Trabajo de Fin de Grado

Grado en Ingeniería Informática

Una aproximación al pensamiento computacional a través de la nutrición

An approach to computational thinking through nutrition

Rafael Herrero Álvarez

La Laguna, 5 de junio de 2017

Dra. Coromoto 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

Dr. 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:

"Una aproximación al pensamiento computacional a través de la nutrición"

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

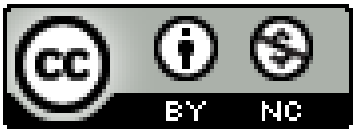
A mi tutora Coromoto, por brindarme esta oportunidad, ser un apoyo incondicional para la realización de este trabajo y por resolver todas las dudas que me surgían.

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 y su personal por permitirme probar la aplicación desarrollada y comprobar su validez en un entorno real.

Licencia



Resumen

El pensamiento computacional es una manera de resolver problemas haciendo uso de las técnicas y habilidades propias de la Computación. Desde hace una década se ha estado trabajando en fomentar su desarrollo desde la infancia, de la misma manera que se aprenden matemáticas o idiomas. El objetivo de este trabajo ha sido analizar el estado del pensamiento computacional en estudios preuniversitarios en la actualidad; investigando las iniciativas y herramientas existentes. Además, se ha prestado especial atención a aquellas actividades que fomenten el pensamiento computacional haciendo uso de conceptos de Nutrición. Ante la carencia de las mismas, se ha desarrollado una aplicación que hemos denominado "Comilona".

En este trabajo se describe el desarrollo y la implementación del juego Comilona. Este se ha desarrollado desde cero haciendo énfasis en los contenidos educativos pero también teniendo en cuenta el disfrute de los jugadores. Se parte de que los jugadores puedan desarrollar sus habilidades de pensamiento computacional sin tener ningún conocimiento de programación previo. El público objetivo son niños de 8 a 12 años y trata de desarrollar conceptos de programación a través de la elaboración de menús dietéticos equilibrados. Se ha desarrollado una aplicación de escritorio independiente y multiplataforma. Funciona en los principales Sistemas Operativos del mercado y es totalmente portable, por lo que puede copiarse a cualquier dispositivo de almacenamiento, como un pendrive o un CD, sin necesidad de instalar nada en la máquina anfitriona. No se necesita conexión a Internet, pero el desarrollo se basó en Tecnologías Web. Se ha programado un cliente en el lenguaje de programación 'Javascript'. La aplicación nativa se creó utilizando el framework 'Electron'. Dado que Comilona se basa en la filosofía de bloques, se ha utilizado la biblioteca 'Blockly' para la construcción de editores de programación visual. Todo el desarrollo está realizado bajo control de versiones y siguiendo la filosofía de código abierto y está disponible en 'Github'.

Finalmente, se realizó un ejercicio estructurado de evaluación con dieciocho jóvenes estudiantes y se analizó y utilizó la evidencia empírica de ese ejercicio para validar este trabajo.

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

Abstract

Computational Thinking is a problem solving strategy which uses the techniques and skills of Computation. In the last decade, much work has been done in the promotion of its development since childhood, just as math or languages subjects. The objective of this work was to analyse the state of the art of computational thinking in pre-university studies today; researching the existing initiatives and tools. In addition, special attention has been paid to those activities that encourage computational thinking activities which uses Nutrition concepts. However, we could not found any and in the face of its lack an application has been developed that we have called "Comilona".

This work describes the development and implementation of the “Comilona” game. It has been crafted from the beginning trying to emphasize the educational content and the players enjoyment at the same time. The users can practice and develop their skills in computational thinking with little or no programming knowledge. The game target audience is 8 to 12 years and it aims is cover programming concepts through the development of menus to a balance diet. A stand-alone, cross-platform desktop application has been developed. It works on the major operating systems on the market and is totally portable, so it can be copied to any storage device, such as a pendrive or a CD, without installing anything on the guest machine. No Internet connection is needed, but the development was based on web technologies. A client has been programmed in the Javascript programming language. The native application was created using the Electron framework. Since Comilona is based on the block philosophy, the library for building visual programming editors Blockly has been used. The development was done under version control and following the open source philosophy and is available in 'Github'.

Finally, a structured evaluation exercise with eighteen young students was performed, and the empirical evidence from that exercise is analysed and used to validate this work.

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

Índice General

Índice de figuras

Índice de tablas

Capítulo 1. 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.

1.1 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 Massachusetts (MIT) en el año 2002.

Figura 1.1.1: 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.

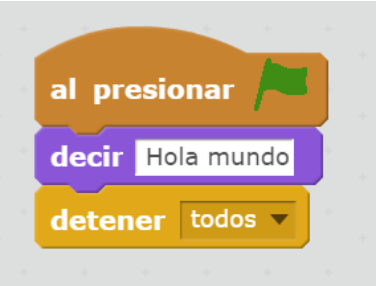


Figura 1.1.2: Ejemplo de mostrar por pantalla 'Hola mundo' en Scratch

Desde que apareció Scratch en el año 2002, han surgido numerosas iniciativas y asociaciones que 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 1.1.1: 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 iniciativas que trabajasen ambos conceptos. Sin embargo, solo se encontró 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 original, donde alguien tiene que decir una letra y otra persona elige una categoría, siendo la meta que se digan la mayor cantidad de palabras posibles en un tiempo determinado, salvo que ahora todas está basado 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 1.1.3: Ventana principal de Acomola

1.2 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.

1.3 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 se envió y ha sido aceptada en el V Congreso Internacional de Videojuegos y Educación (CIVE), celebrado en junio de 2017 en el Puerto de la Cruz, una comunicación sobre el trabajo realizado con la aplicación desarrollada [11].

1.4 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.

Capítulo 2. 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.

2.1 Electron

Electron [12] 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 [13], que a su vez es una adaptación de Javascript, y Chromium [14], 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 2.1.1: Logo de Electron

2.2 Blockly

Como se comentó en el capítulo I, la tecnología principal en la que se basará nuestro desarrollo es Blockly [15], 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 [16], 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 [17], 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.

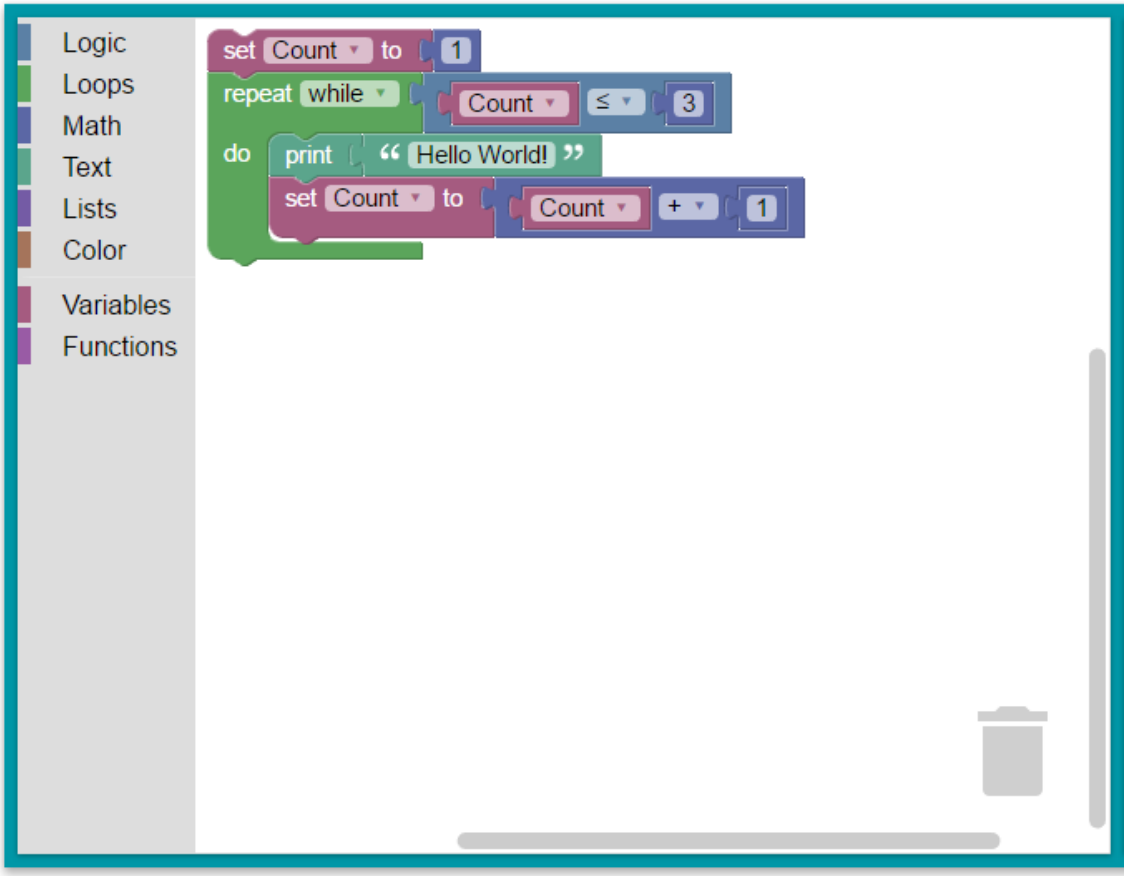


Figura 2.2.1: Interfaz de usuario de Blockly

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 [18], 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.

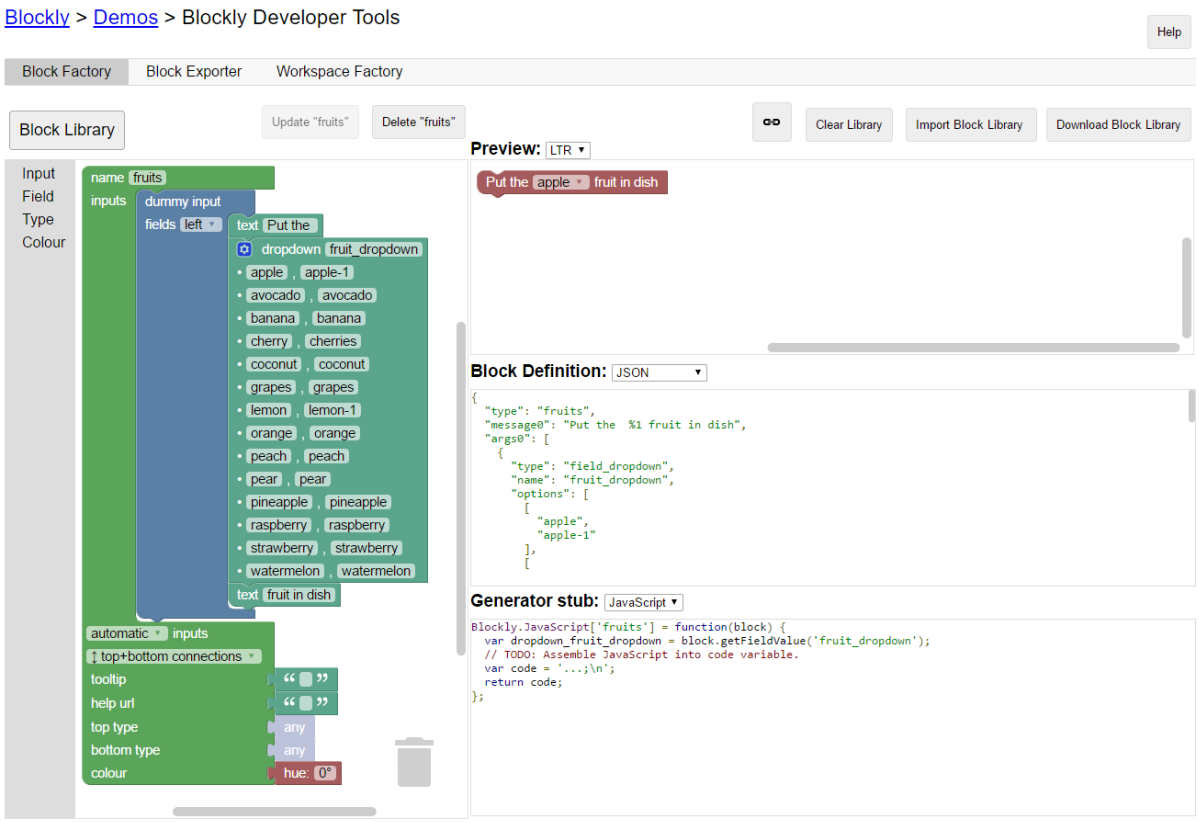


Figura 2.2.2: Ejemplo de bloque generado con Blockly Developer Tools

En la actualidad existen numerosas plataformas que utilizan Blockly como base, siendo las más destacadas Blockly Games [19], una colección de juegos creada por Google para probar la filosofía de bloques, App Inventor [20], 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.

2.3 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 [21], 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 [22].

Figura 2.3.1: Logo de Pure

2.4 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.

2.4.1 jQuery

jQuery [23] 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 [24].

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 2.4.1: Logo de jQuery

2.4.2 JS-Interpreter

JS-Interpreter [25] 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 [26].

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

2.4.3 Chance

Chance.js [27] 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

Capítulo 3.
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.

3.1 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.

