Universidad
Rey Juan Carlos

ESCUELA DE INGENIERÍA DE FUENLABRADA

INGENIERÍA TELEMÁTICA

TRABAJO FIN DE GRADO

DR. SNAP! UNA APLICACIÓN WEB DE EVALUACIÓN DEL LENGUAJE DE PROGRAMACIÓN SNAP!

Autor : Miguel del Prado Ventura

Tutor : Dr. Gregorio Robles Martínez

Curso académico 2024/2025

Trabajo Fin de Grado

DR. SNAP! UNA APLICACIÓN WEB DE EVALUACIÓN DEL LENGUAJE DE PROGRAMACIÓN SNAP!

Autor : Miguel del Prado Ventura

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2025, siendo calificada por el siguiente tribunal:

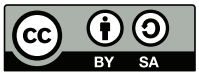
Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente **Calificación:**

Fuenlabrada, a de de 2025



Dedicado a

mi familia

Agradecimientos

Quiero dar las gracias a mi familia por siempre apoyarme. En especial, a mi madre, quien siempre me ha apoyado en todos los aspectos de mi vida y quien es un referente para mí tanto en lo personal como en lo profesional. También a mi padre, que siempre ha confiado en mí para llegar hasta este momento. Y, por último, a mi hermana, que siempre ha estado dispuesta a ayudarme cada vez que lo he necesitado y de la cual estoy cien por cien convencido de que aprobará la oposición. Os quiero mucho.

También quiero agradecer tanto a los amigos que conozco de toda la vida como a los que he conocido gracias a la Universidad. Con ellos he compartido una gran cantidad de horas en clase, en la biblioteca estudiando juntos y con los que he pasado y paso horas en llamada a través del Servidor de Discord, Toro Hits.

Para finalizar, quiero dar las gracias a mi tutor, Gregorio, y, por supuesto, al equipo de Dr. Scratch! por haberme ofrecido siempre su ayuda.

Resumen

En este Trabajo Fin de Grado se desarrolla una aplicación web llamada Dr. Snap!, cuyo objetivo es analizar proyectos con el fin de evaluar el desarrollo de habilidades asociadas al pensamiento computacional. Los proyectos analizados se encuentran en la plataforma Berkeley Snap! y han sido creados utilizando el lenguaje de programación Snap!.

La aplicación dispone de un procedimiento que permite importar proyectos, ya sea mediante la introducción de su URL o la carga directa del archivo correspondiente, para posteriormente analizar su contenido en función de distintas dimensiones, tales como la lógica, la sincronización o la abstracción. Asimismo, la herramienta ofrece distintos modos de análisis que proporcionan perspectivas complementarias sobre el nivel de desarrollo del pensamiento computacional presente en los proyectos.

Además, se han incorporado mecanismos para detectar estilos de programación incorrectos ("Bad Smells"), con el objetivo de ofrecer al estudiante información útil para la mejora de su proyecto.

Para la elaboración de este Trabajo Fin de Grado se han utilizado tecnologías como HTML, CSS, Python o XML. Dr. Snap! está basada en Dr. Scratch!, una aplicación que examina proyectos creados mediante el lenguaje de programación Scratch, el cual presenta similitudes con Snap!.

En conclusión, Dr. Snap! es una herramienta del ámbito educativo, pudiendo servir de apoyo tanto a:

i) los estudiantes que deseen aprender a programar y/o identificar sus puntos fuertes y débiles, como a

ii) los profesores que quieran evaluar los proyectos de sus estudiantes.

Summary

This Final Degree Project develops a web application called Dr. Snap!, whose objective is to analyse projects in order to assess the development of skills associated with computational thinking. The projects analysed are located on the Berkeley platform Snap! and have been created using the Snap! programming language.

The application has a procedure that allows projects to be imported, either by entering their URL or by directly uploading the corresponding file, and then analysing their content according to different dimensions, such as logic, synchronisation or abstraction. . The tool also offers different modes of analysis that provide complementary insights into the level of development of computational thinking present in the projects.

In addition, mechanisms have been incorporated to detect incorrect programming styles ("Bad Smells"), with the aim of providing the student with useful information for the improvement of his project.

Technologies such as HTML, CSS, Python or XML have been used for the elaboration of this Final Degree Project. Dr. Snap! is based on Dr. Scratch!, an application that examines projects created using the Scratch programming language, which has similarities with Snap!

In conclusion, Dr. Snap! is an educational tool that can be used as a support tool by:

i) students who want to learn to program and/or identify their strengths and weaknesses , and

ii) teachers who want to evaluate their students' projects.

Índice general

1. Introducción **1**

1.1. Bases de la Programación y Desarrollo 2

1.1.1. Berkeley Snap! 2

1.1.2. Pensamiento computacional 3

1.2. Estructura de la memoria 6

2. Objetivos **7**

2.1. Objetivo general 7

2.2. Objetivos específicos 7

2.3. Planificación temporal 8

3. Estado del arte **9**

3.1. Berkeley Snap! 9

3.2. HTML 14

3.3. CSS 14

3.4. JavaScript 15

3.5. Python 16

3.6. