal.SIGINT, signal_handler)

i = None
i = 10

while i > 0:
    robot.look_up()
    robot.expressSurprise()
    robot.sleep(1)
    robot.look_front()
    robot.expressJoy()
    robot.sleep(1)
    robot.look_floor()
    robot.expressSadness()
    robot.sleep(1)
    i -= 1

robot.stop()
sys.exit(0)
```

Figura 6.5: Ejemplo 4

En este ejemplo se utilizan funciones motoras del propio cuerpo del robot y funciones con las que éste expresa emociones. También se muestra cómo se introducen parámetros en las funciones que lo requieren, en este caso es un bloque con valor numérico.

- Ejemplo 5: El robot mira continuamente a su alrededor. Cada vez que gira sobre sí mismo, comprueba si hay alguien con expresión neutral y de enfado y, de ser así, repite esa emoción antes de expresar alegría. Código generado para el robot Cozmo.

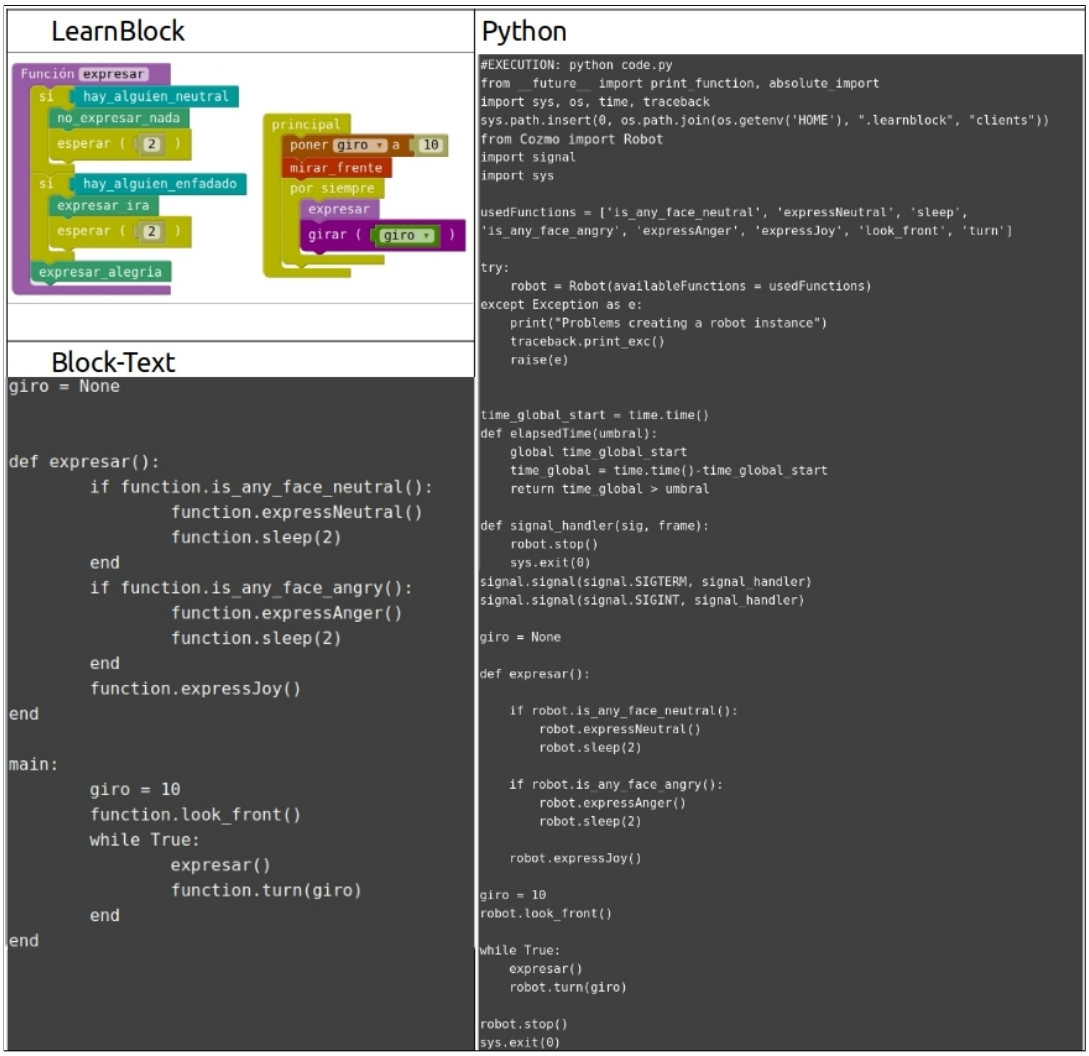


Figura 6.6: Ejemplo 5

En este ejemplo se pretende que el robot perciba si hay alguien en su entorno que esté expresando alguna emoción y, de ser así, la reproduzca. Se muestra también el uso de funciones creadas por el usuario y su posterior llamada en el bloque principal, así como un ejemplo de cómo introducir una variable como parámetro en una función.