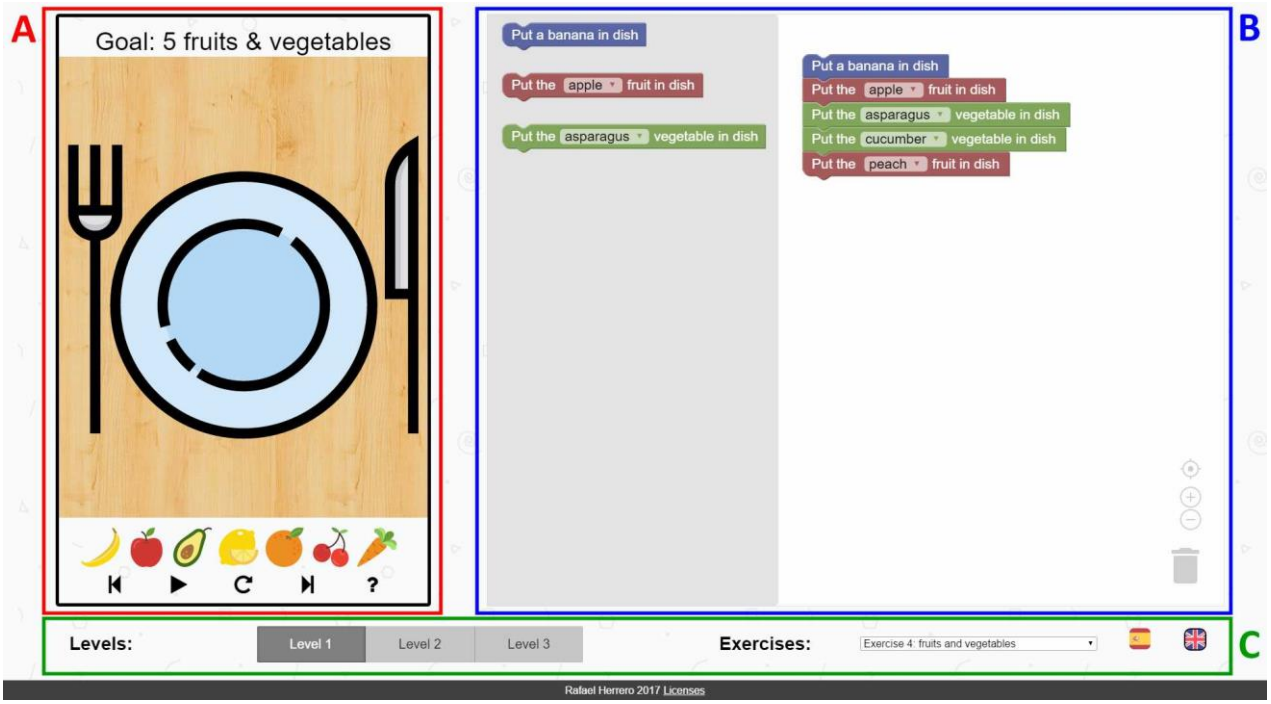


Figura 3.1.1: Ventana principal de Comilona

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 de advertencia 'la resolución actual no es compatible con Comilona'.

3.2 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.

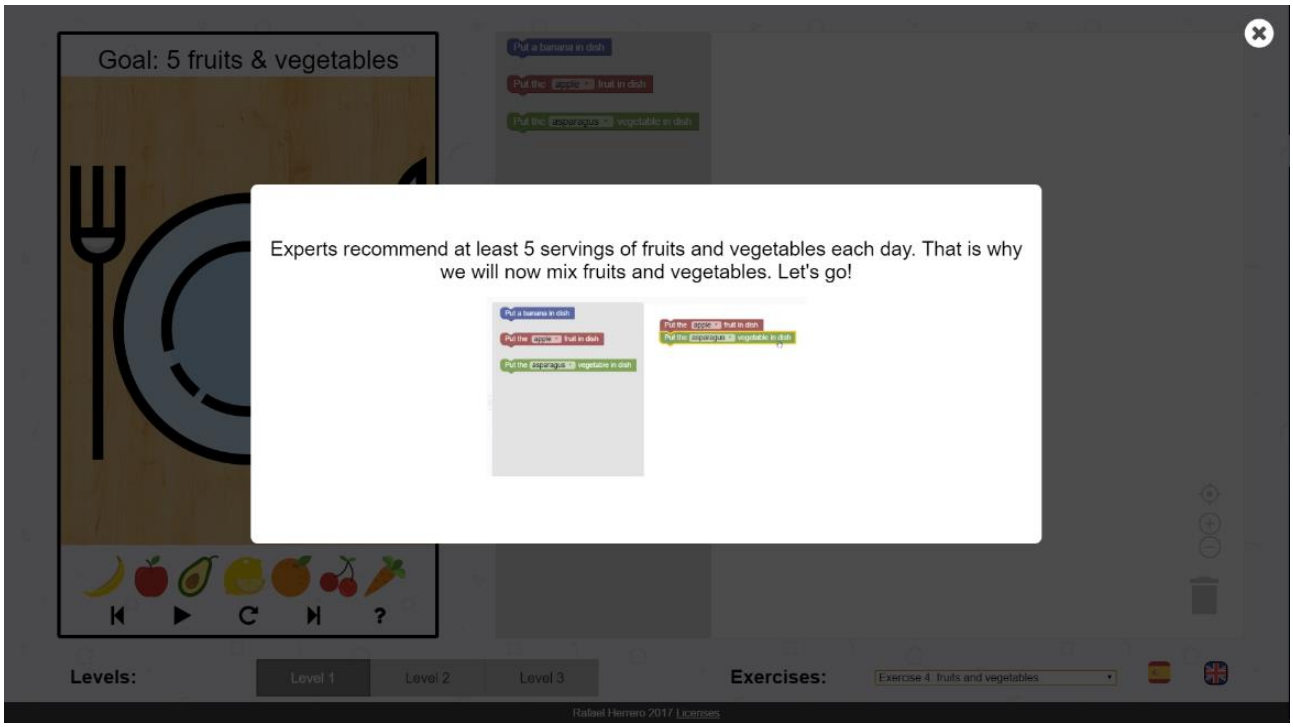


Figura 3.2.1: Ejemplo de ventana informativa para uno de los ejercicios

3.3 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.A.

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 ejercicio junto con un vídeo ilustrativo. Pulsando sobre esta opción, mostramos de nuevo esa ventana.

3.4 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 3.1.1.B. 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.

3.5 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 3.1.1.C, 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 clic 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.

3.6 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.

3.6.1 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. Un

ejemplo de los bloques de este nivel se puede observar en la figura 3.6.1. 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 [28], 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.

Figura 3.6.1: Bloques disponibles en el nivel 1

3.6.2 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.

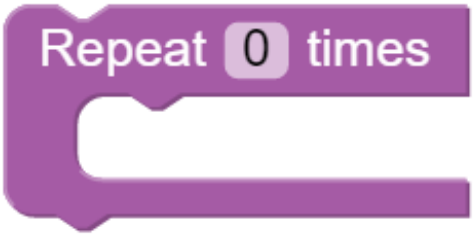


Figura 3.6.2: Bloque diseñado para indicar un bucle

- 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 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. 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.

3.6.3 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 3.6.3.

Figura 3.6.3: Ejemplo de bloque de función

- Ejercicio 2, funciones y bucles: como ocurría en los ejercicios del nivel 2, ahora tenemos que colocar 3 manzanas, 3 naranjas y 3 lechugas utilizando las funciones y los bucles. El uso de las funciones es forzoso, ya que no cabe otra posibilidad. Sin embargo, si no se usa el bucle, se muestra un mensaje informando que esa no es la solución que se pide.
- Ejercicio 3, primer condicional: en este nivel hacemos la primera incursión a los condicionales con uno como el que se muestra en la figura 3.6.4. Para pasar el ejercicio, el usuario debe poner un plátano en el pato, por lo que debe cumplirse la condición que ponga. Se trata de colocar cualquier fruta o verdura que tenga más carbohidratos o grasa o energía (aporte calórico) que otra fruta o verdura. Si esa condición es cierta, se ejecutar lo contenido dentro del bloque. Si no se usa el condicional, se da el ejercicio como no superado.

