



INGENIERÍA EN SISTEMAS AUDIOVISUALES Y
MULTIMEDIA

Curso Académico 2017/2018

Trabajo Fin de Grado

EVALUACIÓN DE PROYECTOS APPINVENTOR

Autor : Roberto Nombela Alonso

Tutor : Dr. Gregorio Robles

Trabajo Fin de Grado

Evaluación de Proyectos AppInventor

Autor : Roberto Nombela Alonso

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de
de 2018, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2018

*Dedicado a
mi familia y, en especial, a Sofía.
Sin ella no habría llegado hasta aquí.*

Agradecimientos

En primer lugar quiero dar las gracias a mi familia. Que siempre ha estado tan pendiente de mí, apoyándome en todo momento, consolándome tras cada fallo y alegrándose de mis logros. Saber que puedo contar con vosotros cuando lo necesite me ha dado (y me da) fuerzas.

Por otro lado, agradecer a Ahmed, Álvaro y Ángel, mis compañeros de batalla. Por las tardes y noches de Skype, las prácticas interminables y los ejercicios irresolubles, pero siempre con buen humor. Si sobrevivo a esta carrera es gracias a vosotros.

Gracias a todos mis compañeros y amigos, no quiero decir nombres porque me dejaría a alguien y no sería justo. A los que se preocuparon por cómo me iba y me animaron a seguir, a los que tuvieron buenas palabras en momentos malos. En definitiva, a los que han estado.

Agradecer a Gregorio la oportunidad única que me ha brindado con la Beca de Colaboración y toda la ayuda que me ha proporcionado.

Por último pero no por ello menos importante, quiero dar las gracias a Sofía. Por todo. Desde que se vino a Madrid no he vuelto a repetir ninguna asignatura. Las casualidades no existen.

Resumen

Este Trabajo Fin de Grado consiste en una evaluación de proyectos del entorno AppInventor, cuyo código se compone de bloques. Durante el proceso, se analizan los distintos bloques que se emplean en los programas con el fin de obtener una visión general de lo que están desarrollando actualmente los usuarios de AppInventor.

Esta idea surge a partir de las herramientas que evalúan el Pensamiento Computacional en los proyectos, como Dr. Scratch o Code Master. Dichas herramientas son de utilidad para dar una puntuación pero no proporcionan suficiente *feedback* para que el usuario mejore. Por ello, el objetivo final es encontrar caminos de mejora de las habilidades del Pensamiento Computacional.

Para conseguirlo, se desarrolla la herramienta Dr. AppInventor, una aplicación web basada en Dr. Scratch para el análisis de proyectos AppInventor. La aplicación está escrita en Django, un framework para aplicaciones web de Python. Después, modificando el algoritmo de Dr. AppInventor, se extraen los bloques y las puntuaciones de un conjunto de proyectos, previamente extraídos de la galería, y se procede al análisis de los datos conseguidos.

Summary

This Thesis consists of an assessment of projects in the AppInventor environment, which code is composed of blocks. During the process, the different blocks used in the programs are analyzed in order to get an overview of what AppInventor users are currently developing.

This idea arises from tools that evaluate Computational Thinking in projects, such as Dr. Scratch or Code Master. These tools are useful for giving a score but do not provide enough feedback for the user to improve. Therefore, the goal is to find ways to improve Computational Thinking skills.

To achieve this, we developed the Dr. AppInventor tool, a Dr. Scratch-based web application for the analysis of AppInventor projects. The application is written in Django, a Python web applications framework. Then, modifying the Dr. AppInventor algorithm, the blocks and scores of a set of projects, previously extracted from the gallery, are extracted and the data obtained are analyzed.

Índice general

1. Introducción	1
1.1. Contexto	1
1.2. Motivación	2
1.3. Estructura de la memoria	3
2. Objetivos	5
2.1. Objetivo general	5
2.2. Objetivos específicos	5
3. Estado del arte	7
3.1. AppInventor	7
3.2. Python	8
3.3. Django	9
3.4. HTML	11
3.5. CSS	11
3.6. JavaScript	11
3.7. jQuery	12
3.8. Dr. Scratch	12
3.9. pandas	13
3.10. Jupyter Notebooks	14
4. Diseño e implementación	17
4.1. Dr. AppInventor como aplicación web	18
4.1.1. Extracción de bloques y componentes	18
4.1.2. Puntuación del proyecto	20

4.2.	Dr. AppInventor para múltiples proyectos	23
4.2.1.	Recopilación de proyectos	23
4.2.2.	Adaptación del algoritmo	24
4.3.	Interfaz de Dr. AppInventor	25
4.3.1.	Página de inicio	25
4.3.2.	Página de resultados	25
5.	Resultados	29
5.1.	Número de bloques	29
5.2.	Variedad en los proyectos	30
5.3.	Uso de bloques	30
5.3.1.	Familias de bloques	30
5.3.2.	Tipos de bloques	31
5.4.	Relación Puntuación/Variedad	33
5.5.	Variedad frente a cantidad	34
6.	Conclusiones	37
6.1.	Consecución de objetivos	37
6.2.	Aplicación de lo aprendido	38
6.3.	Lecciones aprendidas	39
6.4.	Trabajos futuros	39
A.	Manual de usuario	41
	Bibliografía	43

Índice de figuras

3.1. Elementos del modo Blocks.	8
3.2. Elementos del modo Designers.	9
3.3. Estructura MVC de Django.	10
3.4. Análisis de un proyecto en Dr. Scratch.	13
3.5. Código pandas en Jupyter Notebook.	15
4.1. Diagrama de bloques del proyecto.	17
4.2. Contenido de un fichero .bky.	19
4.3. Contenido de un fichero .scm.	20
4.4. Galería de AppInventor.	23
4.5. CSV con las puntuaciones (results.csv).	24
4.6. CSV con los bloques (blocks.csv).	25
4.7. Página de inicio de Dr. AppInventor.	26
4.8. Ayuda para descargar un proyecto AppInventor.	26
4.9. Página de resultados en Dr. AppInventor.	27
5.1. Distribución de la variedad.	31

Capítulo 1

Introducción

Este Trabajo Fin de Grado tiene como objetivo el desarrollo de una herramienta que facilite el análisis de proyectos creados con MIT AppInventor¹. La herramienta, llamada Dr. AppInventor, extrae y clasifica los bloques de cada proyecto y devuelve una puntuación en función de unos criterios que evalúan el desarrollo de Pensamiento Computacional, término sobre el que se cimienta este trabajo.

1.1. Contexto

En el año 2006, J. Wing presentó el concepto del Pensamiento Computacional [14] como un proceso de razonamiento que permite formular y resolver problemas de cualquier ámbito. Esto quiere decir que el Pensamiento Computacional no se restringe a las matemáticas o a la programación, sino que cualquier persona puede beneficiarse de pensar como un informático. El Pensamiento Computacional consta de una serie de aptitudes que permiten abarcar un problema: abstracción, pensamiento paralelo, pensamiento algorítmico, representación de datos.

Una manera de desarrollar dichas aptitudes es mediante la programación, que fomenta la creatividad y el uso de la lógica [7]. Programar implica plasmar pensamientos en forma de código y representar externamente el proceso de resolución de un problema. Los lenguajes de programación visual basados en bloques se caracterizan por ser realmente intuitivos, lo que los hace perfectos candidatos para mejorar las habilidades de las personas que no están del todo familiarizadas con la programación.

¹<http://ai2.appinventor.mit.edu/>

Consecuentemente, han aparecido herramientas que evalúan el desarrollo de Pensamiento Computacional de los aprendices a partir de los proyectos que han creado usando estos lenguajes y entornos de programación. Su objetivo es indicar los puntos fuertes y carencias del aprendiz para que pueda autoevaluarse y seguir aprendiendo. En la sección 4.1 se darán detalles de cómo evalúan. Estas herramientas han comenzado a utilizarse en centros educativos, de modo que profesores y alumnos pueden beneficiarse de la evaluación que ofrecen.

1.2. Motivación

Estudios recientes han descubierto que las herramientas de análisis han resultado ser más útiles para los profesores a la hora de corregir, que para los alumnos a la hora de mejorar [8]. No obstante, si queremos que dichas herramientas cumplan con una función más didáctica, hemos de dedicar mayor atención al proceso de aprendizaje. Por ello realizamos Dr. AppInventor, para tener una herramienta que permite la evaluación de proyectos AppInventor de manera automática. Con Dr. AppInventor hemos realizado un análisis de un gran número de proyectos para obtener una visión general de los proyectos que está desarrollando la comunidad. Con esta visión queremos encontrar posibles formas de ayudar al alumno a mejorar sus habilidades, ya sea mediante lecciones, consejos o ejemplos, con la idea de introducirlas en la herramienta.

Motivación Personal

Desde un punto de vista personal, hay tres elementos clave que han hecho que me decantara por este proyecto:

- Ayudar a otras personas a aprender: siempre me ha gustado compartir lo que sé con compañeros y amigos que lo han necesitado, buscando la mejor manera de explicarme para que lo comprendieran.
- Crear una aplicación web: por las comodidades que ofrece, en cuanto a compatibilidad, respecto a las aplicaciones de escritorio.
- Realizar un análisis de datos: la Ciencia de Datos es un campo en crecimiento y ya tenía interés en introducirme. Ha sido la excusa perfecta.

1.3. Estructura de la memoria

En esta sección se detalla la estructura de la memoria, la cual está dividida en 6 capítulos. Cada uno de estos capítulos se explica a continuación:

- El capítulo 1 es una Introducción al proyecto, donde se da a conocer el contexto en el que se enmarca, se explica la motivación por la cual se ha llevado a cabo y se muestra la estructura que sigue.
- En el capítulo 2 se muestra el objetivo principal del proyecto, junto con los subobjetivos necesarios para cumplirlo. Cada subobjetivo muestra el periodo de tiempo durante el que se realizó.
- Estado del arte: en este capítulo se exponen con una breve explicación las tecnologías existentes que se han utilizado para la realización del trabajo.
- Diseño e implementación: este capítulo explica detalladamente la creación de la herramienta Dr. AppInventor y la preparación del algoritmo para extraer los datos de los proyectos.
- El capítulo 5 muestra los resultados obtenidos tras analizar los datos extraídos en el capítulo anterior. Proporciona información sobre el número de bloques, la variedad, los tipos de bloques más y menos utilizados
- Conclusiones: aquí se detalla la consecución de los objetivos y los problemas que se han presentado durante la realización del trabajo. También se habla de los conocimientos previos aplicados, de los conocimientos que se han adquirido y de los posibles trabajos futuros.

