



Trabajo de Fin de Grado

Grado en Ingeniería Informática

Pensamiento computacional en estudios preuniversitarios

*Computational thinking in pre-university studies*

Rafael Herrero Álvarez

La Laguna, 5 de junio de 2017

D. Coromoto Antonio León Hernández, con N.I.F. 78.605.216-W profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. Carlos Segura González, con N.I.F. 78.404.244-S profesor Investigador Asociado tipo C adscrito al Departamento de Ciencias de la Computación del Centro de Investigación Matemática (CIMAT) de México, como cotutor

**C E R T I F I C A (N)**

Que la presente memoria titulada:

*“Pensamiento computacional en estudios preuniversitarios”*

ha sido realizada bajo su dirección por D. Rafael Herrero Álvarez, con N.I.F. 54.063.043-W.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos ﬁrman la presente en La Laguna a 5 de junio de 2017.

Agradecimientos

A mi tutora Coromoto, por brindarme esta oportunidad, ser un apoyo incondicional para la realización de este trabajo y por resolver todas las dudas que me surgían.

A Daniel Ramos por acompañarme durante estos 4 años de grado en los buenos y malos momentos a pesar de mi locura, mis cambios de humor y mi forma de ser. Simplemente, gracias.

A mis compañeros de grado por las conversaciones, las risas y los juegos de cartas que hicieron inolvidable la experiencia universitaria.

Al Colegio Nuryana por permitirme probar la aplicación desarrollada y comprobar su validez en un entorno real.

Licencia

C:\Users\Usuario\Desktop\memoriaTFG_1415_LaTeX\tfg\images\by-nc_88x31.png

© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial 4.0 Internacional.

**Resumen**

*El objetivo de este trabajo ha sido .... bla, bla, bla bla, bla, bla bla, bla, bla*

*La competencia [E6], que figura en la guía docente, indica que en la memoria del trabajo se ha de incluir: antecedentes, problemática o estado del arte, objetivos, fases y desarrollo del proyecto, conclusiones, y líneas futuras.*

*El documento de memoria debe tener un máximo de 50 páginas.*

*No se deben dejar páginas en blanco al comenzar un capítulo, ya que el documento no está pensado para sea impreso sino visionado con un lector de PDFs.*

*También es recomendable márgenes pequeños ya que, al firmar digitalmente por la Sede, se coloca un marco alrededor del texto original.*

*El tipo de letra base ha de ser de 14ptos.*

**Palabras clave:** pensamiento computacional, estudios preuniversitarios, nutrición, programación visual, tecnologías web.

**Abstract**

Here should be the abstract of the work in a foreign language.

***Keywords:*** computational thinking, pre-university studies, nutrition, Scratch, Blockly, visual programming, block programming, Comilona.

Índice General

[Capítulo 1. Introducción 1](#_Toc484173682)

[1.1 Antecedentes y estado actual del tema 2](#_Toc484173683)

[1.2 Objetivos 5](#_Toc484173684)

[1.3 Metodología 5](#_Toc484173685)

[1.4 Organización de la memoria 6](#_Toc484173686)

[Capítulo 2. Herramientas y tecnologías 8](#_Toc484173687)

[2.1 Electron 8](#_Toc484173688)

[2.2 Blockly 9](#_Toc484173689)

[2.3 Pure 13](#_Toc484173690)

[2.4 Librerías Javascript 14](#_Toc484173691)

[2.4.1 jQuery 14](#_Toc484173692)

[2.4.2 JS-Interpreter 15](#_Toc484173693)

[2.4.3 Chance 16](#_Toc484173694)

[Capítulo 3. Modo de uso 17](#_Toc484173695)

[3.1 Ventana principal 17](#_Toc484173696)

[3.2 Ventana de información 19](#_Toc484173697)

[3.3 Columna izquierda 20](#_Toc484173698)

[3.4 Columna derecha 22](#_Toc484173699)

[3.5 Barra inferior 23](#_Toc484173700)

[3.6 Niveles propuestos 24](#_Toc484173701)

[3.6.1 Nivel 1 25](#_Toc484173702)

[3.6.2 Nivel 2 26](#_Toc484173703)

[3.6.3 Nivel 3 28](#_Toc484173704)

[Capítulo 4. Desarrollo 31](#_Toc484173705)

[Capítulo 5. Verificación, pruebas, resultados y discusión 32](#_Toc484173706)

[Capítulo 6. Conclusiones y líneas futuras 33](#_Toc484173707)

[Capítulo 7. Summary and Conclusions 34](#_Toc484173708)

[7.1 First Section 34](#_Toc484173709)

[Capítulo 8. Presupuesto 35](#_Toc484173710)

[8.1 Sección Uno 35](#_Toc484173711)

[Bibliografía 36](#_Toc484173712)

Índice de figuras

[Figura 1.1.1: Logo de Scratch 2](#_Toc484173713)

[Figura 1.1.2: Ejemplo de mostrar por pantalla 'Hola mundo' en Scratch 3](#_Toc484173714)

[Figura 1.1.3: Ventana principal de Acomola 4](#_Toc484173715)

[Figura 2.1.1: Logo de Electron 9](#_Toc484173716)

[Figura 2.2.1: Interfaz de usuario de Blockly 11](#_Toc484173717)

[Figura 2.2.2: Ejemplo de bloque generado con Blockly Developer Tools 12](#_Toc484173718)

[Figura 2.3.1: Logo de Pure 13](#_Toc484173719)

[Figura 2.4.1: Logo de jQuery 14](#_Toc484173720)

[Figura 2.4.2: Logo de Chance 16](#_Toc484173721)

[Figura 3.1.1: Ventana principal de Comilona 17](#_Toc484173722)

[Figura 3.1.2: Mensaje en Comilona si se usa una resolución menor a la permitida 18](#_Toc484173723)

[Figura 3.2.1: Ejemplo de ventana informativa para uno de los ejercicios 19](#_Toc484173724)

[Figura 3.3.1: Ejemplo de la columna izquierda en Comilona 21](#_Toc484173725)

[Figura 3.4.1: Ejemplo de la columna derecha en Comilona 23](#_Toc484173726)

[Figura 3.5.1: Ejemplo de barra inferior con Comilona 24](#_Toc484173727)

[Figura 3.6.1: Captura del ejercicio 4 del nivel 1 26](#_Toc484173728)

[Figura 3.6.2: Bloque diseñado para indicar un bucle 26](#_Toc484173729)

[Figura 3.6.3: Captura del ejercicio 3 del nivel 2 28](#_Toc484173730)

[Figura 3.6.4: Ejemplo de bloque de función 28](#_Toc484173731)

Índice de tablas

[Tabla 1.1‑1: Iniciativas para la promoción del pensamiento computacional 3](#_Toc484173732)

[Tabla 8.1‑1. Tabla resumen de los Tipos. 35](#_Toc484173733)

## Introducción

El concepto de pensamiento computacional, usando las palabras de J. Wing [1], se define como una forma de pensar en la que se afronta el análisis, la formulación y la resolución de problemas utilizando un enfoque analítico y algorítmico. Además, Wing destaca que este no debe ser exclusivo de los informáticos, ya que es una habilidad fundamental para todos. Es por ello que el pensamiento computacional debería considerarse una habilidad analítica más, como lo son la lectura, la escritura o la aritmética. Tras este trabajo se han publicado muchos otros que tratan sobre el uso del pensamiento computacional para resolver distintos problemas [2].

En los estudios preuniversitarios, en nuestro caso en los cursos con niños de entre 8 a 12 años, se enseña muchas veces como utilizar un procesador de texto, como navegar por la web o como elaborar una presentación con diapositivas. Sin embargo, esto no les lleva a realizar un análisis profundo que les permita pensar de una manera creativa y crítica [3].

Por ello, se quiere desarrollar una aplicación, a la que llamaremos Comilona, que utilizando diferentes ejemplos de una nutrición saludable nos permita introducir conceptos sobre pensamiento computacional y de programación en estudios preuniversitarios, usando un lenguaje de programación visual, en este caso, uno de programación por bloques.

Si analizamos el estado actual de la informática en los estudios preuniversitarios, vemos que todo se basa en la alfabetización digital, pero no en indagar en el pensamiento computacional. Por ello, surge la idea de la creación de este tipo de juego con el que intentamos expandir este campo a edades más tempranas, debido a la inexistencia hasta el momento de juegos que mezclasen pensamiento computacional y nutrición.

### Antecedentes y estado actual del tema

Al analizar el mundo del pensamiento computacional, vemos que los estudios que existen son muy recientes, muchos de ellos de la última década, destacando algunos de la década pasada como el de J. Wing [1] del año 2006, cuando el mundo de la computación tal y como la conocemos existe desde el año 1953 con el primer grado en ciencias de la computación por la Universidad de Cambridge [4].

El potencial del pensamiento computacional en los colegios es muy alto, sin embargo, es muy complicado enseñar en estas edades las herramientas tradicionales de codificación y programación, ya que suelen ser más un obstáculo que una ventaja. Es por ello que cuando queremos aplicar el pensamiento computacional en estudios preuniversitarios, podemos ver que las principales herramientas utilizan lenguajes de programación visuales, la mayoría basados en bloques. La más utilizada es Scratch [5], un lenguaje de programación visual desarrollado por el Instituto de Tecnología de Massachussetts (MIT) en el año 2002.

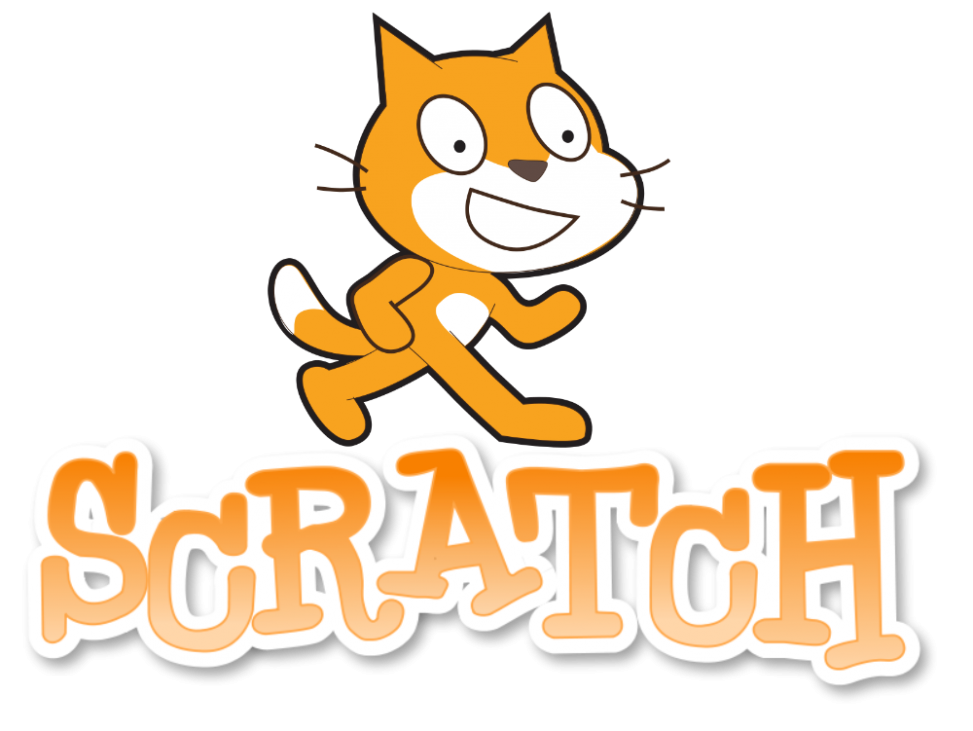


Figura .: Logo de Scratch

Esta herramienta pretende que los estudiantes, principal público objetivo, desarrollen distintas habilidades mentales utilizando la programación, pero sin mostrar en ningún momento código, aplicando una filosofía de bloques, en el que tenemos que ir uniéndolos para formar un puzle con el reto propuesto. En figura 1.1.2 podemos ver cómo sería el programa para mostrar ‘Hola mundo’ por pantalla en el lenguaje Scratch.