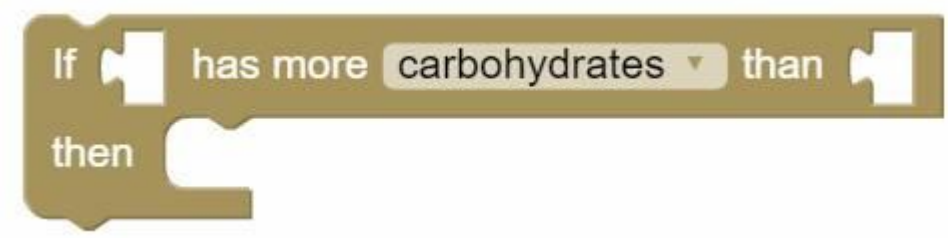


Figura 3.6.4: Bloque condicional

- Ejercicio 4, condicional con bucle: ahora introducimos el bucle con los condicionales. Para superar el nivel, se debe colocar 6 plátanos en el plato, cumpliendo la condición del condicional y utilizando un bucle. Si no se utiliza el bucle, se pide al usuario que reintente el ejercicio.
- Ejercicio 5, ejercicio final: en este último ejercicio se mezcla todo lo aprendido durante el juego, secuencias, bucles, funciones y condicionales. La idea es que utilizando todo ello, el usuario sea capaz de poner 6 frutas y verduras, por lo que no pueden ser todos los alimentos del mismo tipo. Si no se superase este nivel, se le pide al usuario que comience de nuevo todo el nivel. En caso de dar con la solución correcta, se invita a que vuelva a jugar a Comilona desde el primer ejercicio del nivel 1.

Capítulo 4. Desarrollo

Este capítulo se centra exclusivamente en los datos técnicos del desarrollo de la aplicación. Como jugar y todos los detalles sobre la interfaz del mismo se encuentran en el capítulo III sobre el modo de uso. Todo el desarrollo de esta aplicación se llevó a cabo siguiendo la filosofía de código abierto y se realizó bajo git, un software de control de versiones. De igual manera, se fue subiendo a la plataforma GitHub, desde donde podemos descargarlo [29].

4.1 Estructura de la interfaz de la aplicación

La interfaz de la aplicación se estructuró utilizando el framework de CSS Pure. Para conseguir el resultado de la figura 3.1.1, utilizando el lenguaje de marcado HTML se dispuso de dos filas, una para las dos columnas y otra para la barra inferior. En el caso de la columna izquierda, esta tiene el tamaño de 7 porciones de las 24 disponibles, mientras que la izquierda mide el doble, 14. Las 3 porciones restantes se aprovechan como separadores de las esquinas de la ventana y entre ellos. En el caso de la barra inferior, tiene el tamaño completo de las 24 porciones, 10 para el selector de niveles, 8 para el de los ejercicios y 1 para cada idioma.

Todos los campos de texto de la interfaz, así como los botones que aparecen, no tienen ningún valor en el HTML, sino simplemente una etiqueta identificadora, que como veremos más adelante, se utiliza para poder cargar dichos valores de manera personalizada, por lo que este archivo de marcado se podría decir que está vacío.

El diseño de la interfaz se lleva a cabo utilizando una hoja de estilos CSS independiente. En ella se modifica, por ejemplo, el fondo de la aplicación, o el estilo de los botones cuando uno de ellos está activo. El patrón de fondo, así como la madera que se encuentra bajo el plato en la columna izquierda, se ha descargado de Subtle Patterns [30]. Los dibujos utilizados, así como las banderas de los idiomas pertenecen a Freepik by Flaticon [31].

4.2 Javascript y Node.js

Como mencionamos en capítulos anteriores, el lenguaje de programación utilizado ha sido Javascript. Comilona cuenta, ignorando los propios de Blockly o jQuery, con un único archivo Javascript, que se encarga del manejo de la aplicación. Todo está escrito en inglés, tanto las funciones como las variables, así como el nombre de todos los ficheros, por lo que si se realiza algún cambio, recomendamos que siga utilizando dicho idioma para mantener cierta consistencia.

Gracias al archivo principal, podemos controlar, por ejemplo, que el usuario no pueda utilizar la interfaz si su resolución no es compatible con Comilona, o por ejemplo, gestionar la carga de los ejercicios o el cambio de idioma.

Se ha intentado no utilizar el lenguaje Node en todo lo posible, ya que si en un futuro se quiere portar a un servidor, los navegadores no son capaces de renderizarlo, a pesar de que esté basado en Javascript. Sin embargo, ha sido necesario para poder cargar archivos locales desde el archivo principal de Javascript, ya que ningún navegador lo permite, puesto que supone un peligro.

A pesar de contar con un solo fichero de control, hemos desarrollado todo de manera modular. Por ello, existe una única función que se encarga de cargar los ficheros de manera externa, por lo que si queremos portarlo en un futuro, solo tendríamos que reescribir dicha función para que fuese capaz de recibir los ficheros del servidor y no buscarlos de manera local.

También contamos con funciones para recuperar desde la interfaz cual es el nivel actual, el ejercicio cargado o el idioma que se está actualizando, de manera que en cualquier momento podemos llamar a dichas funciones. En los dos primeros casos nos devolverían un número y el caso del segundo se ha adoptado que deba ponerse el nombre del idioma y luego la palabra language, de manera que el español sería '`spanish_language`'.

4.3 Carga de los datos

En el apartado anterior se mencionó que Comilona solo cuenta con un archivo de Javascript para controlarlo, obviando otras tecnologías. Sin embargo, es imposible conseguir lo que queremos con un único archivo, y lo hemos conseguido siguiendo un esquema completamente modular.

Todos y cada uno de los ejercicios se encuentran de manera totalmente separada del fichero principal. Para ello, se ha diseñado una estructura, que, utilizando el formato JSON, nos permite definir los ejercicios, los niveles y los idiomas. Como comentamos anteriormente, el HTML se encuentra 'vacío', es decir, que si no pudiésemos ejecutar Javascript, tendríamos una interfaz prácticamente en blanco, ya que no se cargarían los textos.

En el árbol de directorios, encontramos 3 carpetas que son fundamentales para Comilona, '`json`', que alberga los ficheros que nos permiten definir los retos, los idiomas, etc., '`js`', que contiene cada uno de los ficheros de corrección de los retos, es decir, el conjunto de código que se encarga de definir si la respuesta es la correcta o no para cada ejercicio por separado, ya que las posibles soluciones cambian de uno a otro. Y por último, '`video`', que debe incluir los vídeos informativos para cada ejercicio. Dentro de estas carpetas se encuentran otras 3, una para cada uno de los niveles, llamadas '`levelx`', donde '`x`' es el número del nivel. De manera paralela se encuentra la carpeta '`blocks`', que contienen los ficheros Javascript con los bloques personalizados, incluyendo tanto la parte de definición del bloque como de generación del código. Todos los bloques del juego se definieron utilizando la herramienta de Blockly Developer Tools [18].