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo principal de este proyecto es crear una herramienta que evalúe el desarrollo de Pensamiento Computacional de proyectos de MIT AppInventor. De esta manera se pueden buscar caminos de aprendizaje que permitan a los alumnos (o cualquier persona interesada en aprender) mejorar sus habilidades de Pensamiento Computacional utilizando AppInventor. Para ello, vamos a analizar los proyectos de la comunidad AppInventor, con el que obtendremos una visión general de los proyectos que se están desarrollando.

2.2. Objetivos específicos

En el proceso de alcanzar el objetivo general, ha sido necesario realizar antes unas tareas u objetivos específicos que se exponen a continuación, junto con el periodo de tiempo en el que se llevaron a cabo:

1. Finales de septiembre de 2017 - octubre de 2017: Documentación sobre el contexto en el que se desarrolla el trabajo: Pensamiento Computacional, lenguajes de programación visual basados en bloques y herramientas de análisis de programas en dichos lenguajes.
2. Octubre de 2017 - noviembre de 2017: Conocer la estructura de AppInventor: uso y funcionamiento, los tipos de bloques y componentes que pueden emplearse en los proyectos y los ficheros que forman un proyecto.

3. Noviembre de 2017 - febrero de 2018. Estudiar la aplicación de Dr. Scratch¹, que evalúa proyectos Scratch, para crear una adaptación que permita evaluar proyectos AppInventor. El desarrollo de Dr. AppInventor se explicará con detalle en el capítulo 4.
4. Febrero de 2018: Adaptar el algoritmo de Dr. AppInventor para extraer bloques y evaluar varios proyectos, guardando esta información en ficheros CSV.
5. Finales de febrero de 2018 - Marzo de 2018: Obtener de la galería de AppInventor una gran cantidad de proyectos.
6. Marzo de 2018 - abril de 2018: Analizar los CSV obtenidos con Dr. AppInventor utilizando la librería pandas en un Jupyter Notebook.

¹<http://drscratch.org/>

Capítulo 3

Estado del arte

En este apartado se describen las diferentes tecnologías que se han utilizado para llevar a cabo este proyecto.

3.1. AppInventor

AppInventor [15] es un entorno de programación visual que permite crear aplicaciones móviles para Android. Surge para ayudar a las personas a pasar del consumo de tecnología a la creación. El “código” que utiliza son bloques que los usuarios pueden arrastrar y soltar, para unirlos entre sí como si fuera un *puzzle*.

Este entorno utiliza dos modos para la creación de una aplicación: *Blocks* y *Designer*. En las figuras 3.1 y 3.2¹, se muestran las pantallas de ambos modos con los diferentes elementos que los conforman.

Blocks: en este modo se define el comportamiento de la aplicación, es decir, se implementa el algoritmo. Hay 8 tipos diferentes de bloques: Control, Logic, Math, Text, Lists, Colors, Variables y Procedures. Además, se pueden ver bloques específicos de componentes. Estos componentes deben haberse seleccionado previamente en el modo Designer. El aspecto visual del modo “blocks” se puede encontrar en la figura 3.1.

Designer: en este modo se seleccionan los componentes que van a constituir la aplicación. Estos componentes pueden ser visuales (los que formarán la interfaz de usuario) o no visuales (diversas funcionalidades del dispositivo como el acelerómetro o Bluetooth, bases de datos o

¹Fuente: <http://explore.appinventor.mit.edu/designer-blocks>

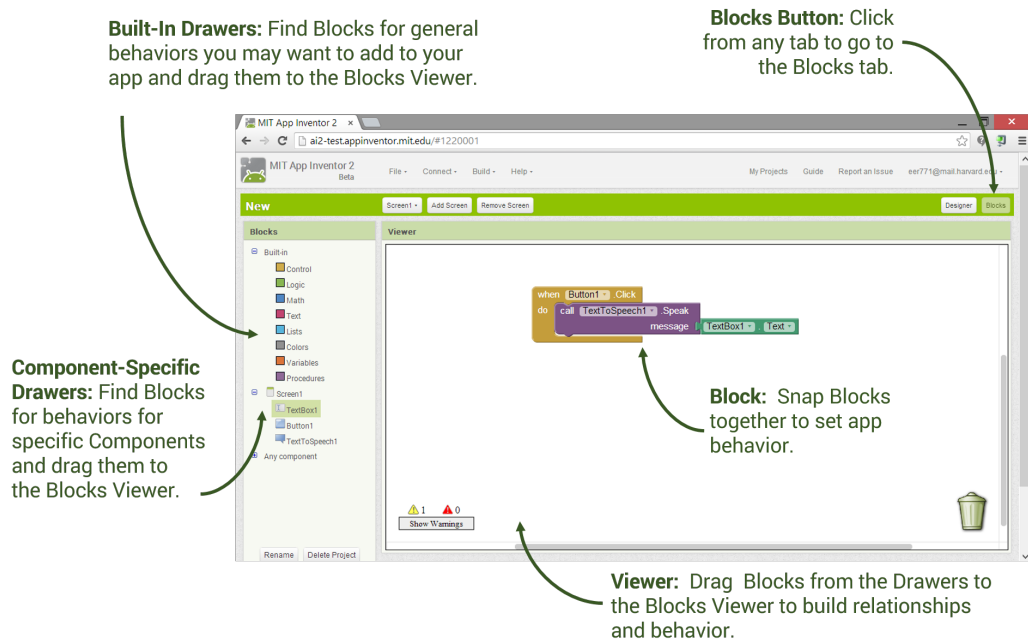


Figura 3.1: Elementos del modo Blocks.

comandos que permiten controlar robots LEGO® MINDSTORMS® NXT). Se puede ver un ejemplo del modo “designer” en la figura 3.2.

3.2. Python

Python es un lenguaje de programación interpretado, interactivo y orientado a objetos [3]. Su creador, Guido van Rossum, explicó que Python surgió como un “hobby” para mantenerse ocupado durante las Navidades de 1989, basándose en el lenguaje ABC, con el pensamiento de que esto atraería a los programadores de Unix/C [4].

Entre sus características destacan el gran número de módulos que posee, la posibilidad de incluir código C o C++ para aumentar la velocidad, la portabilidad (Windows, Mac y algunas variantes de Unix) pero, sobre todo, destaca por su filosofía:

Esta filosofía, conocida como “El Zen de Python”, fue escrita por Tim Peters. Forma parte de las Propuestas de Mejoras de Python (PEP 20²) y aboga por que el código sea:

- Simple: “Simple is better than complex”.

²<https://www.python.org/dev/peps/pep-0020/>

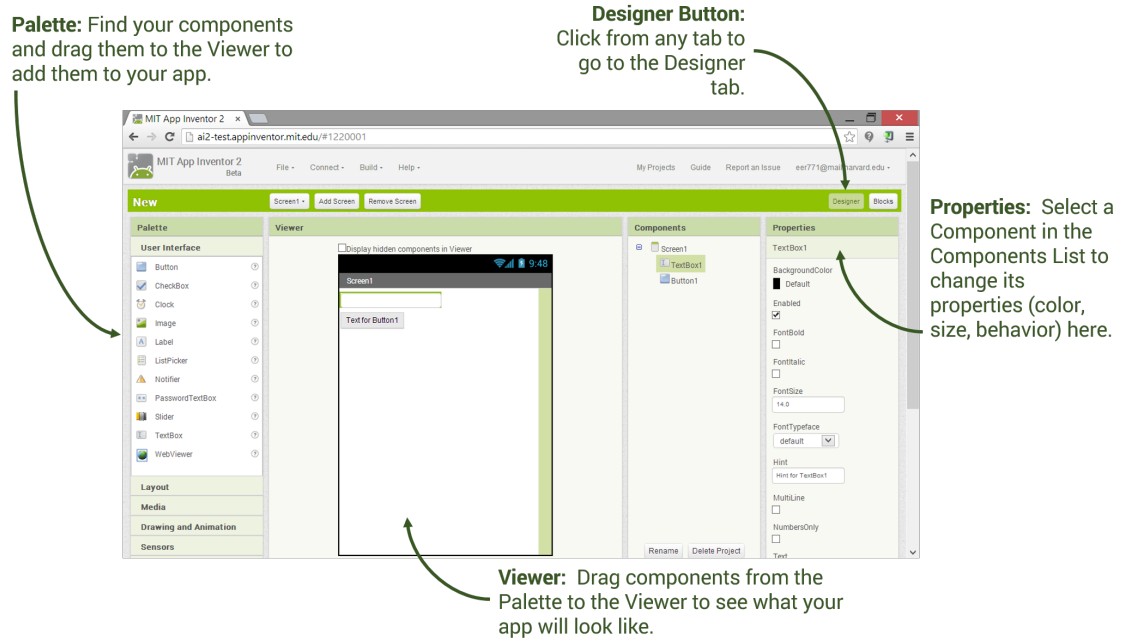


Figura 3.2: Elementos del modo Designers.

- Legible: “Readability counts”.
- Claro: “If the implementation is hard to explain, it’s a bad idea”.

3.3. Django

Django³ es un framework para aplicaciones web, escrito en Python. Permite un desarrollo rápido, con un diseño limpio y práctico gracias a su filosofía DRY (Don’t Repeat Yourself). Debido a esto, destaca por tener un número reducido de ficheros y carpetas. En la Tabla 3.1, se observa una comparación con otros framework [1].

	Rails	Symfony	Django
Ficheros	149	117	5
Carpetas	35	29	2

Tabla 3.1: Ficheros y carpetas en Django vs. otros frameworks.