- Ejemplo 6: El robot avanza hacia delante sin parar. Mientras no haya ningún obstáculo, avanza. Cando percibe algún obstáculo, se detiene y gira 90º hacia su derecha. Código generado para el robot EV3 simulado.

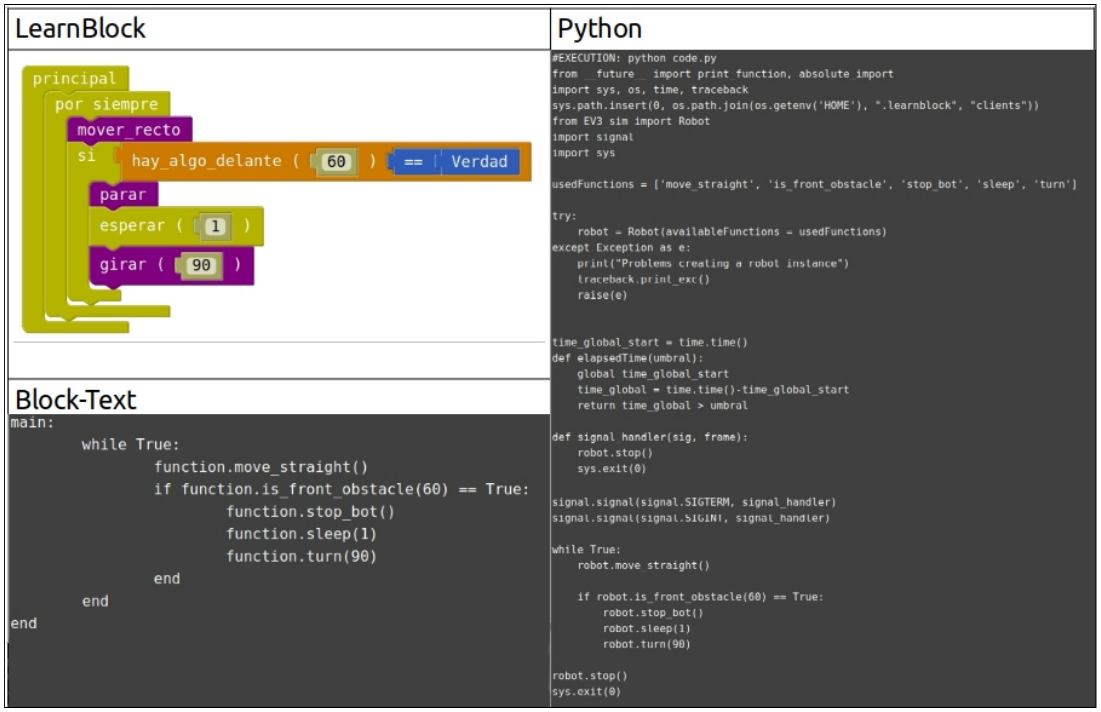


Figura 6.7: Ejemplo 6

Con este código, el robot se moverá a lo largo de una superficie, detectando en todo momento si hay algún obstáculo a una determinada distancia. Continúa o se detiene y gira en función de si detecta algún obstáculo cerca o no. Es un ejemplo que se encarga de mostrar funciones de movimiento sobre una base y funciones de detección de objetos en el entorno del robot.

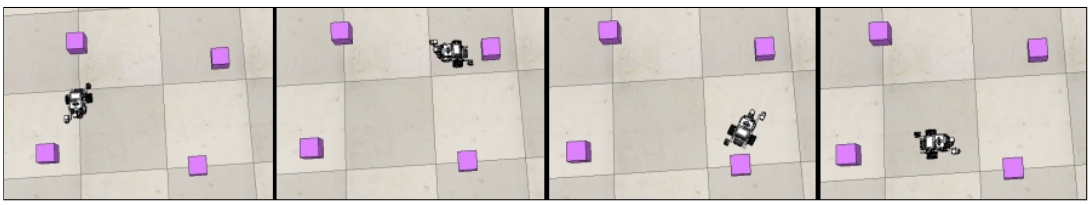


Figura 6.8: Ejecución del ejemplo 6

Vídeo de la ejecución completa: `https://drive.google.com/file/d/1vo-0ib5Adtz8nCzJXrqYYKxS2ZBsyd1p/view?usp=sharing`

Capítulo 7

Conclusiones y trabajos futuros

Uno de los principales propósitos en esta época actual a la hora de desarrollar proyectos o herramientas, independientemente del campo que se abarque, es buscar la manera de que puedan ser utilizadas por el mayor número de personas posible dentro de su público objetivo. En el ámbito de las tecnologías, cada vez son más las herramientas que ofrecen una versión online con características similares a la original, e incluso en ocasiones se opta por desarrollar directamente esas versiones sin crear previa ni posteriormente una alternativa local.

Internet se está expandiendo continuamente y cada vez a mayor escala. El número de personas que tiene acceso a internet en su entorno doméstico y de trabajo aumenta a diario, con lo que disponen por completo de todo lo que está almacenado y desarrollado en la nube y pueden acceder a ello con poco más que unos clicks. Esta gran comodidad y completa disponibilidad que ofrece internet es algo que hace que el usuario esté más dispuesto a utilizar herramientas online, sobre todo si se trata de un primer contacto con ellas o si éstas ofrecen una funcionalidad de la que se tenga que hacer uso de manera esporádica.