El funcionamiento para la carga es muy sencillo. Tenemos que tener en cuenta que lo único que permanece invariable es el número de niveles, que se ha fijado en 3, el resto es dinámico. Lo primero que hace Comilona, es leer el archivo '`english_language`', ubicado en la carpeta '`languages`', dentro de la ya mencionada '`json`'. Este archivo tiene la estructura siguiente, para la cual se ha generado una plantilla por si se quiere añadir un nuevo idioma:

```
{
    "title": "template_language",
    "buttons":[
        {"id": "button_level1", "value": ""},
        {"id": "button_level2", "value": ""},
        {"id": "button_level3", "value": ""},
        {"id": "check_level_previous_exercise", "value": ""},
        {"id": "check_level_retry_exercise", "value": ""},
        {"id": "check_level_next_exercise", "value": ""}
    ],
    "texts":[
        {"id": "level_text", "value": ""},
        {"id": "exercise_text", "value": ""}
    ]
}
```

Como vemos, cuenta con un campo para cada uno de los elementos de la interfaz que contienen texto. Una vez cargado el idioma, Comilona necesita los datos del nivel, por defecto, el número 1. El fichero que define a los niveles se encuentra dentro de la carpeta '`json`', dentro de la subcarpeta del nivel, en este caso, dentro de '`level1`' En este caso, la definición del mismo se hace en base a los idiomas que tiene, por lo que se debe de añadir al mismo si contamos con alguno nuevo. También se ha generado una plantilla por defecto:

```
{
    "spanish_language":[
        {
            "title": "",
            "description_level": ""
        }
    ],
    "english_language":[
        {
            "title": "Level 1",
            "description_level": ""
        }
    ],
    "exercises": "x"
}
```

Es necesario incluir en el campo '`exercises`', la cantidad de ejercicios del nivel, ya que a diferencia de la cantidad de niveles, este si puede variar. Además, una vez cargado el nivel, se procede a cargar los ejercicios. Primero, el sistema, según la cantidad de ejercicios indicados, lee el campo '`title`' de todos los ficheros de los ejercicios para incluirlos en el selector. Tras esto, únicamente se carga el fichero correspondiente al ejercicio 1, con nombre de fichero '`exercisex`', donde la '`x`' representa el número del reto.

Ese fichero es como el que podemos ver a continuación. Contiene el título del ejercicio, una breve descripción del reto, que aparecerá en la ventana de información del mismo, y de la meta a lograr, así como tantos campos con textos como sean necesarios, que indicarán en la fase de corrección si la solución planteada es la correcta o no, si está bien pero no perfecto, etc. Esto es así, puesto que la corrección no es universal, como veremos a continuación. También es necesario incluir el nombre de los bloques en '`blocks`' que se usarán, y el identificador único de ejercicio '`levelx_exercisey`', donde la '`x`' corresponde al

número de nivel y la ‘y’ al número de ejercicio. Esta es la plantilla de definición de un ejercicio:

```
{
    "spanish_language":[
        {
            "title": "Ejercicio x: ",
            "description_game": "",
            "goal": "",
            "text_1": ""
        }
    ],
    "english_language":[
        {
            "title": "Exercise x: ",
           "description_game": "",
            "goal": "",
            "text_1": ""
        }
    ],
    "blocks":[""],
    "value": "levelx_exercisey"
}
```

Además, el vídeo con las instrucciones de cada ejercicio deberá incluirse en la carpeta ‘video’, dentro de la subcarpeta correspondiente a cada nivel, y con el título ‘exercisex’, donde la última ‘x’ corresponde al número del ejercicio.

En caso de los bloques, estos se encontrar dentro de la carpeta ‘blocks’ y dentro de la carpeta correspondiente a su idioma. Debemos copiar el mismo bloque tantas veces como idiomas tengamos. El fichero debe llamarse ‘blocks.js’ y en este caso es necesario cargarlo manualmente desde el HTML, ya que si no, para poder cambiar a otro idioma sería necesario reiniciar la aplicación, o al menos Blockly, lo que ocasiona un esfuerzo doble. Todos los bloques tienen que llamarse ‘x_language_y’, donde la ‘x’ representa el idioma del bloque y la ‘y’ el nombre del bloque. Para cargarlo, desde el fichero de ejercicios mostrado en la página anterior, tenemos que poner únicamente la ‘y’ en el campo ‘blocks’, es decir, solo el nombre, sin que sea necesario indicar el idioma. En este caso no hay plantilla puesto que los bloques son totalmente personalizados.

Por último, falta la corrección de los ejercicios. En este caso, tampoco existe una manera determinada de hacerlo, puesto que cada ejercicio tendrá su forma de corregirse según el reto que se haya planteado. Estos ficheros se tienen que ubicar en la carpeta ‘js’, dentro de la carpeta correspondiente al nivel y como

nombre del mismo, '`exercisex`', donde la última '`x`' representa el número del nivel.

Por esto, a pesar de que Comilona es un juego y no una plataforma de desarrollo de juegos, se permite la creación de los ejercicios del mismo, pudiendo modificar los ficheros del idioma, del nivel y del ejercicio, además de poder incluir nuestros propios bloques.

4.4 Puesta en marcha y empaquetado de la aplicación

Para poder usar Comilona o para empaquetarlo, necesitamos tener instalado Node.js en nuestra máquina, siguiendo las instrucciones de la página oficial [13]. Una vez se haya empaquetado, ya no necesitaremos Node.

Cuando terminemos la instalación, accedemos a la página del repositorio en GitHub [29] y lo descargamos. Lo descomprimimos, y una vez dentro de la carpeta comprobamos que en el fichero '`myjs.js`', dentro de la carpeta '`js`', tiene la variable '`desarrollo`', que está al comienzo del documento, como '`true`'. Una vez lo cambiemos, ejecutamos los siguientes comandos:

```
npm install //Instala todas las dependencias necesarias
npm start //Ejecuta Comilona utilizando Electron
```

Ya lo hemos puesto en marcha, y podemos ver los cambios que hagamos recargando la página, o bien pulsando sobre el botón de recargar de la columna izquierda, o pulsando la combinación 'Ctrl + R'.

Sin embargo, si lo que buscamos es empaquetarlo, para Windows o cualquier distribución basada en Linux, tenemos que poner la variable '`desarrollo`' mencionada anteriormente a '`false`', ya que si no, Comilona no será capaz de encontrar los ficheros JSON de los ejercicios y veremos una interfaz en blanco.

Para empaquetarlo, abrimos una terminal y ejecutamos lo siguiente:

```
Npm run pack… //Ejecuta la tarea correspondiente al empaquetado
```

En los puntos suspensivos podemos poner las siguientes combinaciones:

- winx64: si el sistema operativo objetivo es un Windows de 64 bits.
- winx32: si el sistema operativo objetivo es un Windows de 32 bits.
- linuxx64: si el sistema operativo objetivo es un Linux de 64 bits.
- linuxx32: si el sistema operativo objetivo es un Linux de 32 bits.

El proceso de empaquetado se puede realizar bajo cualquiera de los dos sistemas operativos. La aplicación empaquetada se encuentra en la carpeta '`dist`', dentro del mismo directorio del repositorio.

Capítulo 5. Verificación, pruebas, resultados y discusión

En este capítulo analizaremos el correcto funcionamiento de la aplicación en distintos entornos, así como los resultados de realizar una prueba real con niños de 10 a 11 años.

5.1 Pruebas sobre distintos entornos

Para comprobar el correcto funcionamiento de la aplicación, se probó en modo desarrollador bajo los sistemas Windows, Mac OS y cualquier distribución basada en Linux. En todos ellos, funcionaba a la perfección. Sin embargo, cabe destacar que no fuimos capaces de que la aplicación empaquetada funcionase sobre Mac OS, debido a un fallo en la carga de los ficheros JSON, que como mencionamos anteriormente, se cargan de manera dinámica, no cuando arranca la aplicación.

Otra de las pruebas realizadas se basaba en probar en comportamiento de la aplicación en distintas resoluciones. En el caso de Linux, si la resolución era muy justa, cerca de los 700 píxeles de alto, podía ocurrir que al arrancar apareciese el mensaje advirtiendo que la resolución no era compatible. Esto se solucionaba restaurando y maximizando la ventana de nuevo, o entrando en el modo de pantalla completa pulsando 'F11'.

5.2 Resultados de pruebas en un aula

Gracias al Colegio Nuryana, pudimos comprobar con una clase de 5º de primaria, con niños de 10 y 11 años, si realmente Comilona cumple con el objetivo o no de introducir nuevos conceptos. Para ello, se estuvo en el aula durante 1 hora y media aproximadamente. Se les pasó un test antes de que probasen Comilona. Luego se les dejó jugar libremente hasta que todos terminasen todos los ejercicios. Y por último, se les pasó el mismo test que al

comienzo, pero esta vez se añadieron nuevas preguntas relacionadas con la aplicación. En cuanto a los datos generales, contamos con un total de 18 respuestas de niños de 10 y 11 años, en el que destacamos que 10 son mujeres y 8 hombres.

En el desarrollo de la prueba se observó que la media general era buena, es decir, avanzaban prácticamente al mismo nivel todos los estudiantes, sin embargo, cabe destacar que dos de ellos acabaron bastante rápido, en comparación con sus compañeros. Por esto, podemos añadir que la curva de aprendizaje sobre el juego fue muy rápida, ya que solamente preguntaron al comienzo sobre como colocar los bloques, o que tenían que hacer, y luego empezaron a pasar los ejercicios ellos solos. Cabe destacar que los condicionales resultaron lo más complicado del juego, ya que muchos de ellos preguntaban cómo funcionaban esos ejercicios. Sin embargo, esto era algo que ya preveíamos y esperábamos.

Gracias a estas pruebas, pudimos recabar algunos cambios que debemos hacer sobre la interfaz o los ejercicios planteados. Lo primero, cambiar la barra inferior de frutas de la columna izquierda, ya que confundía al pedir que el usuario pusiese un plátano en el plato, lo que llevaba a algunos a arrastrar la imagen en vez del bloque. También deberíamos plantear la reducción de la dificultad de algunos ejercicios, sobre todo en los últimos de condicionales, así como plantear una nueva manera de mostrar la información sobre los niveles, ya que algunos de los niños cerraban la ventana directamente y se fijaban únicamente en el indicador de la meta, en la parte superior de la columna izquierda.

Igualmente, sería beneficioso poder bloquear el pase al siguiente ejercicio si el usuario no consigue generar la solución correcta, ya que nosotros informamos que la solución no es la correcta, pero permitimos que siga avanzando. Además, en el caso de Blockly se detectó un problema, y es que a la hora de generar el código de los bloques colocados en el área de trabajo, da igual si están en una secuencia o no, ya que, por ejemplo, en los primeros ejercicios de colocar un plátano en el plato, el código de cada bloque simplemente coloca el plátano, por lo que da igual si están en una secuencia o no.

En cuanto a los test pasados a los alumnos, destacamos las siguientes preguntas, como por ejemplo, '¿te gusta la computación?' y '¿estudiarías computación?', ya que como podemos ver en la figura 5.2.1, donde se puede

observar que, a pesar de que en general sí que les gusta, a la mayoría de ellos no les interesa estudiar nada relacionado.

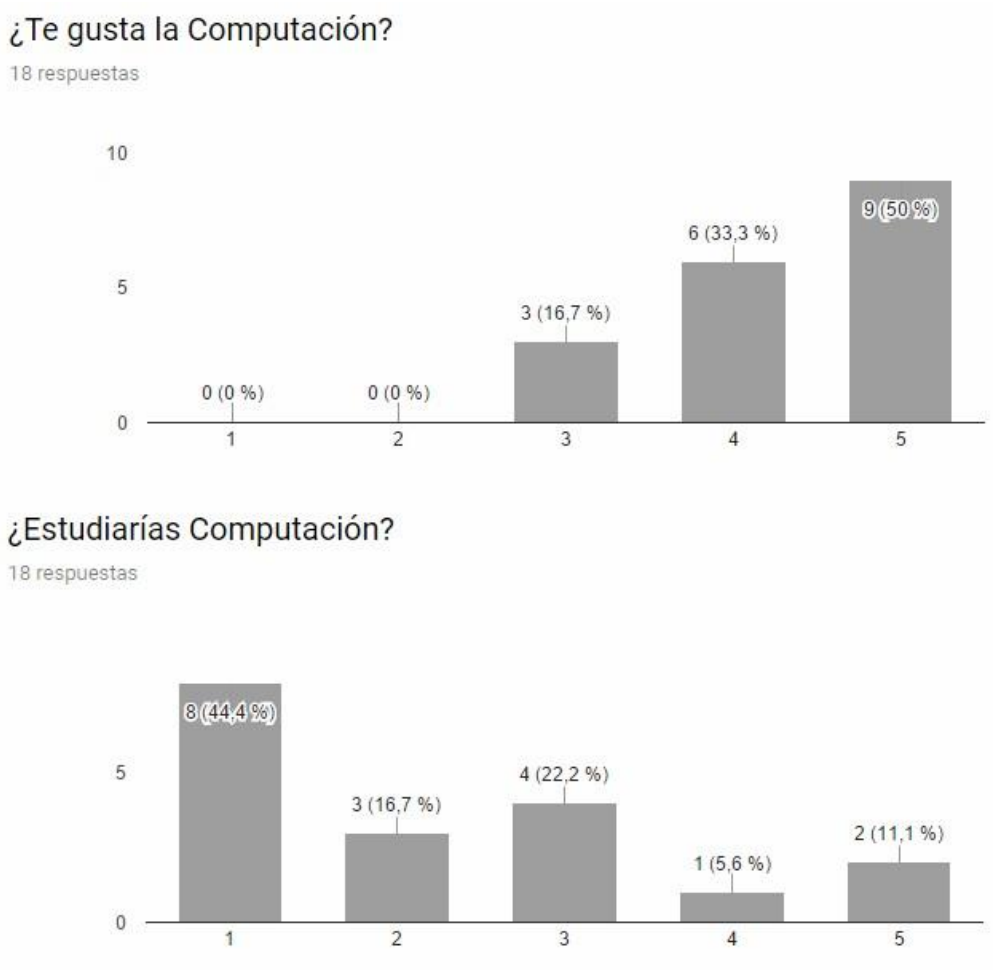


Figura 5.2.1: resultado 1 de preguntas

Otra que destacamos es '¿qué grado de interés ha despertado en ti este juego de programación?, donde se puede observar que a la mitad les ha llamado la atención y a otro tercio del total de los alumnos también, pero no igual de llamativo. Otras de las preguntas, ahora más relacionadas con el juego, las encontramos en la figura 5.2.2, y en este caso fueron sobre la evaluación de la facilidad de aprendizaje y sobre la apariencia en general de Comilona. En cuanto a la primera, vemos lo que comentábamos anteriormente sobre que la curva de aprendizaje era muy rápida, mientras que en la segunda, podemos confirmar que la interfaz resultaba muy visual y sencilla.

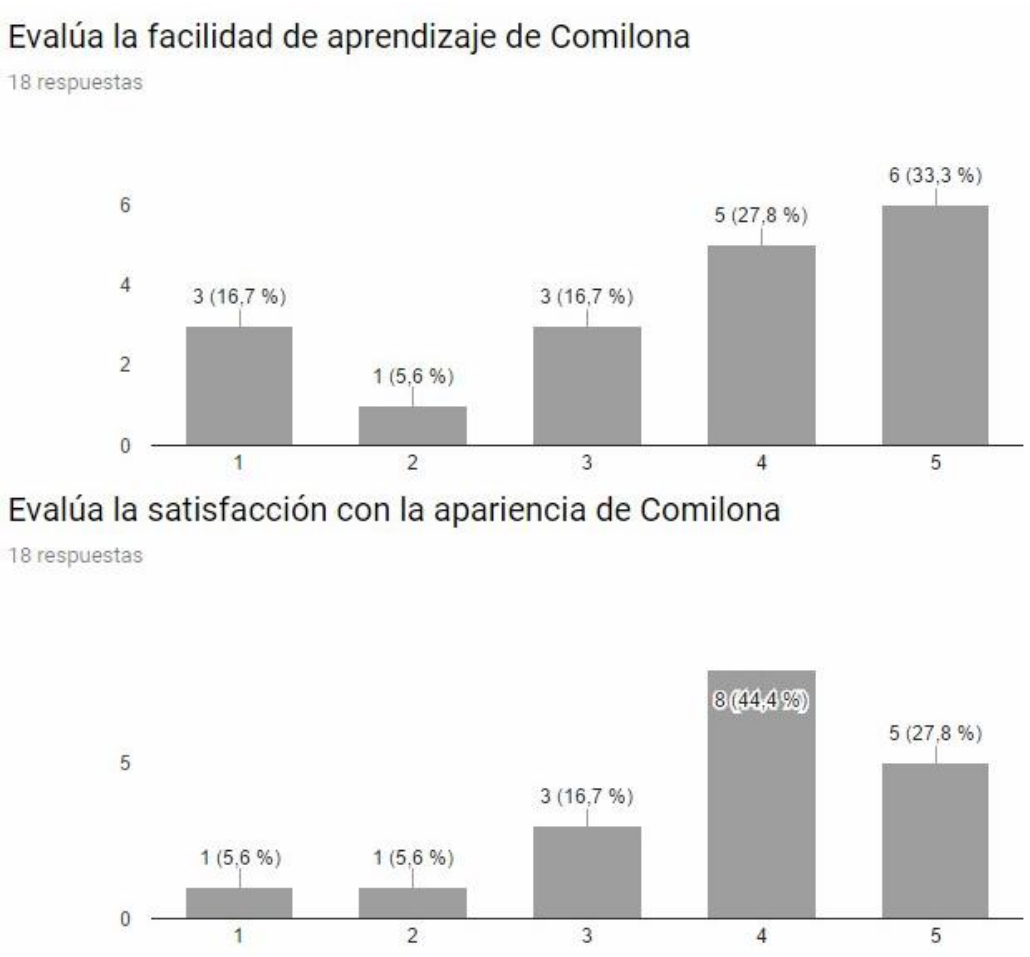


Figura 5.2.2: resultado 2 de preguntas

La última de las preguntas a destacar está relacionada con la valoración global de Comilona, en la que, siguiendo lo dispuesto en la figura 5.2.3, podemos observar que a la mayoría, a más de dos tercios, le ha gustado mucho, mientras que 4 alumnos piensan que no les ha gustado nada.

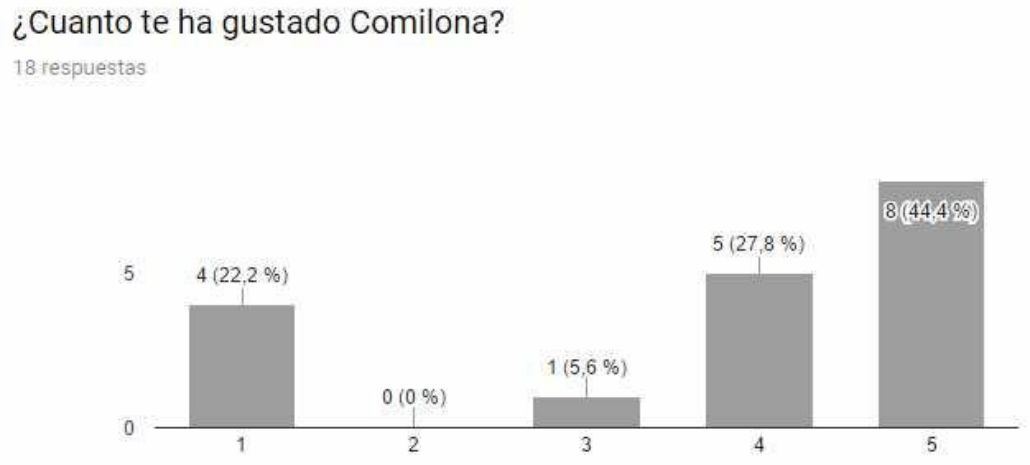


Figura 5.2.3: resultado 3 de preguntas

Además de las preguntas anteriores, también se realizaron otras de respuesta libres. Destacamos algunas de las respuestas a la pregunta 'explica con tus palabras lo que es para ti la Computación', ya que vemos algunas como 'Es una mezcla de seguridad, peligro, información, salud... Gracias a ella mucha gente avanza pero, a veces ese avance provoca riesgos', en la que observamos cierta madurez que no esperábamos en niños de 5º de primaria, u otras como 'Todo lo que está relacionado con los ordenadores, la informática, etc.', en la que vemos que no están muy perdidos en ese ámbito. Otra es la siguiente, 'Una manera de hacer las cosas sin moverte', en que se puede analizar como banalizan lo que realmente es, pero no se alejan de lo que nos permite realizar un ordenador.

En el caso de la pregunta 'Explica con tus palabras lo que crees que hacen los Programadores en su trabajo', se puede apreciar como muchos se acercan, con respuestas como 'Crear nuevos programas y juegos y ordenadores', 'Crear cosas como programas, páginas y facilitar la navegación por internet' o 'Crean las webs, las actualizan, las mejoran, las protegen. Básicamente hacen que internet tenga sentido', viendo que muchos reducen los ordenadores a Internet.

A la hora de realizar los test, nos dimos cuenta que sería mejor utilizar un vocabulario más sencillo, ya que palabras como interfaz o computación alguno de los niños no sabían a que se referían, mientras que en el caso de mencionar apariencia u ordenadores en general sí que se hacían una idea.

Capítulo 6. Conclusiones y líneas futuras

6.1 Conclusiones

El pensamiento computacional ha adquirido cierta notoriedad en el campo de la computación en los últimos años desde que en el 2006 Wing publicase el primer artículo sobre el tema. Esta idea se basa en el humano piense de una manera parecida a como lo haría un ordenador. Además, es innegable que el crecimiento en este campo nos lleva a buscar nuevas formas de enseñarlo, sobre todo a los más jóvenes.

Vimos en el comienzo de esta memoria que existen muchos trabajos en este campo, pero ninguno mezclaba la nutrición con el pensamiento computacional, solamente se realizaban distintos ejercicios utilizando la idea de la programación visual utilizando bloques. Cabe destacar que esa es la forma elegida en la mayoría de los trabajos, pero no es la única para explicar y enseñar sobre el pensamiento computacional, sin embargo sí que resulta muy sencillo y cómodo, gracias en parte a plataformas tan maduras como Scratch, la cual sentó precedentes.