³<https://www.djangoproject.com/>

Se basa en el patrón MVC (Modelo-Vista-Controlador). El modelo se define en `models.py` y contiene la información de los datos que maneja el servidor. El controlador está formado por dos scripts: `urls.py` que, dependiendo de la petición HTTP, invoca a una vista o a otra y `views.py` que genera la respuesta HTTP, consultando al modelo si es necesario. La vista es la respuesta generada por el controlador, comúnmente es el HTML de una página web.

Otras ventajas que presenta Django son el ORM y el lenguaje de plantillas:

- ORM (Object-Relational Mapping): realiza un mapeo de diferentes bases de datos, de modo que el usuario no necesita conocer los comandos de SQLite o MySQL, ya que puede manejar las bases de datos empleando código de Python.
- Lenguaje de plantillas: permite introducir variables y lógica simple en un HTML del siguiente modo: `{{ variable }}`, `{% código %}`. Para el uso de variables, en el render se pasa como argumento el contexto: un diccionario clave-valor, donde la clave es el nombre que va a tomar la variable dentro de la plantilla y el valor es la variable en sí.

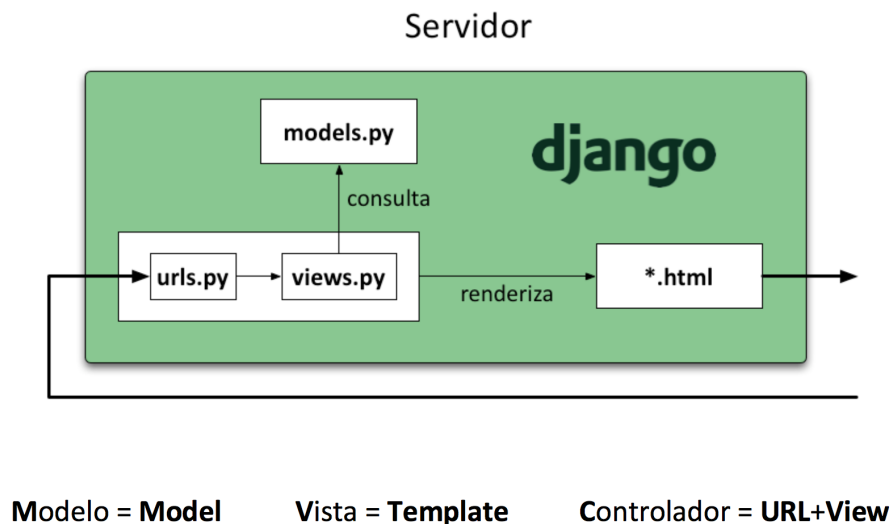


Figura 3.3: Estructura MVC de Django.

3.4. HTML

HTML (HyperText Markup Language) [11], es el lenguaje principal de marcado de la web. Inicialmente fue planteado para describir semánticamente documentos científicos. Actualmente, debido a su diseño, se ha adaptado para describir otros tipos de documentos e incluso aplicaciones. El uso más conocido de HTML es la creación y representación visual de páginas web. Este lenguaje define la estructura de una página, pero no su funcionalidad.

Los documentos HTML tienen estructuras de árbol de elementos y texto. Cada elemento se denota en el código con etiquetas de inicio, junto con las que encontramos los atributos, y etiquetas de fin. El contenido se encuentra entre las dos etiquetas. Un ejemplo sería:

```
<etiqueta atributo="valor"> contenido <etiqueta>.
```

Se complementa con otras tecnologías para modificar la apariencia de la página (CSS) o para determinar la funcionalidad (JavaScript).

3.5. CSS

CSS (Cascade Style Sheets) [10] son las siglas de hojas de estilo en cascada. Es el lenguaje que describe la presentación de una página web: colores, diseño y fuentes. Permite adaptar la presentación a distintos dispositivos, como pantallas de ordenador, móviles o tablets. Se puede utilizar con cualquier lenguaje de marcado basado en XML pero sobre todo se utiliza junto con HTML.

Para dar estilo, se referencia el elemento deseado y se incluyen los atributos, por ejemplo:

```
elemento {  
  color: "color";  
}
```

3.6. JavaScript

JavaScript es un lenguaje de programación desarrollado por Netscape que, originalmente, se llamó LiveScript. En 1995, como Java estaba en crecimiento, por razones de marketing comenzó a llamarse JavaScript, aunque los lenguajes no están relacionados entre sí [2].

JavaScript es interpretado, basado en prototipos, dinámico y débilmente tipado. Está compuesto de sentencias, comandos que ejecutará el navegador, separadas con punto y coma (;). Se caracteriza por ser el lenguaje del lado cliente en las aplicaciones web y se suele utilizar para: interacción con el navegador o con el documento HTML, para abrir conexiones con el servidor (AJAX) o para dibujar en la página (canvas).

3.7. jQuery

jQuery es una librería de JavaScript rápida, ligera y rica en funciones con la idea de *escribir menos y hacer más* [5]. El propósito es facilitar el uso de JavaScript en los sitios web. Para ello, jQuery utiliza métodos de una sola línea que realizan tareas que con JavaScript ocuparían varias. También simplifica muchas de las funcionalidades que son más complicadas con JavaScript, como las llamadas AJAX o la manipulación de los elementos de un HTML.

Entre sus prestaciones destacamos: manipulación de HTML o CSS; manejo de eventos; desarrollo de animaciones; interacción con AJAX.

La Tabla 3.2 muestra la reducción que proporciona jQuery respecto a JavaScript:

JavaScript	jQuery
<pre>function changeBackgroundColor(color){ document.body.style.background = color;} onload = "changeBackgroundColor('red');"</pre>	<pre>\$('body').css('background', '#ccc')</pre>

Tabla 3.2: Comparación del mismo código en JavaScript *a secas* y utilizando la biblioteca jQuery.

3.8. Dr. Scratch

Antes de empezar a implementar la aplicación web Dr. AppInventor, fue precisa una comprensión de lo que se pretendía desarrollar. Esta aplicación está basada en Dr. Scratch⁴ que realiza una evaluación del Pensamiento Computacional en proyectos Scratch⁵. Dr. Scratch ob-

⁴<http://www.drscratch.org/>

⁵<https://scratch.mit.edu/>

tiene una puntuación en función de los siguientes aspectos: abstracción, pensamiento lógico, sincronización, paralelismo, control de flujo, interactividad con el usuario y representación de la información. Cada uno se valora del 0 al 3 y la suma de todos es la puntuación final [6]. Además de la puntuación, ofrece al usuario *feedback* para mejorar dichos aspectos y, dependiendo de la puntuación, resalta malos hábitos de programación como puede ser código repetido o código que nunca se llega a ejecutar. Los usuarios tienen la opción también de descargar un diploma con los resultados de su proyecto. En la figura 3.4 se puede ver un ejemplo de análisis de un proyecto.



Figura 3.4: Análisis de un proyecto en Dr. Scratch.

3.9. pandas

Es una librería de código abierto nacida en 2008 [13]. Proporciona estructuras de datos de alto rendimiento y fáciles de usar, además de herramientas de análisis de datos. Su objetivo es ser el componente fundamental de alto nivel para realizar análisis de datos prácticos y reales en Python.

Algunos de los elementos que forman pandas son:

- Estructuras de datos etiquetadas: Series (matriz unidimensional capaz de contener cualquier tipo de datos) y DataFrame (estructura bidimensional, puede verse como un conjunto de Series). Los DataFrames son los más comúnmente utilizados en pandas.
- Objetos que permiten tanto la indexación de ejes simples como la indexación de ejes jerárquicos/multiniveles.
- Herramientas para agregar y transformar conjuntos de datos.
- Herramientas de entrada/salida: carga de tablas desde archivos planos (CSV, delimitados, Excel 2003), y almacenamiento y carga de objetos pandas desde el formato PyTables/HDF5.
- Combinación con otros paquetes como NumPy (computación científica) y matplotlib (representación de gráficos).