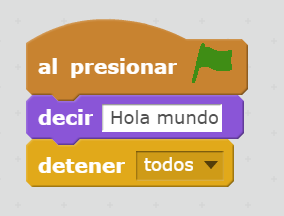


Figura .: Ejemplo de mostrar por pantalla 'Hola mundo' en Scratch

Desde que apareciese Scratch en el año 2002, numerosas empresas y asociaciones han creado distintos ejercicios para promover el pensamiento computacional, tanto entre la población en general como en el ámbito educativo, siendo muchas de ellas juegos con bloques. Una de las más conocidas es la Hora de Código [6], promovida por la organización estadounidense Code [7]. Esta iniciativa pretende que la gente realice un ejercicio de computación durante una hora, de manera que su objetivo no es convertir a nadie en un experto programador, si no en demostrar que las Ciencias de la Computación pueden llegar a ser divertidas, además de accesibles para todos los públicos. En la tabla 1.1.1 se pueden observar algunas de ellas.

|  |  |
| --- | --- |
| Iniciativa | Página web |
| Hora del código | https://hourofcode.com/ |
| Made w/Code | https://www.madewithcode.com |
| Code Club | https://www.codeclub.org.uk |
| CoderDojo | https://coderdojo.com/ |
| Code Week | http://codeweek.eu/ |
| Google CS First | https://www.cs-first.com/en/home |

Tabla ‑: Iniciativas para la promoción del pensamiento computacional

Como comentamos al comienzo de este capítulo, otro de los objetivos es el de promover hábitos de alimentación saludables. Para ello se llevó a cabo una revisión sobre juegos e iniciativas que trabajasen ambos conceptos. Sin embargo, no se encontró ninguno, únicamente algunos juegos que nada tienen que ver con el pensamiento computacional, pero que intentan promover modos de vida saludables.

Los juegos analizados han sido una adaptación de Jeopardy [8], otra del famoso juego Scattergories [9] y una plataforma del Gobierno de Canarias llamada Acomola [10]. En el caso del Nutrition Jeopardy, se basa en elegir un tablero que contiene una pista en forma de respuesta, a la cual los concursantes contestan una pregunta. En cuanto al Nutritional Scattergories, también sigue la idea del juego original, donde alguien tiene que decir una letra y otra persona elige una categoría, siendo el fin del juego que se digan o escriban la mayor cantidad de palabras posibles en un tiempo determinado, salvo que ahora todas las categorías y respuestas están basadas en nutrición. Sin embargo, el objetivo de Acomola es totalmente diferente, ya que únicamente tenemos que colocar distintos alimentos por categorías en distintas cajas, ganando más puntos según la dificultad de las categorías y si la respuesta es la correcta. La interfaz de Acomola se puede observar en la figura 1.1.3.



Figura .: Ventana principal de Acomola

### Objetivos

Los objetivos de este trabajo de fin de grado se pueden enmarcar en los siguientes apartados:

* Realizar un análisis del estado de los temas del trabajo, sobre el pensamiento computacional y sobre la utilización de la nutrición en juegos educativos.
* Estudiar las herramientas ya existentes relacionadas con los lenguajes de programación visuales.
* Desarrollar una aplicación que nos permita enseñar conceptos sobre pensamiento computacional y de programación, acercándolos de una manera visual y utilizando el tema de la nutrición como base.

### Metodología

La metodología de desarrollo ha seguido la siguiente planificación establecida al comienzo del trabajo:

Tarea 0.- Coordinación.

La primera de las tareas se basa en la coordinación del TFG, fijando reuniones periódicas con el tutor. Tratamos de reunirnos, como mínimo una vez al mes, siendo todas estas los últimos miércoles de cada mes. Sin embargo, intentaremos llevar un control del trabajo de manera semanal.

Tarea 1.- Revisión bibliográfica.

Una vez analizados los antecedentes del tema que vamos a tratar, se llevará a cabo una revisión de los documentos elegidos para determinar la validez y el valor de los mismos sobre el tema.

Tarea 2.- Diseño del prototipo de la herramienta.

Se busca que esta herramienta sea de tipo web, por lo que se da preferencia a los lenguajes de la misma, HTML, CSS y JavaScript. Toda la información sobre los alimentos, recetas, platos, etc., estará almacenada en un fichero, pudiendo ampliarse a una base de datos en un futuro. Todo este trabajo se sincronizará con un repositorio git en GitHub.

Tarea 3.- Implementación de la herramienta.

Actualmente, Google ha desarrollado una herramienta que será la piedra angular de este proyecto, Blockly. Se trata de una librería escrita en JavaScript y que nos permite trabajar la programación visual utilizando bloques. Todo ello se empaquetará como una webapp gracias a Electron, un framework de GitHub que permite construir aplicaciones de escritorio multiplataforma utilizando tecnologías web como Chromium y Node.js. Para generar la memoria y presentación, se barajó la posibilidad de utilizar LaTeX y BibTeX, herramientas que finalmente no se han utilizado, optando por soluciones ofimáticas estándar.

Utilizaremos estas soluciones puesto que se encuentran actualizadas y son ampliamente utilizadas, por lo que los recursos disponibles son mayores y a la vez conseguimos crear una herramienta versátil.

Tarea 4.- Validación y resultados computacionales (pruebas).

Realizaremos pruebas de la herramienta en cada una de las plataformas, así como con el público al que va dirigido este proyecto, de manera que podamos obtener las opiniones de los mismos.

Tarea 5.- Difusión de los resultados.

Para ilustrar los resultados del trabajo se elaborará una memoria y una presentación con los aspectos generales del mismo. Así mismo se dotará con la licencia GNU GPL, de manera que sea totalmente libre, así como las ampliaciones o modificaciones que hagan los usuarios.

Para esta última tarea ha sido aceptada en el V Congreso Internacional de Videojuegos y Educación, celebrado en junio de 2017 en el Puerto de la Cruz, una comunicación sobre el trabajo realizado con la aplicación desarrollada.

### Organización de la memoria

El presente documento se divide en cuatro apartados o capítulos principales de la siguiente manera:

* El capítulo II describe las herramientas que se utilizaron para desarrollar la aplicación.
* En el capítulo III podremos comprobar cómo funciona el programa y algunos de sus ejercicios.
* El capítulo IV trata exclusivamente del desarrollo del juego. Por ejemplo, que lenguajes se usaron, como se han definido los ejercicios y niveles, etc.
* El capítulo V contiene el resultado de la realización de unas pruebas sobre un grupo de niños, así como una discusión sobre los mismos.
* El capítulo VI abarca las conclusiones del trabajo, además de especificar cuáles serán las líneas de actuación futuras.
* El capítulo VII se presenta el presupuesto y la factura final que ha supuesto la elaboración de este proyecto.

## Herramientas y tecnologías

En el capítulo anterior se ha comentado la idea de desarrollar una aplicación que mezclase los contenidos relacionados con el pensamiento computacional, la programación y la nutrición. Para llevar a cabo dicho proyecto, se han utilizado las siguientes herramientas y tecnologías.

### Electron

Electron [11] se define como un framework que nos permite crear aplicaciones de escritorio independientes y multiplataforma, utilizando tecnologías que hasta ahora estaban reservadas a la web, como son HTML, CSS y Javascript. Electron está basado en el lenguaje Node.js [12], que a su vez es una adaptación de Javascript, y Chromium [13], el código fuente del popular navegador Chrome. Gracias a la utilización de Node.js para su base, todos los paquetes y programas escritos en este lenguaje se pueden utilizar en nuestra aplicación, lo que aumenta aún más las posibilidades y la versatilidad de la aplicación desarrollada.

Como veremos más adelante, en los capítulos IV, V y VI, evitaremos el uso de Node.js, en favor de tecnologías puramente web, y se probará el uso de la aplicación ya desarrollada sobre los sistemas operativos Windows, Mac OS y Ubuntu.

Electron nos permite integración nativa con el gestor de ventanas de cada sistema operativo, lo que nos da acceso, por ejemplo, en el caso de Windows, a utilizar y personalizar el menú superior, donde habitualmente se encuentran las funciones de inicio, archivo, ventana o ayuda. Gracias a esto, podremos ubicar nuestras propias funcionalidades en dicho menú, como por ejemplo, recargar o abandonar el juego.

Además, gracias a que detrás del desarrollo se encuentra una empresa como GitHub, todo lo referente con el desarrollo de Electron está basado en la filosofía del código abierto, donde cualquiera puede acceder al código fuente, por lo que podemos analizarlo si nos hiciese falta.

La primera versión existe desde el año 2013, sin embargo, la primera versión estable (1.0.0) fue liberada en mayo de 2016. Actualmente está disponible en varios idiomas, entre los que se encuentran el brasileño, chino, coreano, español, inglés o turco. Sobre él se han basado varias aplicaciones como los editores de texto y código Atom y Visual Studio Code, o la aplicación de escritorio de Wordpress.



Figura .: Logo de Electron

### Blockly

Como se comentó en el capítulo I, la tecnología principal en la que se basará nuestro desarrollo es Blockly [14], una librería escrita en Javascript y que permite, sobre el lado del cliente, crear distintos lenguajes visuales de programación basada en bloques. Su desarrollo comenzó en el año 2011 y la primera publicación data del 2012.

Cuando hablamos de Blockly lo primero que recordamos es Scratch, la tecnología mencionada al principio de esta memoria, y en cuya filosofía se basa la herramienta de Google. Sin embargo, son bien distintas, puesto que Scratch en si es un lenguaje de programación visual, mientras que Blockly es una herramienta que nos permite crear nuestros propios lenguajes. Además, al ser Blockly una tecnología más moderna, abandona el uso de Flash por Javascript y el formato SVG para representar los bloques generados [15], lo que nos permite crear una aplicación multiplataforma sin problemas.