Este es el continua planteamiento detrás de LearnBlockWeb. La versión web de LearnBlock proporciona esa accesibilidad, disponibilidad y comodidad de la que carece la aplicación de escritorio. De esta manera, se logra que la idea de LearnBlock tenga un alcance superior y logre llegar a un mayor número de personas.

LearnBlockWeb facilita la utilización de la herramienta al no necesitar ningún tipo de instalación, requerimientos físicos o conocimientos previos de software, y consigue que su uso no esté tan limitado como lo está la versión local.

No obstante, la mayor limitación de la página web aparece en la fase de la ejecución del código. Mientras que la aplicación de escritorio permite integrar diferentes robots con los que posteriormente puede establecer conexiones, desde la aplicación web no existe esta posibilidad por limitaciones de software: es necesario tener instalados de forma local en el equipo una serie de programas y paquetes para lograrlo.

Teniendo esto en cuenta, desde la página web se le ofrece al usuario un paquete que se puede descargar e instalar en el ordenador. Con él, es posible crear esas conexiones con los robots disponibles y el código Python generado en LearnBlockWeb puede ser ejecutado por ellos.

Ya que LearnBlockWeb no abarca la funcionalidad completa de LearnBlock, sino parte de ella, es un proyecto al que se le pueden añadir nuevas funcionalidades, mejoras y detalles en un futuro para que exista una mayor conexión y similitud con la versión de escritorio. A continuación se lista un conjunto de propuestas que se podrían llevar a cabo para mejorar el proyecto.

- LearnBlock ofrece dos paradigmas de programación: Estructural y Orientado a Eventos. LearnBlockWeb solo incluye el primero. Una mejora inmediata podría ser añadir la posibilidad de crear programas orientados a eventos en la versión web.

- LearnBlock permite que el usuario implemente la definición y el funcionamiento de nuevos bloques que luego pueden ser ejecutados por los robots conectados. Esta funcionalidad no existe en LearnBlockWeb, y añadirla sería una manera de completar más la versión online de la herramienta.

- Crear una versión Android de LearnBlock, o hacer que LearnBlockWeb sea completamente funcional desde los navegadores instalados en dispositivos móviles. Esta es otra manera de lograr que LearnBlock aumente su alcance.

Bibliografía

[1] LearnBlock: A Robot-Agnostic Educational Programming Tool,
https://ieeexplore.ieee.org/document/8986589

[2] Front-End y Back-End,
https://es.wikipedia.org/wiki/Front_end_y_back_end

[3] HyperText Markup Language,
https://developer.mozilla.org/es/docs/Web/HTML

[4] HyperText Markup Language 5,
https://developer.mozilla.org/es/docs/HTML/HTML5

[5] Cascading Style Sheets,
https://www.w3schools.com/css/css_intro.asp

[6] JavaScript,
https://developer.mozilla.org/es/docs/Web/JavaScript

[7] AngularJS,
https://angular.io/

[8] React,
https://es.reactjs.org/

[9] Vue,
https://vuejs.org/v2/guide/

[10] Meteor,

`https://docs.meteor.com`

[11] Node,

`https://ifgeekthen.everis.com/es/que-es-node-js-y-primeros-pasos`

[12] Ember,

`https://ifgeekthen.everis.com/es/consejos-para-aprender-programar-con-ember-js`

[13] JQuery,

`https://jquery.com/`

[14] AJAX,

`https://www.w3schools.com/xml/ajax_intro.asp`

[15] Bootstrap,

`https://getbootstrap.com/docs/4.5/getting-started/introduction/`

[16] Foundation,

`https://get.foundation/`

[17] HTML5 Boilerplate,

`https://html5boilerplate.com/`, `https://www.htmlcinco.com/html5-boilerplate/`

[18] Blockly,

`https://developers.google.com/blockly`, `https://developers.google.com/blockly/guides/overview`

[19] App Inventor,

`https://appinventor.mit.edu/`

[20] Code Studio,

`https://studio.code.org/courses`

[21] Microsoft MakeCode,

`https://www.microsoft.com/en-us/makecode`

[22] Micro:Bit,

`https://makecode.microbit.org/`

[23] OzoBlockly,

`https://ozobot.com/ozoblockly`

[24] CodeBug,

`http://www.codebug.org.uk/`

[25] Python,

`https://es.wikipedia.org/wiki/Python`

[26] Django,

`https://www.djangoproject.com/`

[27] Flask,

`https://flask.palletsprojects.com/en/1.1.x/`

[28] Pyramid,

`https://trypyramid.com/`

[29] Web2Py,

`http://www.web2py.com/`

[30] CherryPy,

`https://cherrypy.org/`

[31] Bottle,

`https://bottlepy.org/docs/dev/`