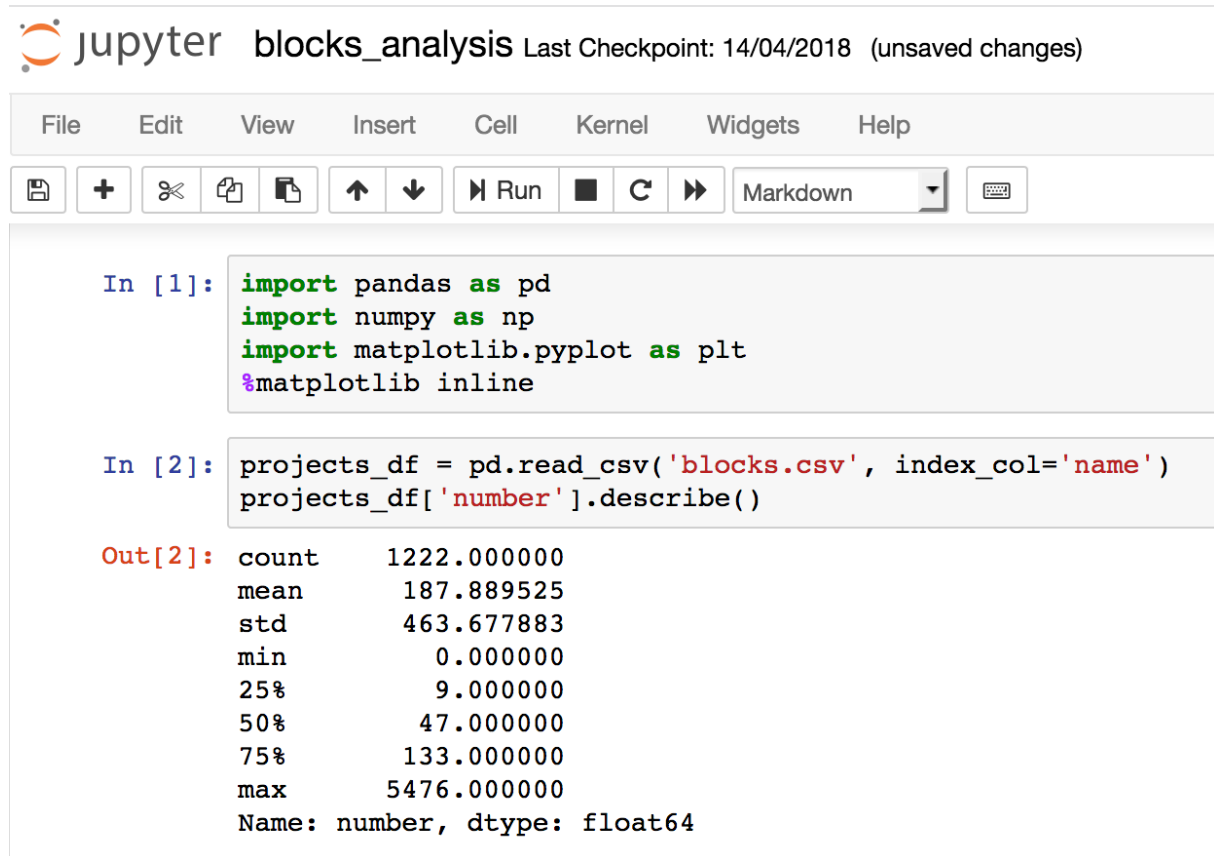
3.10. Jupyter Notebooks

Junto con pandas, se suele utilizar Jupyter Notebook⁶. Este “cuaderno” es una aplicación web que ofrece una interfaz al proceso de programación: desarrollo, documentación y ejecución, además permite visualizar los resultados cómodamente [12]. A continuación, se muestran algunas características de la aplicación:

- Edición del código en el navegador, con resaltado de sintaxis, indentación y completación de tabulaciones.
- Posibilidad de ejecutar código desde el navegador y mostrar los resultados de los cálculos bajo el código ejecutado.
- Visualización de resultados utilizando representaciones multimedia, como HTML, LaTeX, PNG, SVG, etc. Por ejemplo, las figuras renderizadas con la librería matplotlib.
- Capacidad de incluir fácilmente notación matemática utilizando LaTeX, renderizado de forma nativa por MathJax.

⁶http://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html

La figura 3.5 es un ejemplo de instrucciones de pandas en un Jupyter Notebook.



The screenshot shows a Jupyter Notebook window titled 'blocks_analysis' with a last checkpoint of '14/04/2018' and '(unsaved changes)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for saving, adding cells, undo, redo, up/down arrows, running, and clearing. The notebook content consists of two input cells and one output cell.

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
In [2]: projects_df = pd.read_csv('blocks.csv', index_col='name')
projects_df['number'].describe()
```

```
Out[2]: count      1222.000000
mean         187.889525
std          463.677883
min           0.000000
25%           9.000000
50%          47.000000
75%         133.000000
max         5476.000000
Name: number, dtype: float64
```

Figura 3.5: Código pandas en Jupyter Notebook.

Capítulo 4

Diseño e implementación

En este capítulo se explican los pasos necesarios para poder llevar a cabo el proyecto, que ya se mencionaron brevemente en la sección 2.2. La figura 4.1 representa un diagrama de bloques con los hitos seguidos.

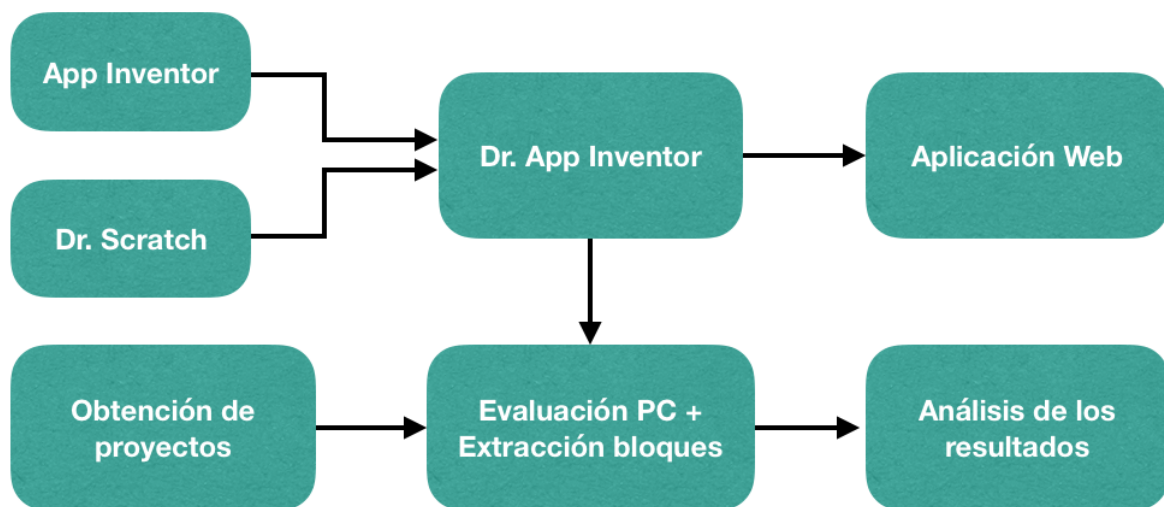


Figura 4.1: Diagrama de bloques del proyecto.

- AppInventor/Dr. Scratch: Antes de comenzar a implementar Dr. AppInventor, un paso previo fue la familiarización con Dr. Scratch y AppInventor. Estas dos tecnologías fueron explicadas en el capítulo 3. A diferencia de Dr. Scratch, en el caso de Dr. AppInventor únicamente se realiza la evaluación de los proyectos ya que el análisis se centra en el Pensamiento Computacional. No se proporciona *feedback* para cada uno de los criterios

ni tampoco se informa de las malas prácticas.

- Dr. AppInventor como aplicación web: la implementación de la aplicación desde la extracción de bloques y componentes hasta la evaluación de los proyectos. La implementación se explica con detalle en la sección 4.1.
- Recopilación de proyectos: qué proyectos se han utilizado para el análisis y cómo se han conseguido.
- Dr. AppInventor para múltiples proyectos: modificar el algoritmo de Dr. AppInventor para analizar los proyectos obtenidos. La sección 4.2 detalla esta modificación.
- Análisis de los resultados: se realiza un estudio de los bloques con los datos proporcionados por el algoritmo. En el capítulo 5 se desarrolla este análisis.

4.1. Dr. AppInventor como aplicación web

Una vez conocido el funcionamiento de Dr. Scratch y de AppInventor, se comienza con el proyecto en Django que dará lugar a la aplicación web.

4.1.1. Extracción de bloques y componentes

El algoritmo comienza con la extracción de las carpetas y ficheros que forman un programa en AppInventor, para ello hay que descomprimir el archivo del programa como si fuera un ZIP.

A continuación se puede ver en forma de árbol la estructura de los directorios obtenidos:

```

project
├── assets
│   └── MultimediaFiles
├── src
│   ├── appinventor
│   │   ├── ai_UserName
│   │   │   └── ProjectName
│   │   │       ├── Screen1.bky
│   │   │       ├── Screen1.scn
│   │   │       ├── ...
│   │   │       ├── ScreenN.bky
│   │   │       └── ScreenN.scn
└── youngandroidproject
    └── project.properties

```

La carpeta *assets* contiene los archivos multimedia (imágenes, audios y vídeos) que utiliza la aplicación.

La carpeta *youngandroidproject* contiene el fichero con las propiedades de la aplicación, como por ejemplo: el nombre del proyecto en AppInventor, el nombre de la app o la versión.

La carpeta de verdadero interés para el análisis es *src*, dentro de la cual están los ficheros *.bky* y *.scm*, cada una de las pantallas de la aplicación está formada por estos dos ficheros, cuyo nombre coincide con el nombre de la pantalla:

- **bky**: este fichero contiene, en forma de XML, los bloques que componen el programa, sus propiedades y cómo se relacionan entre sí. Para extraer los bloques, el algoritmo tendrá que buscar todas las etiquetas *block* y almacenar de cada una el atributo *type*. Podemos ver un ejemplo de este tipo de fichero en la figura 4.2. En este ejemplo, el primer bloque que se almacenaría lo vemos en la segunda línea: *component_event*.

```

1 <xml xmlns="http://www.w3.org/1999/xhtml">
2 <block type="component_event" id="P9b1W.)E[czeHf@9j(Dp" x="-745" y="1202">
3 <mutation component_type="Form" instance_name="Settings" event_name="Initialize">
4 <field name="COMPONENT_SELECTOR">Settings</field>
5 <statement name="D0">
6 <block type="controls_if" id="r[+]t+,,)dQ(aGC3A`t04">
7 <mutation elseif="1"></mutation>
8 <value name="IF0">
9 <block type="logic_compare" id="TK+S(i)YBPYlt$u+9.e/R">
10 <field name="OP">EQ</field>
11 <value name="A">
12 <block type="component_method" id="ingS8|ByEsMZTgQyRR="{
13 <mutation component_type="TinyDB" method_name="GetValue" is_generic="
14 <field name="COMPONENT_SELECTOR">TinyDB1</field>
15 <value name="ARG0">
16 <block type="text" id="Azmw$Q;5/f{y9u7fude8">
17 <field name="TEXT">fl_mess</field>
18 </block>
19 </value>
20 <value name="ARG1">
21 <block type="text" id="f-XfU+eD)+Q^9fPT$jT2">
22 <field name="TEXT"></field>
23 </block>
24 </value>
25 </block>
26 </value>
27 <value name="B">
28 <block type="text" id="~HjQy7(K9)4JQU{ozj1_">
29 <field name="TEXT">1</field>
30 </block>
31 </value>
32 </block>
33 </value>
34 <statement name="D00">

```

Figura 4.2: Contenido de un fichero *.bky*.

- **scm**: de manera similar al fichero *.bky*, aquí encontramos, con estructura JSON, los com-

ponentes y sus propiedades. Para extraer los componentes, el algoritmo busca todos los elementos con clave *\$Name* y almacena su valor, excepto del primer elemento ya que no es un componente en sí, sino que representa los ajustes (“Settings”) de la aplicación. La figura 4.3 es un ejemplo de este tipo de fichero. El primer componente en el ejemplo se encuentra en la línea 5: `HorizontalArrangement1`.