Esta herramienta está desarrollada principalmente en Javascript, ya que su diseño original estaba pensado para funcionar sobre navegadores web. Sin embargo, también se ha implementado para los sistemas operativos móviles Android e iOS, a pesar de que algunas características de la versión web no están disponibles, como una de la que hacemos uso, que nos permite iluminar los bloques según definamos, en nuestro caso, para mostrar al usuario cual es el paso que se está ejecutando en ese momento.

Como se ha comentado, Blockly es una herramienta para crear lenguajes de programación visual, pero no es un lenguaje de programación, ya que no se puede ejecutar un programa de Blockly, sino que es capaz de traducir el código contenido en los bloques a lenguajes que si son de programación, como Javascript, PHP, Python o Dart.

Gracias a que ha sido desarrollado por Google, se consigue una integración con Google App Engine [16], de manera que si nuestra aplicación está alojada en dicha plataforma, también podemos guardar, cargar, compartir y publicar los programas que generamos bajo Blockly. En nuestro caso, como buscamos que la aplicación sea de escritorio y totalmente independiente, no hemos explorado esta vía, aunque para un desarrollo futuro sobre servidores, esta es una solución que no se descarta.

Lo que encontramos a la hora de utilizarlo, es una interfaz de usuario dividida en dos partes, en la que veremos los bloques disponibles a la izquierda ordenados por categorías, en lo que se llama caja de herramientas, y un área de trabajo a la derecha en la que colocar la secuencia de bloques. Incluye muchas opciones de personalización, como por ejemplo, decidir si se muestra o no la papelera para eliminar los bloques, o una herramienta que es capaz de escalar la interfaz de manera que podamos aumentar o disminuir la cantidad de zoom realizadas sobre los bloques. Un ejemplo de esta estructura lo podemos encontrar en la figura 2.2.1.

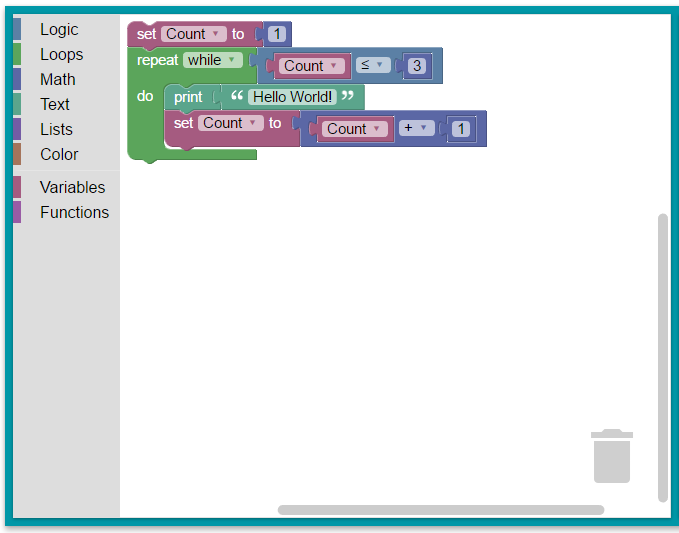


Figura .: Interfaz de usuario de Blockly

Hasta ahora se ha mencionado en varias ocasiones que Blockly nos permite desarrollar nuestro propio lenguaje de programación visual, y para ello necesitamos contar con nuestros propios bloques. Cada bloque, sea personalizado o de los que ya integra Blockly, está formado por dos partes, un apartado para la definición del bloque, como se ve, que aspecto general tiene, que textos contiene, de qué color es, si tiene o no conexiones con otros bloques y de qué tipo son, etc., y otro para la función generadora de código, la cual se encarga de generar la cadena de código correspondiente a ese bloque. Esa parte se escribe siempre bajo Javascript, aun cuando queramos que la generación de código sea a otro lenguaje distinto.

La modificación o creación de bloques se pueden hacer de manera manual manipulando los ficheros JSON o JS que correspondan al mismo. Sin embargo, puede llegar a resultar un trabajo muy complicado, por lo que bajo la plataforma de desarrolladores de Google se ofrece una herramienta generadora de bloques, denominada Blockly Developer Tools [17], la cual nos permite generar bloques de manera personalizada utilizando bloques. Esta herramienta está dividida en dos secciones. La parte izquierda nos sirve para generar el bloque, mientras que en la derecha podemos ir viendo el resultado final, así como el código correspondiente a las dos partes del bloque mencionadas anteriormente.

A pesar de contar con esta herramienta, lo único que conseguimos es la parte del bloque relacionada con la definición, es decir, su aspecto, los texto que contiene, etc. Por ello, la parte de generación de código, se debe de escribir de manera manual una vez tengamos el bloque. Un ejemplo de definición de bloque usando esta herramienta se puede observar en la figura 2.2.2.

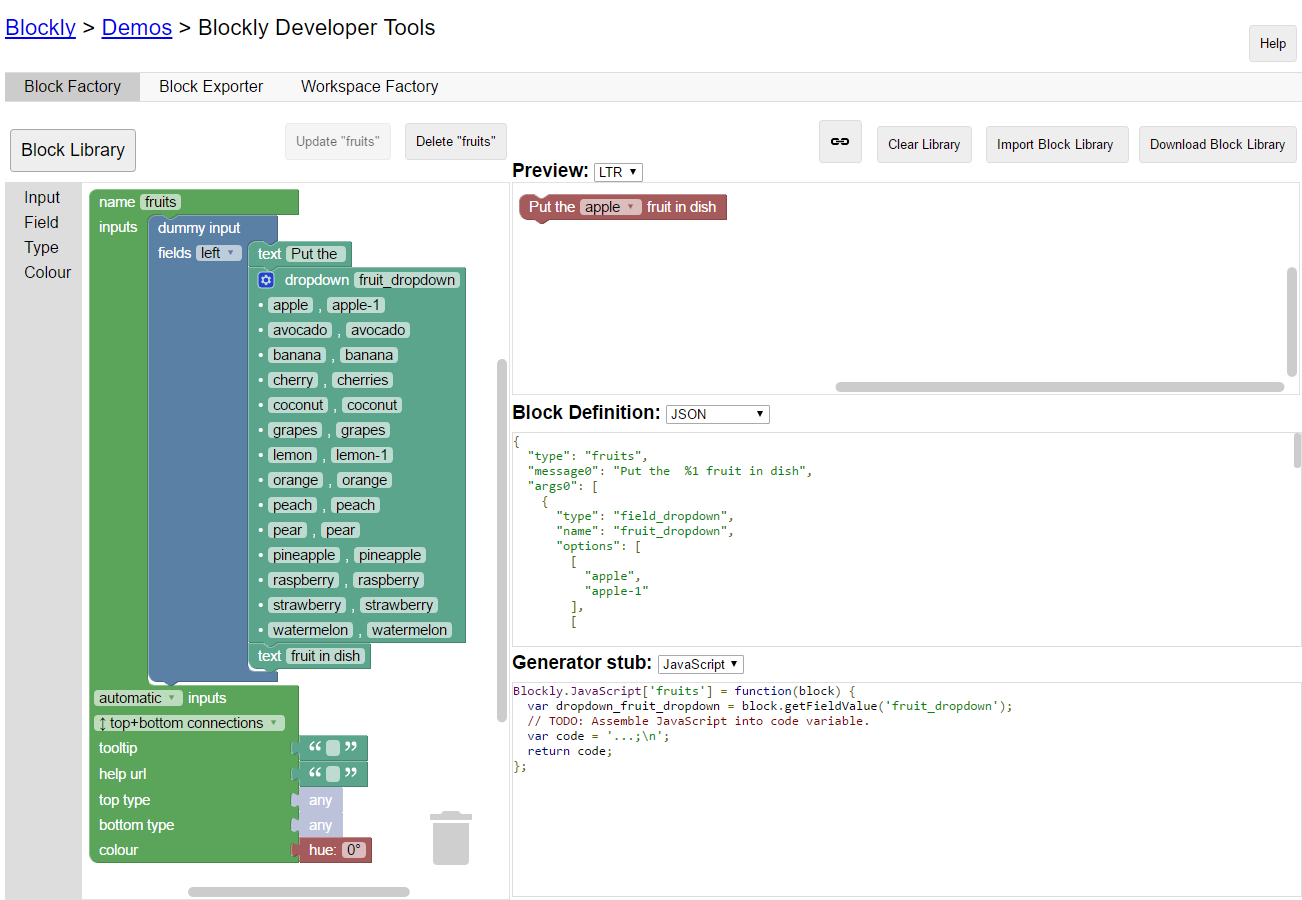


Figura 2.2.2: Ejemplo de bloque generado con Blockly Developer Tools

En la actualidad existen numerosas plataformas que utilizan Blockly como base, siendo las más destacadas Blockly Games [18], una colección de juegos creada por Google para probar la filosofía de bloques, App Inventor [19], un entorno de desarrollo creado por Google Labs y que nos permite crear aplicaciones para el sistema operativo Android utilizando bloques, y por último, Code [7], una organización no gubernamental de Estados Unidos que busca promover la programación en las escuelas, contando con apoyos como Google o Microsoft.

### Pure

Como hemos visto en los anteriores apartados, nuestra aplicación está desarrollada utilizando tecnologías web. Con esto, necesitamos una manera de poder colocar los diferentes objetos de la interfaz de una manera limpia y ordenada utilizando CSS, un lenguaje de diseño que se basa en los lenguajes de marcado. En la actualidad, con el crecimiento de los dispositivos móviles se está estilando el uso de frameworks de este lenguaje, los cuales son librerías que de manera genérica nos ayudan a realizar el diseño de nuestras páginas web. Gracias a ellos, es mucho más fácil y rápido poder crear una página web, con elementos como imágenes, tablas, etc., y que estos se adapten a distintas resoluciones de pantalla. Además, muchos de ellos también nos aseguran que funcionarán sobre distintos navegadores, ya que cumplen los estándares al respecto.

En nuestro caso, entre las numerosas alternativas, se decidió utilizar Pure [20], una solución desarrollada por Yahoo!, y que destaca por ser extremadamente ligera. Este fue el punto decisivo que nos llevó a utilizarlo, desechando otras ideas que son incluso más potentes, ya que vamos a utilizar muchos elementos, como imágenes, vídeos, etc., lo que ocasionará un incremento en el peso global de la aplicación, por lo que necesitamos aligerar el sistema de todas las maneras posibles.

A pesar de utilizar Pure, tenemos que mencionar a Bootstrap, otro framework de CSS que también se barajó. Sin embargo, este es mucho más pesado, puesto que incluye muchas más opciones de personalización y estilos propios. Por esto, el peso total se eleva de los 3.8Kb que ocupa Pure a 134Kb, es decir 35 veces más. A pesar de todo ello, Bootstrap, desarrollado por Twitter, es hasta el momento el segundo repositorio más popular de GitHub [21].



Figura .: Logo de Pure

### Librerías Javascript

Este apartado contiene las distintas librerías y bibliotecas escritas en Javascript que se han utilizado para el correcto funcionamiento de la aplicación desarrollada.

#### jQuery

jQuery [22] es un biblioteca multiplataforma escrita en Javascript y que nos permite interactuar con los documentos HTML y manipular el árbol DOM. Gracias a ella trabajar con los elementos etiquetados en el HTML se vuelve una tarea mucho más sencilla. Fue presentada en el año 2006 y su gran utilidad y versatilidad la han convertido en la biblioteca de Javascript más utilizada en el mundo [23].

Este es un ejemplo en el que podemos manipular un elemento HTML con un identificador ‘mi\_etiqueta’ con código escrito en Javascript sin jQuery:

document.getElementById("mi\_etiqueta")

Y aquí su equivalencia usando jQuery:

$(‘#mi\_etiqueta’)

En nuestro trabajo, jQuery es una herramienta indispensable, ya que como se puede observar, el uso de esta biblioteca favorece el desarrollo de la aplicación, permitiendo una mayor comprensión sobre el elemento con el que se está trabajando, así como conocer o modificar sus atributos escribiendo todo en una sola línea. Como veremos en el capítulo IV de esta memoria, el juego cuenta con numerosos objetos que debemos manipular de manera correcta.



Figura .: Logo de jQuery

#### JS-Interpreter

JS-Interpreter [24] es un intérprete de Javascript escrito en el mismo lenguaje, y que nos permite ejecutar código Javascript de una manera segura y línea a línea. El entorno de ejecución se encuentra aislado del entorno principal de nuestro programa, lo que aporta una mayor seguridad, y además permite mantener varias instancias del intérprete corriendo, especialmente diseñado para aquellas aplicaciones multihilo.

Un ejemplo de uso de JS-Interpreter lo encontramos bajo su misma página web. Por ejemplo, en este caso se quiere probar el siguiente bucle, en el que se multiplica la variable a por valor de i en 4 ocasiones:

var myCode = 'var a=1; for(var i=0;i<4;i++){a\*=i;} a;';

var myInterpreter = new Interpreter(myCode);

Ya hemos cargado la cadena en el intérprete. Para probarla podemos usar myInterpreter.run(), lo que ejecuta el código de myCode. Sin embargo, Javascript ya cuenta con una función parecida y de manera nativa, que es eval().

A pesar de que ya hemos visto que para nuestro caso la aplicación anterior no tendría mucho sentido, también hemos comentado que permite la ejecución paso a paso o línea a línea. Para ello, se define la siguiente función:

function nextStep() {

if (myInterpreter.step()) {

window.setTimeout(nextStep, 0);}

}

nextStep();

Con esa función se puede ejecutar cada paso según nuestra conveniencia. Por ejemplo, en el caso del bucle, podríamos avanzar cada iteración según el usuario pulsase algún botón o cambiando el valor que aparece en ‘0’ en window.setTimeout(nextStep, 0). Esta solución es una de las recomienda Google para la ejecución del código generado con los bloques en nuestra aplicación, e incluso, a pesar de ser un proyecto independiente, nos da los paso para conectarlo con Blockly, de manera que la opción de iluminar los bloques según se va ejecutando el código, sea tratado fuera del entorno de JS-Interpreter, es decir, en el entorno principal [25].

La aplicación real de esta librería se verá con más detalle en el capítulo IV de esta memoria.

#### Chance

Chance.js [26] es una librería ligera escrita en Javascript que nos permite generar números, letras, cadenas de caracteres y otros tipos de datos de manera totalmente aleatoria. Sin embargo, Chance va un paso más allá y también es capaz de generar otros datos, como nombres de personas, números de teléfonos o fechas de cumpleaños. En el caso de Javascript encontramos de manera nativa la función random(), sin embargo, esta función, presente desde ECMAScript 1, solamente nos permite generar números, y en nuestro caso buscábamos cadenas de caracteres.

Para el desarrollo de la aplicación no fue necesario su uso, sin embargo, como veremos en el capítulo IV, para algunos de los ejercicios propuestos esta era la solución más valida. Un ejemplo para generar una cadena de texto aleatoria con 5 letras usando únicamente a, b, c, d y e, es el siguiente:

chance.string({length: 5, pool: 'abcde'});



Figura 2.4.2: Logo de Chance

## Modo de uso

Una vez analizados los antecedentes y las herramientas y tecnologías usadas, mostraremos como funciona nuestra aplicación, a la que hemos llamado Comilona.

### Ventana principal

La ventana principal de nuestra aplicación nos mostrará una imagen como la que vemos en la figura 3.1.1. La interfaz está dividida en tres secciones, la columna izquierda, que contiene el resultado al reto propuesto, la columna derecha, donde se ubica el área de Blockly con su caja de herramientas y su área de trabajo y la barra inferior, que contendrá un selector de niveles, otro para los ejercicios de cada nivel y las banderas de los idiomas disponibles, en un principio solo español e inglés, el cuál es el idioma por defecto.

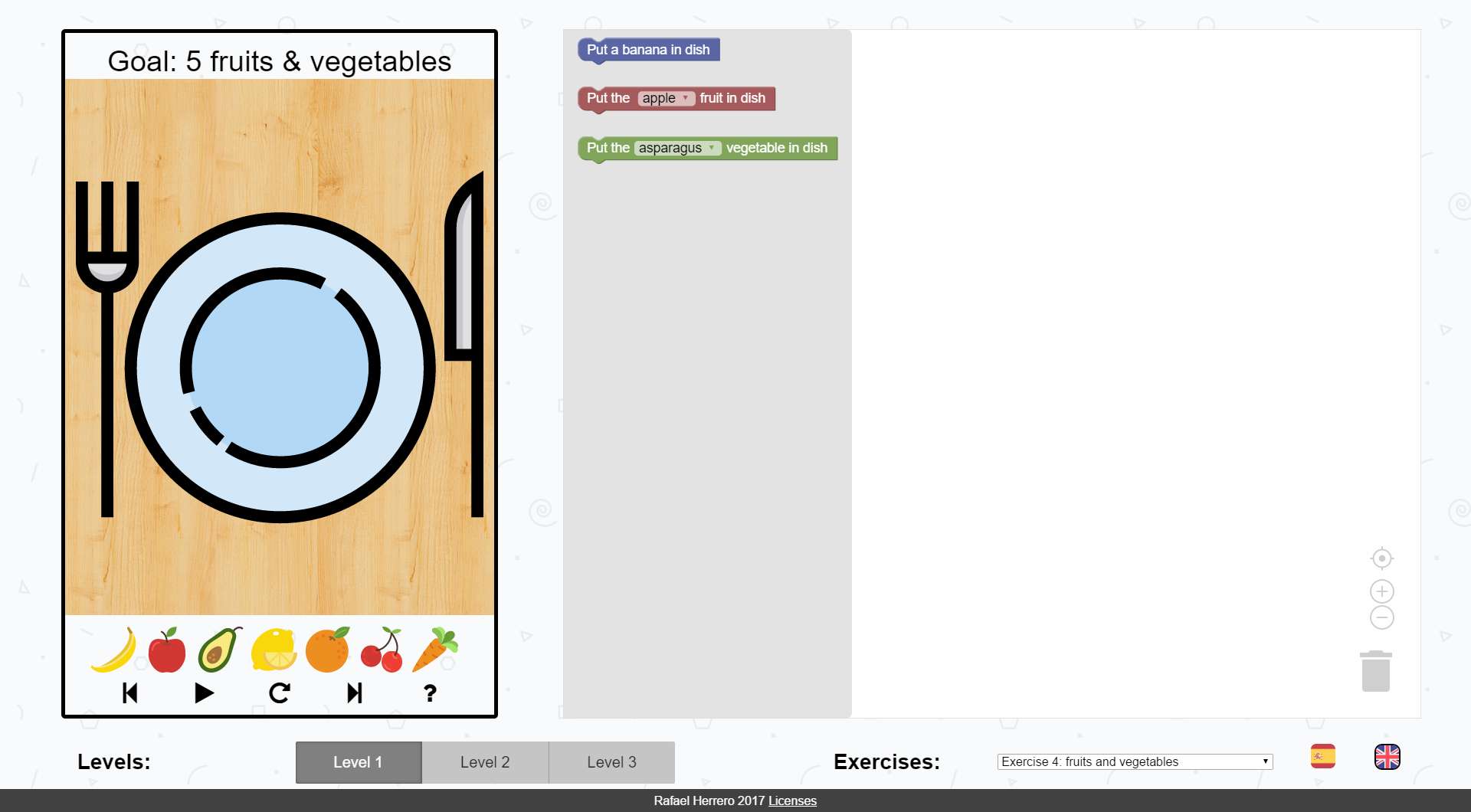


Figura .: Ventana principal de Comilona

Nada más entrar a la aplicación, o a la hora de cargar cualquier nivel o ejercicio, o de cambiar el idioma de la interfaz, nos aparecerá encima de la misma una ventana de información como la que veremos en el siguiente apartado. Esta ventana no influye en el desarrollo de los ejercicios, y cuenta con un botón para cerrarla, de manera que podamos seguir nuestro trabajo sobre la principal.

La distribución está hecha de tal forma que se adapta a la resolución de cualquier pantalla, ya que el framework CSS utilizado, Pure, utiliza un sistema de rejillas, por el cual los contenidos tienen un tamaño por defecto. En nuestro caso, contamos con dos filas, una con las dos columnas, y otra con la barra de selección de niveles y ejercicios. Se ha determinado como resolución mínima de pantalla 1200x700 píxeles, ya que, si bien se puede utilizar en menores medidas, el área de trabajo que nos queda para Blockly se vuelve prácticamente invisible, representando una pequeña porción de la ventana, por lo que directamente, si se quiere utilizar una resolución menor, se mostrará un mensaje como el de la figura 3.1.2.

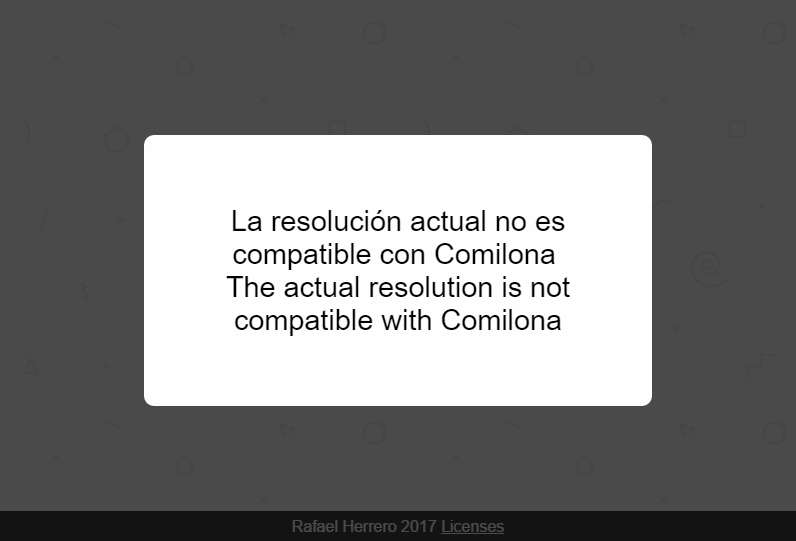


Figura 3.1.2: Mensaje en Comilona si se usa una resolución menor a la permitida

### Ventana de información

Para mostrar la información sobre el reto que se propone a superar, se barajaron varias posibilidades, entre ellas, la opción de utilizar pequeñas ventanas emergentes que indicasen o diesen pistas sobre cómo superar el ejercicio, o también utilizar ventanas de alerta, ya que Electron las trata por defecto como una ventana del sistema, lo que daría mayor cohesión a la aplicación. Sin embargo, estas dos soluciones planteadas pueden llegar a ser muy molestas si se abusa de ellas.

La solución tomada es que la se puede observar en la figura 3.2.1, donde se muestra una ventana blanca con un párrafo de texto y un vídeo instructivo, con un fondo negro y ubicado encima de la ventana principal. El contenido del mismo varía según el ejercicio que se encuentre cargado.

Además, se aprovecha la misma distribución para mostrar, por ejemplo, un mensaje si la resolución de pantalla no es compatible con Comilona, o los datos referidos a la licencia. En caso de que comprobemos un nivel, también saldrá esta ventana con un mensaje modificado, ya que se da la posibilidad de poner tantos como se deseen según el nivel, puesto que uno sencillo solamente contará con dos, si se ha superado o si no, pero uno más complicado puede necesitar más, ya que puede estar bien, pero no ser la solución correcta. Además, debajo del mensaje se mostrarán 3 botones, correspondientes al cambio al ejercicio anterior, a reintentarlo o a pasar al siguiente.

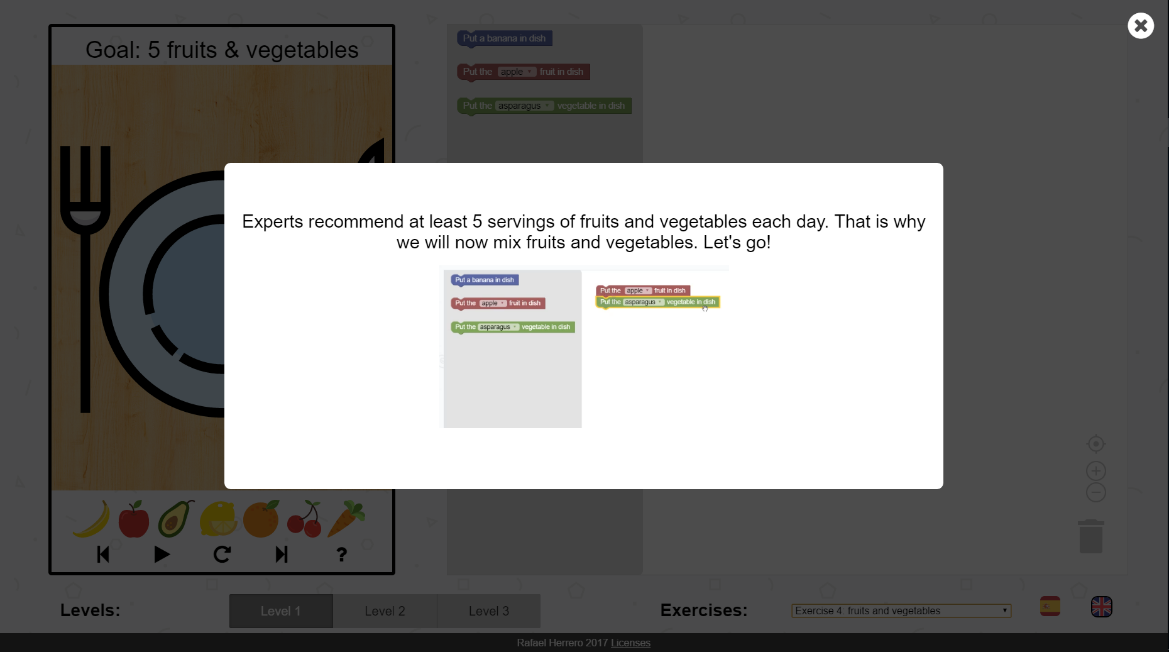


Figura 3.2.1: Ejemplo de ventana informativa para uno de los ejercicios

### Columna izquierda

Siguiendo la distribución utilizada por otras plataformas basadas en la filosofía de bloques como Scratch, la parte izquierda de la ventana principal contiene el resultado de ejecutar el código formado por la cadena o puzle de bloques. Esta parte se muestra en la figura 3.3.1.

En nuestro caso, Comilona tiene dicha sección divida en cuatro partes. La primera de ellas se ubica en la parte superior y contiene un pequeño texto con el objetivo global del ejercicio que se está resolviendo. La segunda, y la más importante, es la parte central, que contiene un plato sobre el que se irán colocando los alimentos. Como veremos en el apartado 3.5, los ejercicios planteados se basan en colocar distintos alimentos en plato, ya sea utilizando secuencias, bucles, funciones o condicionales. La tercera parte actualmente muestra algunos de los alimentos con los que cuenta Comilona. Sin embargo, este segmento está preparado para ser utilizado en un futuro para otros fines.

La última parte se ubica en la zona inferior de la columna, y contiene los siguientes botones de control de la aplicación:

* Retroceso: permite volver al ejercicio anterior. En el caso de estar en el primer ejercicio del primer nivel, este se recargará. Si estamos en el primer ejercicio del segundo o tercer nivel, se cargará el último ejercicio correspondiente al nivel anterior.
* Ejecutar: invoca a la función encargada de generar el código formado por los bloques, además de ejecutar ese mismo y por último comprobar que esa sea la solución que se buscaba para ese ejercicio.
* Recargar: este botón nos permite recargar la aplicación por completo, lo que nos devolverá a los valores por defecto, es decir, el idioma cargado es el inglés y el ejercicio es el primero del primer nivel.
* Siguiente: gracias a esta función, podremos cambiar al siguiente ejercicio dentro del nivel. En caso de estar en el último ejercicio del nivel 3, se cargará el primer ejercicio del primer nivel. Si nos encontramos en el último ejercicio del nivel 1 o 2, se pasará al primer ejercicio del siguiente nivel.
* Información: como vimos en el apartado anterior, Comilona dispone de una ventana para mostrar la información acerca del reto a superar, junto con un pequeño vídeo ilustrativo. Pulsando sobre esta opción, mostraremos de nuevo dicha ventana.

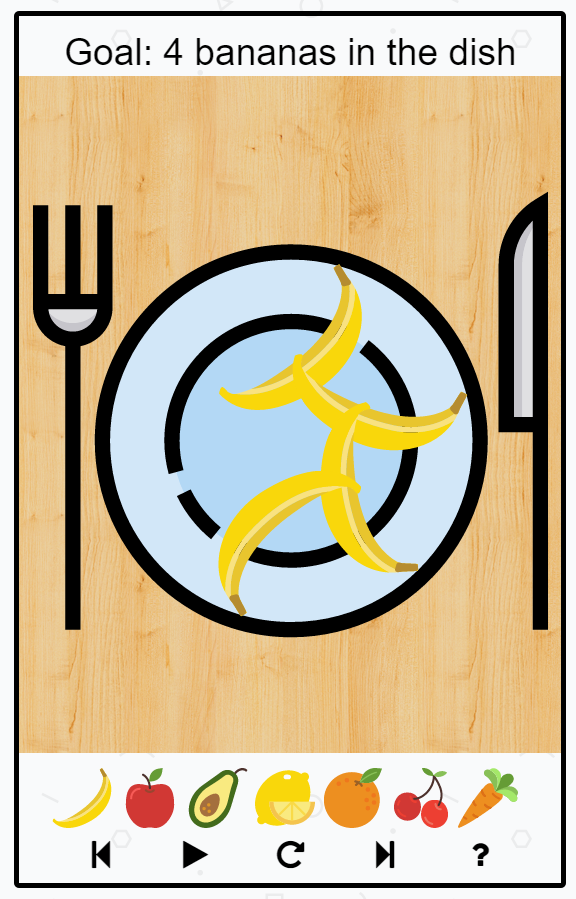


Figura 3.3.1: Ejemplo de la columna izquierda en Comilona

### Columna derecha

Como se ve en el apartado dedicado a la ventana principal, la interfaz cuenta principalmente con dos columnas. En el caso de la derecha, es del doble de tamaño que la de la izquierda, ya que será el objeto principal de todo el juego. Esta sección está dedicada en su totalidad a Blockly, ya que será aquí donde se mostrará su caja de herramientas y su área de trabajo.

Como se explicó en el capítulo II sobre Blockly, para poder utilizarlo necesitamos, al menos, dos partes, una para la caja de herramientas, donde estarán ubicados los bloques que se cambiarán según el nivel que se cargue, y el área de trabajo, donde iremos colocando los bloques que conformarán el puzle con la solución planteada en el reto.

La caja de herramientas permite agrupar los bloques según las categorías que designemos, por ejemplo, una para las operaciones aritméticas y otra para las variables. En nuestro caso, no haremos uso de ellas, puesto que son pocos los bloques que estarán disponible para el usuario en cada nivel.

En el caso del área de trabajo, se trata de una superficie en la que los usuarios colocan los bloques y que además, es adaptable, es decir, se puede aumentar o disminuir la vista para mostrar más o menos bloques. Esta rejilla es transparente, pero puede mostrarse a modo de malla de puntos, de cruces o simplemente en forma de cuadrícula. La misma se puede configurar si queremos que los bloques se organicen en ella o si los queremos de manera libre. Al igual que con la caja de herramientas, este área también es personalizable, pudiendo modificar si queremos mostrar o no los botones de zoom, el botón para centrar la vista o la papelera. Además, también se puede configurar para impedir que podamos cambiar ese aumento con la rueda del ratón, o la velocidad a la que queremos que cambie.

El ejemplo de columna derecha se puede observar en la figura 3.4.1. Como se comentó, la caja de herramientas en nuestro caso no tiene categorías en ningún nivel, y la rejilla del área de trabajo es transparente, al igual que no se organizan los bloques en torno a la cuadrícula, se muestran todos los botones disponibles y se puede realizar zoom con la rueda del ratón.