A pesar de mencionar siempre al rango de edades de 8 a 12 años, con esta memoria y con el desarrollo de Comilona intentamos crear un modo de acercamiento de este campo a los más jóvenes, pero no cerrando las puertas directamente a otro rango de edades, ya que como vimos, los niveles se pueden adaptar con facilidad. Otra de las ideas que perseguimos con este trabajo era aplicarlo en un entorno real, y como pudimos comprobar, conseguimos nuestro objetivo en gran parte, ya que se esperaba que, al menos, aumentase el interés general por la computación y todo su campo.

Este trabajo sigue la filosofía de código abierto, por lo que se anima a cualquiera que lo desee a utilizarlo, modificarlo y mejorarlo como lo crea conveniente, siempre y cuando su código también siga esta filosofía.

6.2 Líneas futuras

Una vez realizado el desarrollo de Comilona, definimos unas líneas de actuación futuras, relacionadas con la mejora de la aplicación en general o con facilitar el uso de la misma. Se espera para un futuro:

- Despliegue en servidores: gracias al uso de tecnologías web, se espera poder desplegar la aplicación en un servidor y que los clientes accedan a la misma a través de un navegador, ya que en casos con muchos ordenadores puede resultar lioso tener una copia del juego en cada uno.
- Crear una aplicación para la personalización de niveles: a pesar de comentar que se trata de un juego, debido al carácter modular con el que se desarrolló, sería beneficioso crear una webapp o algún otro tipo de aplicación que nos permitiese crear y modificar nuestros propios niveles, de manera que se generasen los ficheros JSON de manera automática, en lugar de tener que modificarlos nosotros mismos.
- Empaquetado de la aplicación para Mac OS: para aumentar aún más la compatibilidad con los sistemas operativos, deberíamos buscar otro modo de poder cargar los ficheros de los ejercicios de manera que sea posible su uso en el sistema operativo Mac OS.
- Crear un ejecutable: para simplificar aún más el uso de la aplicación en general, se podría crear un ejecutable con Electron que se encargase de desempaquetar todos los ficheros en un directorio, como si de un programa tradicional se tratase, o incluso utilizar únicamente este ejecutable en modo portable, sin necesidad de copiar ficheros.

Capítulo 7.
Summary and Conclusions

Computational thinking has gained some fame in the field of computing in the last few years since the 2006, when Wing published the first article about this subject. This idea is based on the human think in a similar way as a computer would. Also, it is undeniable that growth in this field leads us to seek new ways of teaching it, especially to the youngest.

At the beginning of this report we saw that there are many works in this field, but none mixed nutrition and computational thinking, only different exercises using the idea of visual programming using blocks. It has to be emphasized that this is the form chosen in most of the works, but it is not the only one to explain and teach about computational thinking, however it is very simple and comfortable, thanks to platforms as mature as Scratch, which set precedents.

Although always mentioning the age range of 8 to 12 years, with this memory and with the development of Comilona we try to create a way of approaching this field to the youngest, but not closing the doors directly to another age range because as we have seen, levels can be easily adapted. Another idea that we pursued with this work was to apply it in a real environment, and as we could verify, we achieved our goal largely, since we hope that, at least, increase the general interest in computing and the whole field.

This work follows the philosophy of open source, which encourages anyone who wants to use it, modify it and improve it as it sees fit, as long as its code also follows this philosophy.

Capítulo 8. Presupuesto

Este capítulo recoge un modelo de presupuesto para el tiempo empleado en el desarrollo de este trabajo.

8.1 Presupuesto

Tareas	Horas	Presupuesto
Revisión bibliográfica	30h	5€/h
Estudio de antecedentes	30h	10€/h
Desarrollo de la aplicación	120h	20€/h
Pruebas de la aplicación	60h	15€/h
Prueba en entorno real	10h	15€/h
TOTAL:	**250h**	**3.900€**

Tabla 8.1.1: modelo de presupuesto

Como se puede apreciar en la tabla 8.1.1, el presupuesto total asciende a 3.900€ por 250 horas de trabajo.

Bibliografía

[1] J. M. Wing, "Computational Thinking," Communications of the ACM, vol. 49, no. 3, pp. 33-35, 2006.

[2] 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.

[3] 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.

[4] University of Cambridge, "EDSAC 99," [Online]. Available: http://www.cl.cam.ac.uk/events/EDSAC99/statistics.html.

[5] MIT Media Lab, "Scratch," [Online]. Available: https://scratch.mit.edu/.

[6] Code.org, "Hora de código," [Online]. Available: https://hourofcode.com/.

[7] Code.org, "Code.org," [Online]. Available: https://code.org/.

[8] M. T. Burns, M. Benoit and D. Bulvan, "Nutrition Jeopardy," Journal of Nutrition Education and Behavior, vol. 34, no. 2, pp. 117-118, 2002.

[9] 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.

[10] Davidap, "Acomola: alimentación equilibrada," Gobierno de Canarias, 12 Mayo 2015. [Online]. Available: http://www3.gobiernodecanarias.org/medusa/ecoescuela/recursosdigitales/2015/05/12/alimentacion-equilibrada/.

[11] V Congreso Internacional de Videojuegos y Educación, "Programa V Congreso Internacional de Videojuegos y Educación," [Online]. Available: http://eventos.ull.es/event_detail/7679/programme/v-congreso-internacional-de-videojuegos-y-educacion.html.

[12] GitHub, "Electron," [Online]. Available: https://electron.atom.io/.

[13] Joyent, Inc, "Node.js," [Online]. Available: https://nodejs.org/en/.

[14] Google, "Chromium," [Online]. Available: http://www.chromium.org/.

[15] Google, "Blockly," [Online]. Available: https://developers.google.com/blockly/.

[16] Wikimedia Foundation, Inc, "Blockly - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Blockly.

[17] Google, "Google App Engine," [Online]. Available: https://appengine.google.com.

[18] Google, "Blockly Developer Tools," [Online]. Available: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html.

[19] Google, "Blockly Games," [Online]. Available: https://blockly-games.appspot.com/.

[20] Massachusetts Institute of Technology, "MIT App Inventor," [Online]. Available: http://appinventor.mit.edu/explore/.

[21] Yahoo! Inc, "Pure," [Online]. Available: https://purecss.io/.

[22] GitHub, "Star-filtered repositories," 30 Mayo 2017. [Online]. Available: https://github.com/search?o=desc&q=stars%3A%3E1&s=stars&type=Repositories.

[23] The jQuery Foundation, "jQuery," [Online]. Available: https://jquery.com/.

[24] Q-Success, "W3Techs," [Online]. Available: https://w3techs.com/technologies/overview/javascript_library/all.

[25] N. Fraser, "JS-Interpreter on GitHub," [Online]. Available: https://github.com/NeilFraser/JS-Interpreter.

[26] Google, "Generating and Running Javascript," [Online]. Available: https://developers.google.com/blockly/guides/app-integration/running-javascript.

[27] chancejs, "Chance," [Online]. Available: http://chancejs.com/.

[28] Organización Mundial de la Salud, "Fomento del consumo mundial de frutas y verduras," [Online]. Available: http://www.who.int/dietphysicalactivity/fruit/es/.

[29] GitHub, "Rafa Herrero, tfg-epu," [Online]. Available: https://github.com/Rafaherrero/tfg-epu.

[30] Toptal Designers, "Subtle Patterns," [Online]. Available: https://www.toptal.com/designers/subtlepatterns/.

[31] Graphic Resources S.L., "Flaticon," [Online]. Available: http://www.flaticon.com/.