```
#|
$JSON
{"authURL":["ai2.appinventor.mit.edu"],"YaVersion":"159","Source":"Form","Properties":
{"$Name":"Settings","$Type":"Form","$Version":"20","AppName":"MultifunctionalAplicacion
[{"$Name":"HorizontalArrangement1","$Type":"HorizontalArrangement","$Version":"3","Ali
[{"$Name":"Label1","$Type":"Label","$Version":"4","FontBold":"True","FontSize":"16.0",
{"$Name":"VerticalArrangement1","$Type":"VerticalArrangement","$Version":"3","Backgrou
[{"$Name":"HorizontalArrangement2","$Type":"HorizontalArrangement","$Version":"3","Ali
[{"$Name":"Label2","$Type":"Label","$Version":"4","FontTypeface":"3","HTMLFormat":"True
{"$Name":"Button1","$Type":"Button","$Version":"6","BackgroundColor":"&H00FFFFFF","Hei
{"$Name":"Notifier1","$Type":"Notifier","$Version":"4","Uuid":"1800901659"},{"$Name":"
|#
```

Figura 4.3: Contenido de un fichero .scm.

4.1.2. Puntuación del proyecto

La fase de clasificación comienza cuando se han extraído todos los bloques y componentes. Para clasificar (y, posteriormente, puntuar) nos basamos en la rúbrica de Code Master¹ [9], una herramienta que analiza proyectos Snap! y AppInventor. Los criterios para obtener la puntuación son los siguientes:

Screens: hay que tener en cuenta tres cosas: el número de pantallas, si las pantallas contienen componentes visuales o no y si las pantallas cambian de estado programáticamente. Para obtener el número de pantallas, contamos el número de ficheros .bky (también se podrían contar los .scm y obtendríamos el mismo resultado). Si queremos ver si una pantalla contiene componentes visuales, tenemos que comprobar si alguno de sus componentes están en la lista de visuales. Por último, si la pantalla contiene cualquier bloque se puede decir que, potencialmente, cambiará de estado.

User Interface: la puntuación de este criterio depende del número de componentes visuales en la aplicación y también de las disposiciones (arrangements). Contamos los componentes visuales comparándolos con la lista mencionada en el punto anterior. Las disposiciones son un componente más en la aplicación, pueden llamarse: `HorizontalArrangement`, `Horizontal-`

¹http://apps.computacaonaescola.ufsc.br:8080/rubrica_appinventor.jsp

ScrollArrangement, VerticalArrangement, VerticalScrollArrangement y TableArrangement. Por lo que para contarlas con el algoritmo, buscamos cualquier componente cuyo tipo contenga Arrangement en su nombre.

Naming: este criterio evalúa si el nombre de una variable, componente o procedimiento se ha cambiado o, por el contrario, se ha dejado el nombre por defecto. En el caso de las variables y los procedimientos, los nombres serían: name/procedure, name2/procedure2, . . . Mientras que el nombre por defecto de los componentes depende del tipo, por ejemplo, si el tipo es Ball: Ball1, Ball2, etc. La puntuación dependerá del porcentaje total de nombres por defecto.

Events: la puntuación de los eventos depende del número de manejadores que se empleen. Para contabilizarlos con el algoritmo, hay que localizar todos los bloques del tipo component_event.

Procedural Abstraction: en este criterio se comprueba el uso de procedimientos (procedures). Hay dos maneras de crear procedimientos: procedures_defreturn y procedures_defnoreturn. La máxima puntuación se da cuando hay más llamadas a procedimientos que procedimientos. Esto lo comprobamos comparando el número de bloques mencionados anteriormente con los bloques de llamada: procedures_callreturn y procedures_callnoreturn.

Loops: para puntuar el uso de bucles hay que localizar los bloques: controls_while, controls_forRange, controls_forEach. Si solamente se utilizan while, tendrá puntuación 1. Si se utilizan forRange (con o sin bloques while), la puntuación será 2. Por último, si se encuentran forEach, obtendrá puntuación 3.

Conditional: a diferencia de la rúbrica de Code Master, los condicionales se evalúan comprobando el uso de if-then, if-then-else y de if-then-elseif. Para ello se buscan los bloques de tipo controls_if y controls_choose, este último contabiliza como if-then-else. Para los controls_if, hay que tener en cuenta la etiqueta hija “mutation” donde podemos encontrar los atributos else o elseif. Si no encontramos ninguno de estos, contabiliza como un if-then. La puntuación es una suma: uso de if-then, uso de if-then-else y uso de if-then-elseif, donde cada uno suma uno.

Math and Logic Operations: el algoritmo tiene que hallar los bloques que contengan math o logic en su tipo. La puntuación depende de cuántos bloques de estos tipos se utilicen.

Lists: para la evaluación de este criterio se comprueba el número de listas creadas (bloque lists_create_with). Para conseguir la puntuación máxima, es necesario utilizar una lista de tuplas, lo que en AppInventor se traduce como una lista de listas o también el uso de

lists_lookup_in_pairs.

Data Persistence: la persistencia de datos se puntúa en función de dónde se guardan los datos. Si es en variables o componentes de la interfaz gráfica, no puntúa. Si se almacenan File o FusionTables, puntúan 1. Si los datos se guardan en TinyDB, 2. Por último, si se utilizan TinyWebDB o FirebaseDB se obtendrá una puntuación de 3. Todas estas formas de almacenamiento se localizan en el .scm como componentes.

Sensors: los sensores pueden ser: acelerómetro, lector de código de barras, reloj, giroscopio, sensor de localización, NFC, sensor de orientación, podómetro y sensor de proximidad. La puntuación depende del número de tipos de sensores diferentes que utilice la aplicación, siendo 0 no utilizar ningún sensor y 3 utilizar más de dos tipos.

Media: los componentes multimedia son los siguientes: grabación de vídeo, cámara, selector de imagen, reproductor de audio, grabadora, reconocimiento de voz, texto a voz, reproductor de vídeo y traductor Yandex. La puntuación es análoga a la puntuación los sensores.

Social: los componentes sociales son: selector de contacto, selector de e-mail, llamada, selector de número de teléfono, compartir, enviar mensaje, compartir vía Twitter. La puntuación es análoga a los dos puntos anteriores.

Connectivity: en este criterio se evalúa el uso de los componentes de conectividad. Si no se utiliza ninguno, la puntuación es 0. Si se utiliza el componente activity starter puntúa 1. Usar componentes bluetooth (tanto cliente como servidor) puntúa 2. Para obtener una puntuación de 3, se ha de usar el componente web.

Drawing and Animation: para evaluar los componentes de dibujo y animación se tiene en cuenta el uso de estos componentes: canvas (1 punto), bola (2 puntos) e imagen interactiva (3 puntos).

En la aplicación web, el usuario selecciona el proyecto que desea evaluar y lo sube. La aplicación realiza el análisis de los bloques y componentes siguiendo estos criterios y cuando se tienen todos puntuados, se muestra la página de resultados, similar a la de Dr. Scratch. Más adelante, en la figura 4.9 veremos la evaluación de un proyecto con nivel alto.

4.2. Dr. AppInventor para múltiples proyectos

Ya tenemos el algoritmo que extrae los bloques, los clasifica y devuelve una evaluación de un proyecto, ahora queremos que el algoritmo extraiga los bloques y realice la evaluación varios proyectos a la vez para posteriormente analizar los datos obtenidos.

4.2.1. Recopilación de proyectos

Para obtener los proyectos hay que acceder a la galería de AppInventor (puede verse en la figura 4.4) y seleccionar los proyectos deseados. En nuestro caso, tenemos 62 proyectos de la sección “Popular” y 1.160 de la sección “Recent”, por orden de aparición.

Para descargar los proyectos es necesario iniciar sesión con una cuenta de Gmail, acceder a la galería, copiar el proyecto al perfil propio y, por último, descargarlo. El proceso se trató de automatizar lo máximo posible, tanto como la web de AppInventor permitió. Con un script se copiaban los proyectos de la galería al perfil y, como no se pueden descargar todos los proyectos del perfil a la vez, empleamos otro script para descargarlos uno a uno.

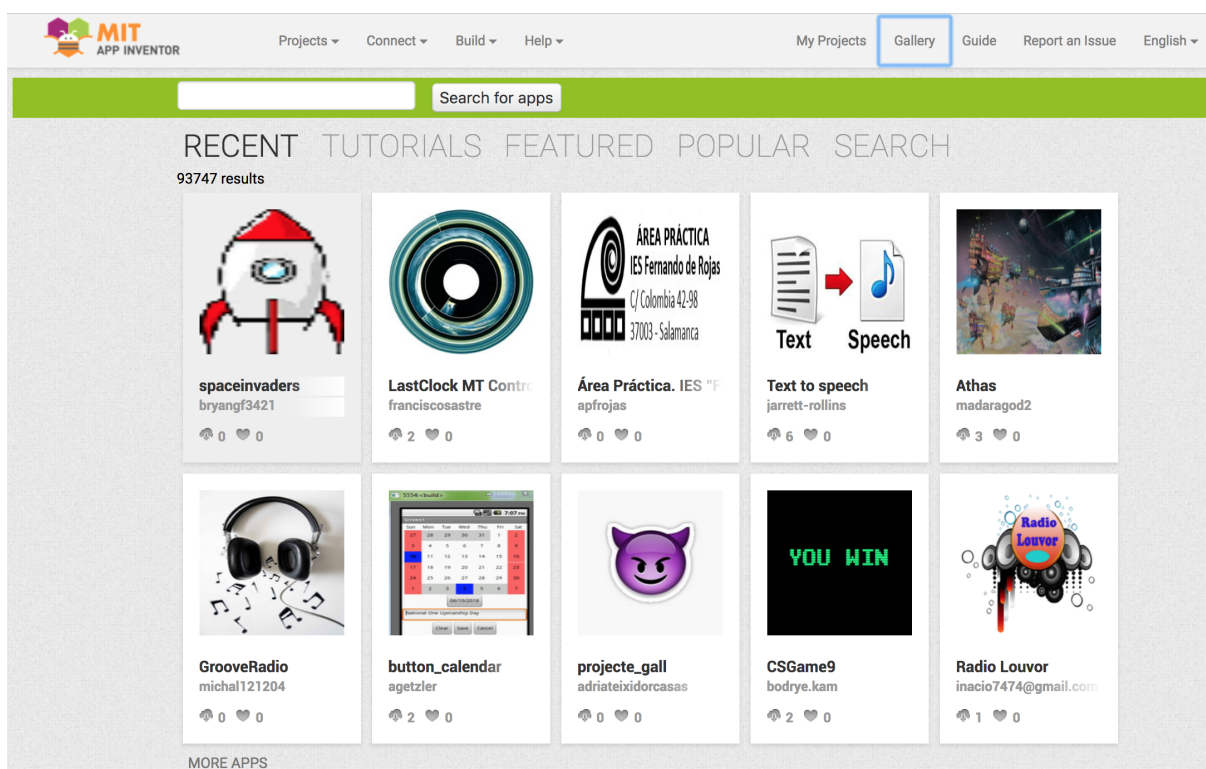


Figura 4.4: Galería de AppInventor.

4.2.2. Adaptación del algoritmo

El nuevo algoritmo analiza cada uno de los proyectos obtenidos, situados en un mismo directorio, de la misma manera que antes, pero ahora se guardan las puntuaciones de los criterios y la puntuación total en forma de CSV (figura 4.5).

```
Catchtheball,1,2,0,3,0,3,1,0,0,0,0,0,2,3,1,16
Delorean_Project,1,3,3,3,0,0,0,2,1,0,0,2,0,3,2,20
MFApp,3,0,3,3,0,0,2,3,1,0,0,0,0,3,3,21
PingPong,2,2,2,3,0,3,1,0,1,1,0,0,3,3,2,23
pop0318204010,3,1,0,1,0,0,0,0,0,0,0,0,1,0,0,1,7
pop0318204115,2,2,0,1,0,0,0,0,0,0,0,0,0,0,0,0,5
pop0318204143,1,2,1,2,0,0,0,2,1,0,0,1,0,3,2,15
pop0318204158,2,2,1,2,0,0,0,2,1,0,0,1,0,3,3,17
pop0318204224,1,2,1,3,0,1,0,0,1,0,0,3,0,0,2,14
pop0318204405,1,2,0,0,0,0,0,0,1,2,0,0,3,0,2,11
pop0318204439,3,2,3,3,0,2,3,2,1,2,0,3,0,3,3,30
pop0318204636,2,2,0,1,0,0,0,0,0,0,0,0,0,0,0,0,5
pop0318204755,1,2,0,1,0,0,0,0,0,1,0,0,0,0,0,0,5
pop0318210018,1,2,0,1,0,0,0,0,0,0,0,0,0,1,0,5
pop0318210240,1,1,0,2,0,0,0,0,0,1,0,0,0,0,1,6
pop0318210328,3,2,2,3,3,3,2,2,2,3,0,1,3,3,3,35
pop0318210537,1,0,0,0,0,0,0,0,0,0,0,0,0,0,3,4
pop0318210607,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,3
pop0318210652,1,2,0,1,0,0,1,0,1,2,0,0,0,2,0,10
```

Figura 4.5: CSV con las puntuaciones (results.csv).

Además, se crea un nuevo diccionario que tiene como clave los tipos de bloques que se emplean en el proyecto y el número de apariciones en los .bky como valor. Este diccionario se adapta para almacenarlo como otro CSV (figura 4.6) para el posterior análisis de ambos ficheros.

name	aia	number	math_subtract	color_green	math_multiply	co
catchtheball	Catchtheball	38	1	1	2	
Delorean	Delorean_Project	92	0	3	0	
Multifunctionalapplication	MFAApp	428	0	0	0	
Pong	PingPong	229	5	0	0	
Kids, STOP!	pop0318204010	108	0	0	0	
Chat	pop0318204115	3	0	0	0	
no_name	pop0318204143	63	0	0	0	
FindThatThing	pop0318204158	63	0	0	0	
no_name	pop0318204224	35	0	0	0	
CREEPER FUN 2.0 DEMO	pop0318204405	0	0	0	0	
ChApp_Inventor	pop0318204439	902	1	0	0	
Messenger	pop0318204636	3	0	0	0	
DIPOLOG_SAPASAP_MagicTrick	pop0318204755	7	0	0	0	
GAME	pop0318210018	5	0	0	0	
TextToSpeech2	pop0318210240	10	0	0	0	
Star_on_the_Earth	pop0318210328	791	8	0	0	
HassansCalculator	pop0318210537	0	0	0	0	
Liekbeforedownload	pop0318210607	0	0	0	0	
SoundWhoGoesfirst	pop0318210652	16	0	0	0	

Figura 4.6: CSV con los bloques (blocks.csv).

4.3. Interfaz de Dr. AppInventor

4.3.1. Página de inicio

La web que verá el usuario al entrar en Dr. AppInventor tiene el aspecto mostrado en la figura 4.7. La apariencia es exactamente la de Dr. Scratch, utilizamos todas las secciones e imágenes cambiando Scratch por AppInventor.

Para subir el proyecto que se desea analizar, a diferencia de Dr. Scratch, sólo se pueden subir archivos locales, ya que AppInventor no proporciona una URL desde la que descargar los proyectos. En el apartado de subida, pulsando la palabra “proyecto” de color rojo, aparece una imagen que muestra cómo el usuario puede descargar los proyectos de AppInventor a su ordenador. Esta imagen puede verse en la figura 4.8.

4.3.2. Página de resultados

En la figura 4.9 se muestra un ejemplo del interfaz web que el usuario ve una vez su proyecto ha sido analizado. Como mencionamos anteriormente, solo se implementa la evaluación de los criterios del Pensamiento Computacional.



Figura 4.7: Página de inicio de Dr. AppInventor.

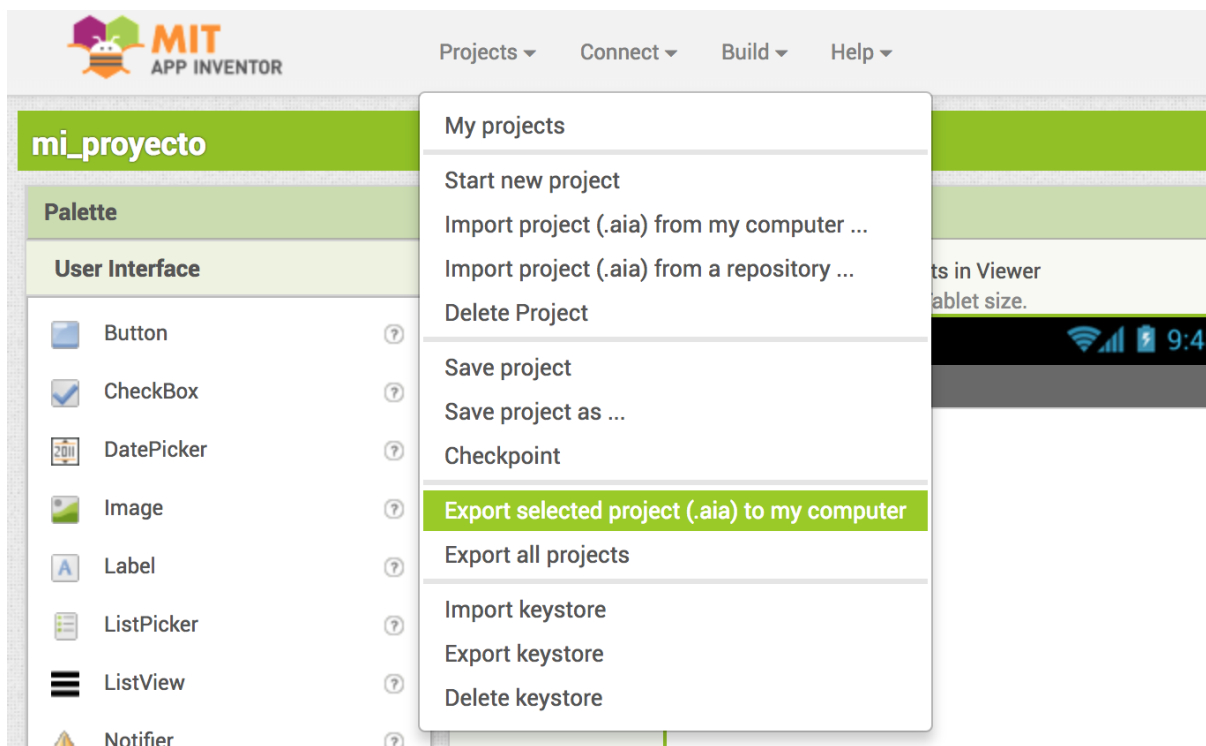


Figura 4.8: Ayuda para descargar un proyecto AppInventor.

A la izquierda se puede ver en primer lugar la puntuación total y debajo las funcionalidades de Dr. Scratch que no se han implementado para este proyecto: “Mejores prácticas” que mues-

tran los malos hábitos de programación en el proyecto y “Certificado del proyecto” donde se puede descargar un diploma con el resultado obtenido.

A la derecha se muestran las puntuaciones de los criterios individualmente. Este ejemplo en concreto es el programa con mayor variedad de bloques, esto es muy importante para la puntuación final como veremos en el siguiente capítulo. Podemos ver que obtiene una calificación alta: 36/45. En más de la mitad de criterios consigue la puntuación máxima, 3 puntos. Cuatro de los criterios se califican con 2 puntos y únicamente hay un criterio con 1 punto y otro que no puntúa.



Figura 4.9: Página de resultados en Dr. AppInventor.

Capítulo 5

Resultados

Con los ficheros generados en la última sección, ya tenemos los datos que vamos a tratar y analizar, para comprobar el estado actual de los proyectos desarrollados por la comunidad AppInventor.

5.1. Número de bloques

En primer lugar, guardamos en una variable tipo DataFrame los datos de blocks.csv y utilizamos la función *describe()* para obtener estadísticas descriptivas del número de bloques total en los proyectos como la media, el mínimo, el máximo o la mediana. Seguidamente se eliminan los proyectos vacíos, es decir, proyectos cuyo número de bloques es igual a cero y se vuelve a llamar a la función *describe()*. Por último, se realiza el mismo proceso con los proyectos duplicados. Para eliminar los proyectos duplicados, pandas proporciona *drop_duplicates()*.

En la Tabla 5.1 podemos visualizar las estadísticas de los tres estados comentados. Originalmente había 1.222 proyectos, tras la eliminación de proyectos vacíos hay 1.102 y al descontar los duplicados obtenemos la cifra final de 730 proyectos. Podemos ver que la media de bloques aumenta cuando descartamos proyectos no válidos. Lo mismo ocurre con las demás estadísticas, excepto el máximo que siempre es 5.476.

	Original	Sin vacíos	Sin duplicados
Proyectos	1.222	1.102	730
Media	187,89	208,35	211,48
Desviación típica	463,68	483,90	502,62
Mediana	47	55	60
Mínimo	0	1	1
Máximo	5.476	5.476	5.476

Tabla 5.1: Estadísticas del número de bloques.

5.2. Variedad en los proyectos

Llamamos variedad al número de bloques de diferente tipo empleados en un proyecto. En el análisis, la variedad se cuenta como el número de columnas (que representan los tipos de bloques) con valor distinto de cero, para todas las filas (que representan los proyectos). La figura 5.1 representa la distribución de la variedad en todos los proyectos, donde el eje X es la variedad y el eje Y, la frecuencia, o las veces que se da una variedad. Observamos que los valores predominantes de la variedad están entre 3 y 6, mientras que según aumenta de valor, disminuye la frecuencia de proyectos. La máxima variedad es 61 y solamente un proyecto la tiene.

5.3. Uso de bloques

En esta sección se estudia el uso de los distintos bloques en los proyectos, de este modo podremos conocer si existe alguno infrutilizado.

5.3.1. Familias de bloques

Inicialmente se analizan las familias, entendiendo por familias los 8 tipos de bloques mencionados en la sección 3.1 sumando una nueva familia formada por los bloques específicos de los componentes, que llamaremos Componentes. En la Tabla 5.2 se pueden ver en orden de uso. En primer lugar con diferencia, encontramos Componentes que, al abarcar diferentes elemen-

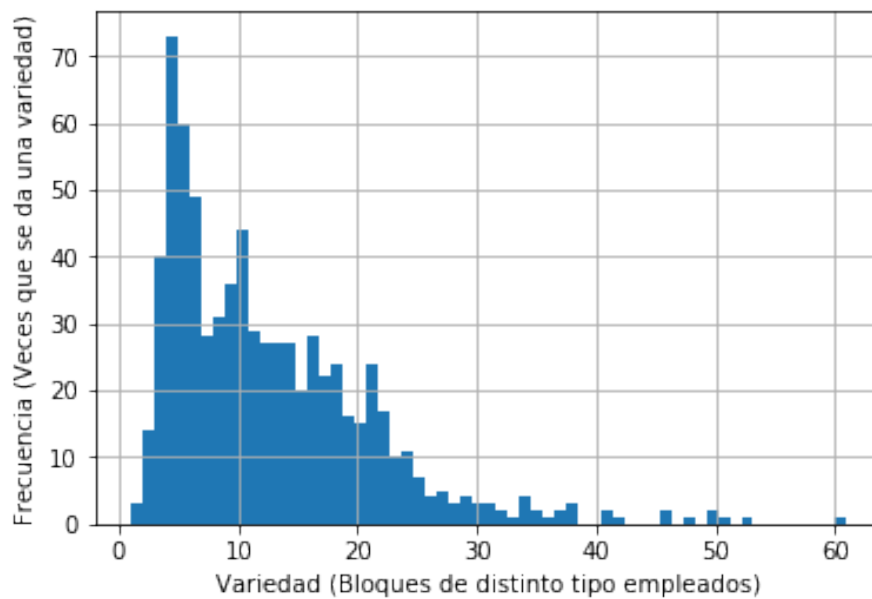


Figura 5.1: Distribución de la variedad.

tos (cada uno con sus bloques de métodos y eventos) es normal que predomine. Mientras que en último lugar encontramos la familia Colores, no es de extrañar ya que en comparación con Componentes su aplicación es mucho más reducida. Por otro lado, desde el punto de vista del Pensamiento Computacional, hay dos familias muy ligadas que no destacan: Control, relacionada con el pensamiento algorítmico y Procedimientos, relacionada con la abstracción. Estas familias ocupan el sexto y octavo lugar respectivamente, cuando deberían ser de las principales. Este es uno de los posibles aspectos a resaltar durante el aprendizaje de los usuarios.

5.3.2. Tipos de bloques

Nos referimos con tipos de bloques a cada uno de los diferentes bloques que se pueden utilizar en AppInventor. Para esta sección, se realiza un sumatorio de los bloques utilizados en todos los proyectos, separados por tipo. En la Tabla 5.3 se muestran las estadísticas descriptivas del número de bloques que conforman los tipos. Vemos que hay 107 tipos diferentes en total con una media de aproximadamente 1.443 bloques. La desviación típica es mucho mayor que la media, lo que significa que existen valores extremos, es decir, hay grandes diferencias entre valores. Otra prueba de ello es la mediana, 133, que nos indica que, por lo menos, el 50% de los proyectos tienen 133 bloques o menos, quedando muy lejos de la media. Esto nos indica que

Familia	Frecuencia
Componentes	49.507
Texto	25.426
Matemáticas	24.862
Variables	21.765
Lógica	12.792
Control	8.915
Listas	4.381
Procedimientos	4.282
Colores	2.452
Total	154.382

Tabla 5.2: Familias ordenadas por uso.

habrá una minoría de proyectos con valores muy elevados que alzarán la media, como puede se puede ver en el máximo: 28.927 bloques.

Tipos	107
Media	1.442,82
Desviación típica	4.232,51
Mediana	133
Mínimo	1
Máximo	28.927
Total	154.382

Tabla 5.3: Estadísticas de los tipos de bloques.

A continuación se analizan los bloques más y menos utilizados. Además de realizar este estudio para todos los proyectos, se estudia también el uso en proyectos con baja variedad. Decimos que un proyecto tiene baja variedad cuando contiene como máximo 10 bloques diferentes. Se elige esta cifra ya que, de este modo los proyectos con baja variedad y los de alta variedad se distribuirán aproximadamente al 50 %, como se puede ver en la Tabla 5.3.

Los resultados en la Tabla 5.4 muestran, fijándonos primero en todos los proyectos, que

los dos bloques más utilizados son *component_set_get*, cuya función es introducir u obtener las propiedades de un componente, y *text* que se utiliza para crear strings. A continuación, está *math_number*, utilizado para representar números. En cuarto lugar, *lexical_variable_get*, proporciona el valor de cualquier variable. Por último, *component_event* toma el valor de cualquier evento que pertenezca a un componente. Estos bloques representan un alto porcentaje de sus correspondientes familias: Componentes: 77,82 %; Texto: 89,06 %; Matemáticas: 61,57 %; Variables: 67,37 %. Mientras tanto, los bloques menos utilizados son aquellos con una función menos práctica desde el punto de vista de una aplicación móvil. Por ejemplo, *math_tan* que calcula la tangente dado un valor o *text_is_string* que comprueba si una variable es un string.

Por otro lado, vemos que tres de los cinco primeros bloques en todos los proyectos aparecen también como los más utilizados en los proyectos con baja variedad. Los dos primeros coinciden, mientras que *component_event* ahora aparece el tercero. Los nuevos bloques son *component_method* con el que se llama a los métodos que pueda utilizar un componente, y *controls_openAnotherScreen* que abre una nueva pantalla con el nombre que se pase como argumento. Dentro de los bloques menos utilizados, descubrimos 42 que no se utilizan ni una sola vez. Se han seleccionado los cinco que aparecen en la tabla porque confirman la necesidad de insistir en las familias Control y Procedimientos a la hora de instruir. Estos bloques ya se habían mencionado en la sección 4.1.2 como factores indispensables para puntuar en, al menos, tres criterios.

5.4. Relación Puntuación/Variedad

En esta sección pretendemos comprobar si es cierto lo que sugiere la intuición: a mayor variedad, mayor puntuación. La Tabla 5.5 muestra una comparación estadística de las puntuaciones de los proyectos con alta variedad frente a los proyectos con baja variedad. Los proyectos con más de 10 bloques diferentes representan el 48,2 % de los proyectos (352 de 730). A diferencia de las estadísticas anteriores, la media y la mediana apenas se diferencian, lo que significa que no habrá grandes saltos entre valores. Observando los datos obtenidos, las medias son 18,56 y 9,89, y los máximos, 36 y 19, respectivamente. Por lo tanto, se confirma la hipótesis inicial, ya que la variedad alta prácticamente duplica todos los valores de la baja.

Todos los proyectos		Proyectos baja variedad	
component_set_get	28.927	component_set_get	5.562
text	22.644	text	4.538
math_number	15.308	component_event	3.960
lexical_variable_get	14.662	component_method	3.484
component_event	9.599	controls_openAnotherScreen	1.277
...
list_to_csv_table	3	controls_forEach	0
math_tan	2	controls_while	0
obfuscated_text	2	controls_choose	0
text_is_string	2	procedures_callreturn	0
controls_closeScreenWithPlainText	1	procedures_defreturn	0

Tabla 5.4: Bloques más y menos utilizados.

	Alta variedad	Baja variedad
Proyectos	352	378
Porcentaje	48,2	51,8
Media	18,56	9,89
Desviación típica	5,33	3,27
Mediana	18	10
Mínimo	7	3
Máximo	36	19

Tabla 5.5: Puntuaciones con alta y baja variedad.

5.5. Variedad frente a cantidad

Conociendo los resultados de la sección anterior, nos hacemos una idea de que es más importante un proyecto con alta variedad antes que un proyecto con muchos bloques de los mismos tipos. Para comprobar esto, se compara el proyecto que posee mayor número de bloques con el proyecto de mayor variedad. En la Tabla 5.6 podemos ver que se cumple. El primer proyecto

contiene 5.476 bloques pero solo 16 son diferentes, en consecuencia obtiene una puntuación de 18. Sin embargo, el proyecto con 4.115 bloques contiene 61 tipos distintos y su puntuación es mucho mayor, 36.

Cantidad	Variedad	Puntuación
5.476	16	18
4.115	61	36

Tabla 5.6: Comparación variedad frente cantidad.

Con estos resultados podemos concluir que la variedad es otro parámetro a tener en cuenta a la hora de buscar mejoras en los usuarios. Un mayor énfasis en la variedad puede llegar a multiplicar la puntuación por dos. Esto se debe principalmente a que los criterios de evaluación se basan, en su mayoría, en la aparición de diferentes bloques.

Capítulo 6

Conclusiones

Este último capítulo es un análisis de los resultados conseguidos con este Trabajo Fin de Grado desde el punto de vista de los objetivos planteados pero también desde el punto de vista de los conocimientos aplicados y adquiridos. Por último, finaliza con el planteamiento de algunos futuros trabajos.

6.1. Consecución de objetivos

Repasando el capítulo 2, el objetivo principal de este proyecto es crear una herramienta que analice el Pensamiento Computacional de proyectos AppInventor. Este objetivo sí se ha cumplido, hemos creado la aplicación web Dr. AppInventor, que sigue el diseño de Dr. Scratch con la rúbrica de Code Master y proporciona una evaluación de los proyectos AppInventor.

Por otro lado, a partir de esta herramienta pretendíamos buscar caminos de aprendizaje para guiar a los usuarios en el desarrollo de sus habilidades del Pensamiento Computacional. Esta búsqueda la realizamos mediante el análisis de varios proyectos de la galería de AppInventor. Este análisis nos ha proporcionado información sobre cómo los alumnos pueden mejorar, ya que hemos encontrado carencias en bloques directamente ligados a las habilidades. Además, hemos confirmado la importancia de la variedad de bloques dentro de un proyecto para conseguir un buen resultado. Podemos decir que esta información nos muestra al menos tres lecciones para los usuarios: bloques de Control, bloques de Procedimientos y variedad en los proyectos. Por lo tanto, el objetivo de localizar caminos de aprendizaje también se ha cumplido.