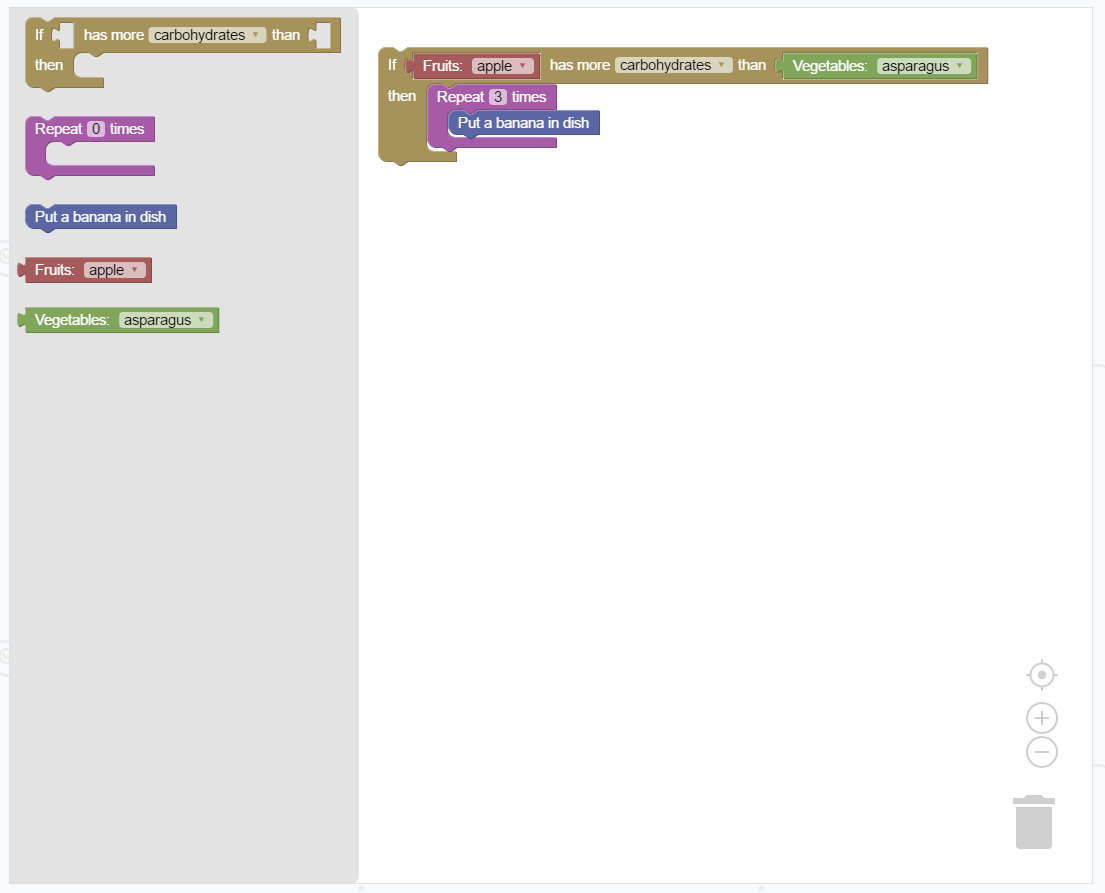


Figura .: Ejemplo de la columna derecha en Comilona

### Barra inferior

Hasta ahora, todo lo visto está relacionado con cómo se trabaja y juega con Comilona. Sin embargo, es necesario contar con alguna manera para cambiar de nivel o ejercicio, así como de adaptar la interfaz a nuestro idioma si así lo deseamos, ya que con los botones de control descritos en el apartado sobre la columna izquierda no somos capaces de abarcar todas las funciones de la aplicación.

En un principio se plantearon varias soluciones distintas a estas, como por ejemplo, controlarlo desde el menú nativo de cada sistema operativo, ya que Electron nos permite crear un menú genérico y que este se muestre sea cuál sea el sistema operativo del usuario. A pesar de ser una característica nativa, nos parecía muy complicado tener que ir a ese menú cada vez que se quería cambiar de ejercicio, lo que también nos obligaba a poner por separado un indicador del nivel en el que se encontraba el usuario.

Por esto, la que resultó más cómoda y flexible es la que podemos ver en la figura 3.5.1, una barra ubicada en la parte baja de la aplicación y que nos permite colocar sobre ella los tres botones relacionados con los niveles de Comilona, así como todos los ejercicios correspondientes a ese nivel. Además, se muestran los iconos de los idiomas disponibles en la aplicación. En caso de la cantidad de niveles creciese, se baraja la posibilidad de cambiarlo por una lista desplegable, como la de los ejercicios, pero que contase con cada icono del idioma junto al nombre.

En caso de que deseemos cambiar de nivel, solamente debemos hacer click sobre el botón correspondiente. Podremos saber en todo momento en que nivel estamos mirando estos botones, ya que el actual se encontrará con un color más oscuro que el resto. Para cambiar de ejercicio, únicamente tenemos que desplegar la lista, seleccionar el ejercicio y listo. El actual se mostrará siempre en el desplegable de la lista, por lo que no tenemos que abrirla para saber en cual estamos. En el caso del idioma, podremos cambiarlo pulsando sobre la bandera del que queramos. Por defecto, el idioma que carga con Comilona es el inglés, y podemos saber cual hemos seleccionado, pues este aparecerá con un borde negro alrededor.



Figura .: Ejemplo de barra inferior con Comilona

### Niveles propuestos

Comilona pretende ser una aplicación de entrada al mundo de la computación y a la vez promover hábitos de alimentación saludables. Para lo primero se siguió la filosofía de programación visual basada en bloques y para lo segundo, se diseñaron 3 niveles con distintos ejercicios en cada uno de ellos, los cuáles van aumentando su dificultad a medida que avanzamos, y en los que se proponen retos como colocar 5 frutas y verduras en un plato, por ejemplo. Al comienzo de cada ejercicio se muestra en la ventana de información un corto vídeo con instrucciones o pistas sobre como colocar los bloques.

#### Nivel 1

Para el primer nivel trabajaremos únicamente el concepto de las secuencias, es decir, ejecutar una línea de código una a una, aunque esta sea repetitiva. Este nivel cuenta con 4 ejercicios, los cuáles analizamos con mayor detalle en el siguiente listado:

* Ejercicio 1, plátanos: este reto es muy sencillo, ya que se trata de la toma de contacto con Comilona. En este ejercicio, el objetivo es colocar un plátano en el plato. Para ello, el usuario solo cuenta con 1 bloque, ‘poner un plátano en plato’, el cuál debe arrastrar desde la caja de herramientas al área de trabajo. Si no coloca ninguno o coloca alguno de más, el sistema le informará que la solución no es la correcta.
* Ejercicio 2, 4 plátanos: siguiendo el caso anterior, ahora se plantea colocar 4 plátanos, en vez de solo 1. Para solucionarlo, se colocan 4 bloques de ‘poner un plátano en plato’ seguidos. Si la cifra de ellos es distinta, se advierte al usuario.
* Ejercicio 3, frutas: en este ejercicio el usuario se encuentra con un bloque más, el cual muestra un desplegable con diversas frutas que puede colocar sobre el plato. Ahora, la meta es conseguir colocar 5 frutas, todas totalmente distintas, en el plato. En caso de colocar 5, pero que no sean diferentes, se informa al usuario de que la solución es correcta, pero no es la planteada para ese reto.
* Ejercicio 4, frutas y verduras: al igual que en el punto anterior, y siguiendo la recomendación de la Organización Mundial de la Salud (OMS) de comer unos 400 gramos diarios de frutas y verduras [27], unas 5 raciones aproximadamente, ahora el usuario tiene la opción de utilizar un nuevo bloque, el cual es igual al de las frutas, pero mostrando hortalizas y verduras. Para superar el ejercicio y pasar al siguiente nivel, se tiene que conectar un total de 5 bloques, cumpliendo como condición que se mezclen ambos alimentos. Este reto es el que se puede observar con detalle en la captura de la figura 3.6.1.

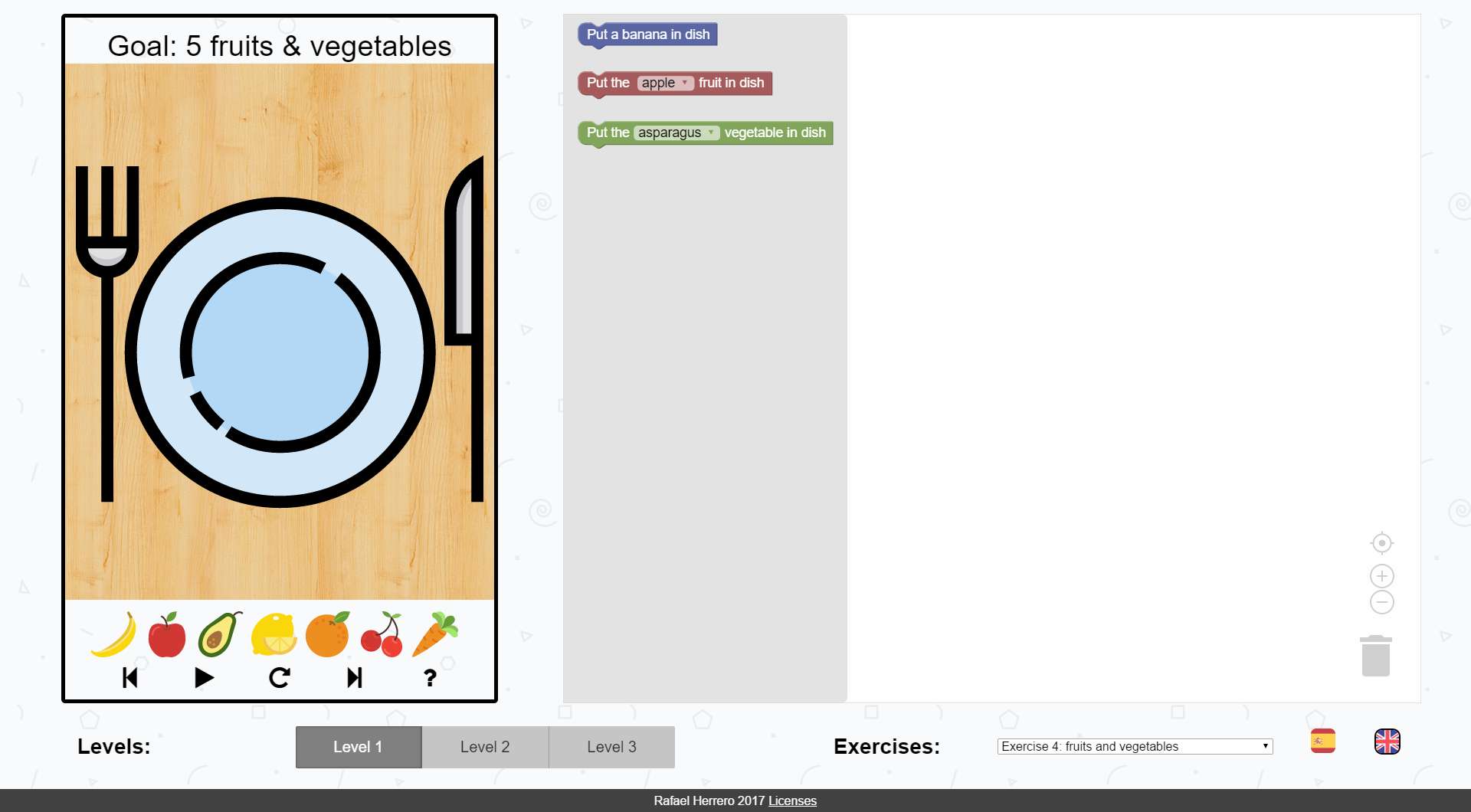


Figura .: Captura del ejercicio 4 del nivel 1

#### Nivel 2

En este nivel podremos trabajar ya con bucles, pero sin abandonar las secuencias. En este caso, el número de ejercicios es igual al del nivel 1, un total de 4, que son los siguientes:

* Ejercicio 1, 4 plátanos: como se puede observar, el título de este ejercicio es igual al del ejercicio 2 del nivel 1, sin embargo, ahora se pone a disposición del usuario un nuevo bloque, el cuál le permite repetir las veces que indique el código que contenga, como podemos observar en la figura 3.6.2. Por ello, para superar este reto, es necesario colocar 4 plátanos en plato, utilizando, por supuesto el bucle. En caso de usar una secuencia y no el bucle, se informa al usuario de que ha alcanzado el objetivo, pero la solución planteada no es la óptima.