Los objetivos específicos, como ya se mencionó en su correspondiente capítulo, eran las tareas que hizo falta completar para conseguir el objetivo general. Por ello, afirmamos que se cumplieron. Sin embargo, la obtención de proyectos se vio limitada debido a problemas con la web de AppInventor. Como ya adelantamos en la sección 4.2.1, utilizamos dos scripts para automatizar la extracción de los proyectos, uno de copia al perfil y otro de descarga. El script de copia solo podía guardar en el perfil entre 30 y 40 proyectos en una pasada, ya que, a partir de estas cifras, la web se bloqueaba y dejaba de responder. Algo similar ocurría con el script de descarga. Esto provocó un aumento del tiempo de descarga y, por lo tanto, una disminución del número de proyectos obtenidos.

6.2. Aplicación de lo aprendido

Para realizar este trabajo, me han resultado indispensables todas las asignaturas que he tenido relacionadas con la programación. A continuación, se enumeran en orden cronológico.

1. Informática I, con el lenguaje “Picky”, donde aprendí los fundamentos básicos de la programación y que los problemas se resuelven “paso a paso”.
2. Informática II, con el lenguaje ADA. Mi primer contacto con la programación contrarreloj, me enseñó que no siempre voy a estar en un entorno cómodo y aún así voy a tener que desenvolverme.
3. Protocolos para la Transmisión de Audio y Vídeo en Internet: aquí conocí Python, lenguaje que he necesitado para crear la aplicación web con Django y para el análisis de datos con pandas.
4. Construcción de Servicios y Aplicaciones Audiovisuales en Internet, Gráficos y Visualización 3D y Laboratorio de Tecnologías Audiovisuales en la Web: estas asignaturas se complementaban entre sí, me enseñaron todo lo que sé de tecnologías web: JavaScript, HTML, CSS...

Sería injusto por mi parte no mencionar las demás asignaturas ya que, de un modo u otro, todas ellas me han aportado algo, me han ayudado a plantear y resolver los problemas que se presenten. En definitiva, me han ayudado a ser ingeniero.

6.3. Lecciones aprendidas

En esta sección se explican los conocimientos que se han adquirido gracias al desarrollo de este trabajo:

1. La herramienta Django de Python: para la implementación de la aplicación web. Además, me sirvió para adelantar trabajo ya que, en la asignatura Laboratorio de Tecnologías Audiovisuales en la Web del segundo cuatrimestre de este curso, fue parte del temario.
2. La librería pandas también de Python, me permitió tomar contacto con el análisis de datos y de la cual todavía me queda mucho por descubrir.
3. Dejando de lado la programación, he descubierto lo que es abarcar un proyecto real, donde no hay un guión a seguir como en las prácticas de la universidad.

6.4. Trabajos futuros

En esta sección se proponen sugerencias o ideas para futuros trabajos con intención de mejorar las habilidades del Pensamiento Computacional:

- En primer lugar, incluir los caminos de mejora en la herramienta Dr. AppInventor.
- Proseguir con el análisis, añadiendo nuevos proyectos para obtener más información.
- Siguiendo las posibles mejoras propuestas en [8], un posible trabajo futuro es mostrar a los aprendices ejemplos y proponer ejercicios para que practiquen. Los ejemplos y ejercicios propuestos deben incluir bloques de Control y/o Procedimientos, una variedad alta o la combinación de ambos. Otro posible paso sería proporcionar retroalimentación personalizada a cada usuario basada en sus progresos y necesidades. Para llevar esto a cabo, podría extrapolarse el análisis realizado en este trabajo a los proyectos de cada usuario en concreto.
- Otra medida que se propone en [8] es integrar las herramientas de análisis en sus entornos de desarrollo correspondientes, a raíz de esto puede surgir una nueva idea aunque más extrema: diseñar un nuevo entorno orientado especialmente al desarrollo del Pensamiento Computacional.

Apéndice A

Manual de usuario

Este apéndice está dedicado a explicar cómo poner en marcha y utilizar la aplicación web Dr. AppInventor. Es necesario tener Python 2.7 y Django 1.8 para ejecutarlo.

Los archivos que necesita Dr. AppInventor se encuentran en un repositorio de Github¹ y para obtenerlos se puede hacer de dos formas: clonar el repositorio o descargarlo como ZIP. Una vez se ha descargado todo, hay que seguir estos pasos:

1. Abrir el terminal en el directorio donde se encuentre el código.
2. Ejecutar el comando: *python manage.py migrate*.
3. Ejecutar el comando: *python manage.py runserver*.
4. Abrir en un navegador la siguiente URL: <http://localhost:8000>.

En el navegador aparecerá la página de inicio, vista en la imagen 4.7. Para subir el proyecto que se desea analizar hay que pulsar en “Elige Proyecto” y seleccionar el archivo. El análisis comenzará tras hacer click en el botón “ANALIZA MI PROYECTO” y la aplicación pasará a la página con los resultados del proyecto, similar a la vista en la imagen 4.9

En un futuro, si la aplicación se ejecuta desde un servidor y se asigna a un dominio, los usuarios no tendrán que realizar estos pasos. Únicamente necesitarán acceder desde su navegador a la URL correspondiente.

¹<https://github.com/robernom/dr-app-inventor>

Bibliografía

- [1] J. Bastida. Python + django. <https://speakerdeck.com/jorgebastida/python-django>, may 2012.
- [2] D. Flanagan. *JavaScript: the definitive guide*. O'Reilly Media, 2006.
- [3] D. Kuhlman. *A Python Book: Beginning Python, Advanced Python, and Python Exercises*. PLATYPUS GLOBAL MEDIA, 2011.
- [4] M. Lutz. *Programming Python: Powerful Object-Oriented Programming*. O'Reilly Media, 2010.
- [5] D. S. McFarland. *JavaScript & jQuery: the missing manual*. .°Reilly Media, Inc.”, 2011.
- [6] J. Moreno-León and G. Robles. Dr. scratch: A web tool to automatically evaluate scratch projects. In *Proceedings of the workshop in primary and secondary computing education*, pages 132–133. ACM, 2015.
- [7] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.
- [8] G. Robles, J. C. R. Hauck, J. Moreno-León, M. Román-González, R. Nombela, and C. G. von Wangenheim. On tools that support the development of computational thinking skills: Some thoughts and future vision. In *Proceedings of International Conference on Computational Thinking Education (CTE 2018)*, S. C Kong, J Sheldon, and K. Y Li (Eds.). *The Education University of Hong Kong*, page to appear, 2018.

- [9] C. G. von WANGENHEIM, J. C. Hauck, M. F. Demetrio, R. Pelle, N. d. CRUZ ALVES, H. Barbosa, and L. F. Azevedo. Codemaster-automatic assessment and grading of app inventor and snap! programs. *Informatics in Education*, 17(1), 2018.
- [10] W3C. Documentación de css. <https://www.w3.org/standards/webdesign/htmlcss#whatcss>.
- [11] W3C. Documentación de html. <http://w3c.github.io/html/>.
- [12] Documentación de jupyter notebook. <https://jupyter-notebook.readthedocs.io/en/stable/notebook.html>.
- [13] Vista general del paquete pandas. <https://pandas.pydata.org/pandas-docs/stable/overview.html>.
- [14] J. M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.
- [15] D. Wolber. App inventor and real-world motivation. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 601–606. ACM, 2011.