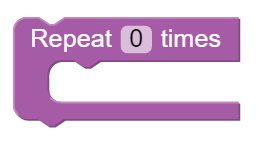


Figura .: Bloque diseñado para indicar un bucle

* Ejercicio 2, 2x2 plátanos: en este ejercicio se pretende demostrar al usuario que es posible utilizar una secuencia dentro de un bucle. Para ello, se dispone de un bloque de bucle, el cual se repite 2 veces, sin posibilidad de cambiar dicha cantidad de repeticiones. De esta manera, para superarlo de correctamente, se deberá utilizar un bucle y dentro colocar dos bloques de ‘poner un plátano en plato’. Si el usuario no coloca 4 plátanos, se informa de que la cantidad total no es la deseada. De igual manera, si la solución que plantea es una secuencia de 4 bloques de ‘poner un plátano en plato’, sin utilizar un bucle, a pesar de que el resultado final es correcto pero no ha cumplido con el objetivo buscado, se informa al usuario de que no ha utilizado ningún bucle y se motiva a que reintente el ejercicio, esta vez haciendo uso del mismo.
* Ejercicio 3, 6 frutas y verduras: al igual que ocurría en el nivel de 5 frutas y verduras, ahora tenemos como objetivo colocar 6, como se ve en la figura 3.6.3. Lo ideal para este, es que se repita el bucle 3 veces y se coloque una secuencia con un bloque de fruta y otro de hortalizas o verduras, de manera que hayan un total de 6. Cualquier solución que cuente con frutas Y verduras, así como con al menos un bucle, se tomará como válida, desechando aquellas que tengan una cantidad distinta de 6 alimentos, que no mezclen frutas y hortalizas o verduras o que no utilicen ningún bucle, sino solo secuencias.
* Ejercicio 4, 2x2x1 plátanos: la meta de este ejercicio es igual al del segundo ejercicio de este nivel, salvo que ahora, para que la solución sea correcta, se tiene que utilizar bucles anidados o en un modo de secuencia, ya que ahora solo se muestra el bloque de ‘poner un plátano en plato’ y un bucle para repetir la acción 2 veces, sin posibilidad de cambiar la cantidad de repeticiones. De esta manera, se fuerza o bien a anidar bucles o a utilizar una secuencia de ellos. En caso de que el usuario no utilizase dos bucles, da igual de qué manera, sino, por ejemplo, una secuencia con 4 bloques de ‘poner un plátano en plato’, se informa de que esa no es la solución correcta.

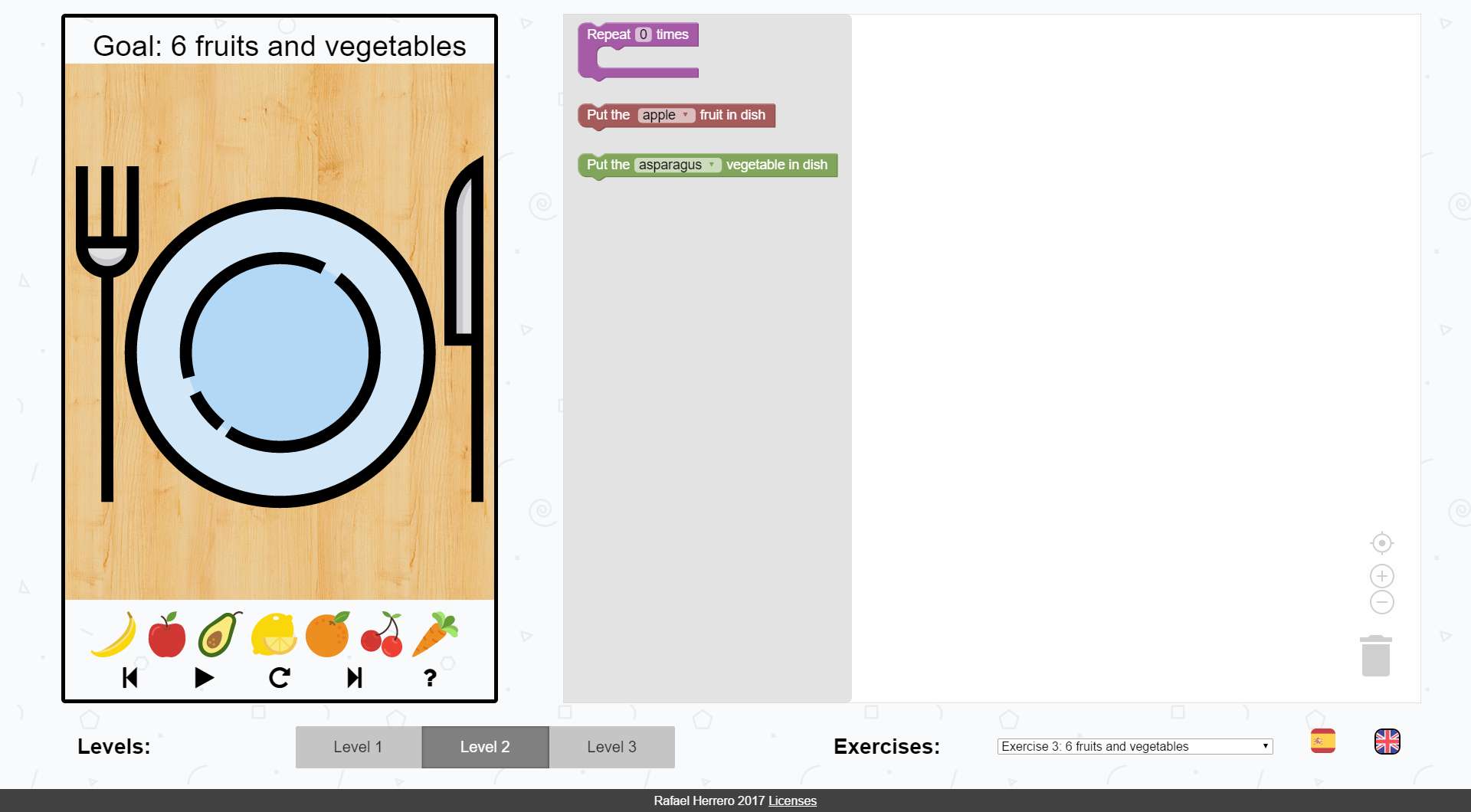


Figura .: Captura del ejercicio 3 del nivel 2

#### Nivel 3

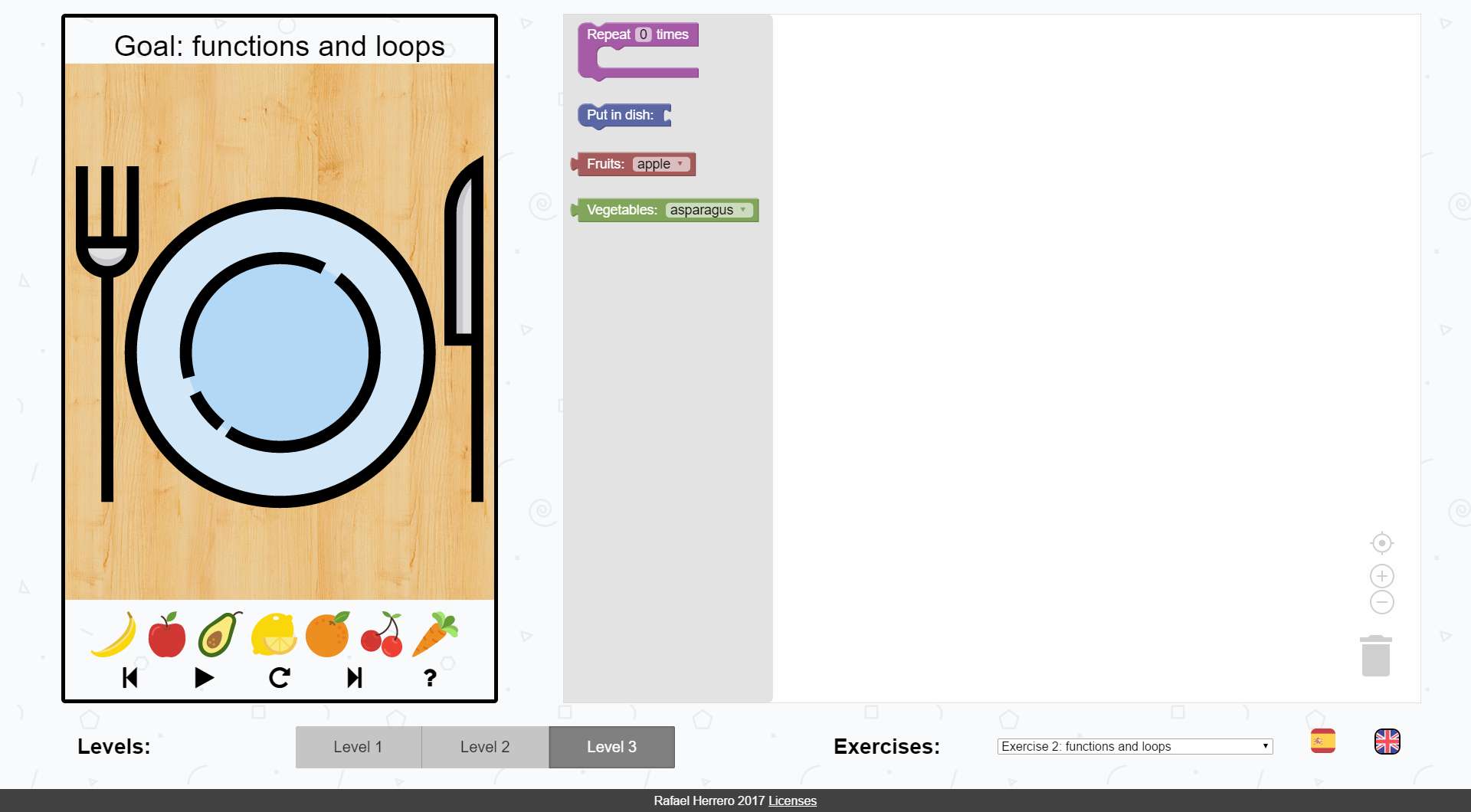
Este nivel se plantea como el último de Comilona. Hasta ahora se han abordado las secuencias y los bucles, por lo que en este nivel se introducen las funciones y los condicionales. Los ejercicios planteados para este nivel son los siguientes:

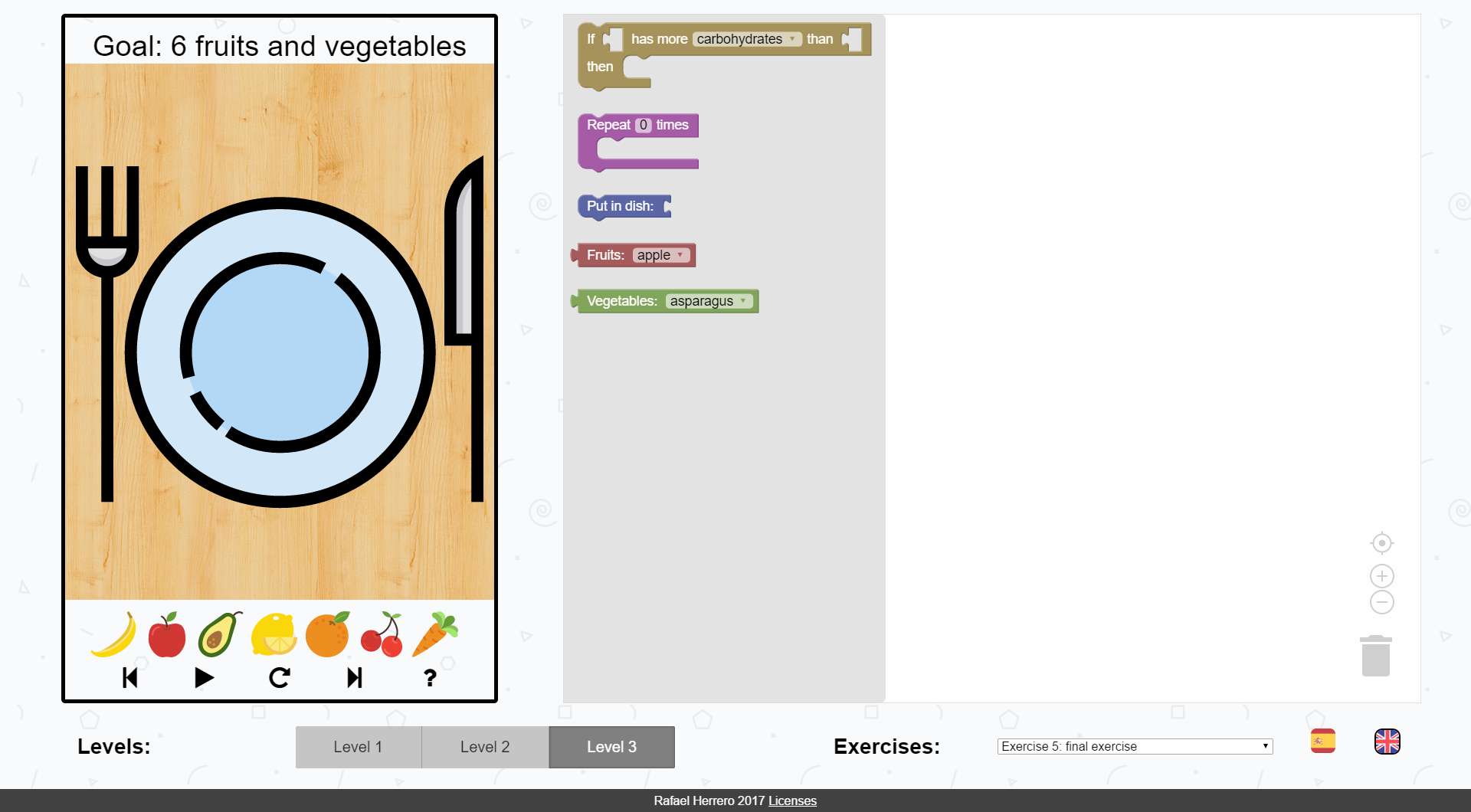
* Ejercicio 1, toda la fruta que quieras: en este nivel, como primer acercamiento a las funciones, se pide al usuario que ponga toda la fruta que quiera en el plato, dando únicamente los bloques con la función ‘poner en el plato’ y un selector de frutas, que sería el dato que recibe la función. El nivel está correcto desde que exista al menos una fruta en el plato, ya que no existe ninguna forma de ponerla si no es usando los dos bloques descritos anteriormente. Para colocar, por ejemplo, una fresa, tenemos que formar el puzle de la figura 3.6.4.



Figura .: Ejemplo de bloque de función

* Ejercicio 2,





## Desarrollo

Los capítulos intermedios servirán para cubrir los siguientes aspectos: antecedentes, problemática o estado del arte, objetivos, fases y desarrollo del proyecto.

En el capítulo 1 se describió bla, bla, bla.....

## Verificación, pruebas, resultados y discusión

Algunos muy avanzados

La curva de aprendizaje muy rápida

Los condicionales fue lo más difícil

---

Cambiar barra inferior de frutas

Disminuir dificultad

Mejorar información mostrada

Dar la posibilidad de bloquear el pase

En caso de que los bloques no estén juntos, dar el ejercicio por malo

## Conclusiones y líneas futuras

Este capítulo es obligatorio. Toda memoria de Trabajo de Fin de Grado debe incluir unas conclusiones y unas líneas de trabajo futuro.

## Summary and Conclusions

This chapter is compulsory. The memory should include an extended summary and conclusions in English.

### First Section

## Presupuesto

Este capítulo es obligatorio. Toda memoria de Trabajo de Fin de Grado debe incluir un presupuesto.

### Sección Uno

|  |  |
| --- | --- |
| Tipos | Descripción |
| AAA | BBB |
| CCC | DDD |
| EEE | FFF |
| GGG | HHH |

|  |  |
| --- | --- |
| Tipos | Descripción |
| AAA | BBB |
| CCC | DDD |
| EEE | FFF |
| GGG | HHH |

Tabla ‑. Tabla resumen de los Tipos.

## Bibliografía

|  |  |
| --- | --- |
| 1. [1] | 1. J. M. Wing, "Computational Thinking," *Communications of the ACM,* vol. 49, no. 3, pp. 33-35, 2006. |
| 1. [2] | 1. S. Bocconi, A. Chioccariello, G. Dettori, A. Ferrari, P. K. K. Engelhardt and Y. Punie, "Exploring the field of computational thinking as a 21st century skill," in *EDULEARN16 Proceedings, ser. 8th International Conference on Education and New Learning Technologies. IATED*, Barcelona, 2016. |
| 1. [3] | 1. T. Bell, J. Alexander, I. Freeman and M. Grimley, "Computer science unplugged: School students doing real computing without computers," *The New Zealand Journal of Applied Computing and Information Technology,* vol. 13, no. 1, pp. 20-29, 2009. |
| 1. [4] | 1. University of Cambridge, "EDSAC 99," [Online]. Available: http://www.cl.cam.ac.uk/events/EDSAC99/statistics.html. |
| 1. [5] | 1. MIT Media Lab, "Scratch," [Online]. Available: https://scratch.mit.edu/. |
| 1. [6] | 1. Code.org, "Hora de código," [Online]. Available: https://hourofcode.com/. |
| 1. [7] | 1. Code.org, "Code.org," [Online]. Available: https://code.org/. |
| 1. [8] | 1. M. T. Burns, M. Benoit and D. Bulvan, "Nutrition Jeopardy," *Journal of Nutrition Education and Behavior,* vol. 34, no. 2, pp. 117-118, 2002. |
| 1. [9] | 1. J. M. Lacey, "The Nutritional SCATTERGORIES® Game: Adding Zest to a Nutrition Course," *Journal of Nutrition Education and Behavior,* vol. 35, no. 6, pp. 333-334, 2003. |
| 1. [10] | 1. Davidap, "Acomola: alimentación equilibrada," Gobierno de Canarias, 12 Mayo 2015. [Online]. Available: http://www3.gobiernodecanarias.org/medusa/ecoescuela/recursosdigitales/2015/05/12/alimentacion-equilibrada/. |
| 1. [11] | 1. GitHub, "Electron," [Online]. Available: https://electron.atom.io/. |
| 1. [12] | 1. Joyent, Inc, "Node.js," [Online]. Available: https://nodejs.org/en/. |
| 1. [13] | 1. Google, "Chromium," [Online]. Available: http://www.chromium.org/. |
| 1. [14] | 1. Google, "Blockly," [Online]. Available: https://developers.google.com/blockly/. |
| 1. [15] | 1. Wikimedia Foundation, Inc, "Blockly - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Blockly. |
| 1. [16] | 1. Google, "Google App Engine," [Online]. Available: https://appengine.google.com. |
| 1. [17] | 1. Google, "Blockly Developer Tools," [Online]. Available: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html. |
| 1. [18] | 1. Google, "Blockly Games," [Online]. Available: https://blockly-games.appspot.com/. |
| 1. [19] | 1. Massachusetts Institute of Technology, "MIT App Inventor," [Online]. Available: http://appinventor.mit.edu/explore/. |
| 1. [20] | 1. Yahoo! Inc, "Pure," [Online]. Available: https://purecss.io/. |
| 1. [21] | 1. GitHub, "Star-filtered repositories," [Online]. Available: https://github.com/search?o=desc&q=stars%3A%3E1&s=stars&type=Repositories. |
| 1. [22] | 1. The jQuery Foundation, "jQuery," [Online]. Available: https://jquery.com/. |
| 1. [23] | 1. Q-Success, "W3Techs," [Online]. Available: https://w3techs.com/technologies/overview/javascript\_library/all. |
| 1. [24] | 1. N. Fraser, "JS-Interpreter on GitHub," [Online]. Available: https://github.com/NeilFraser/JS-Interpreter. |
| 1. [25] | 1. Google, "Generating and Running Javascript," [Online]. Available: https://developers.google.com/blockly/guides/app-integration/running-javascript. |
| 1. [26] | 1. chancejs, "Chance," [Online]. Available: http://chancejs.com/. |
| 1. [27] | 1. Organización Mundial de la Salud, "Fomento del consumo mundial de frutas y verduras," [Online]. Available: http://www.who.int/dietphysicalactivity/fruit/es/. |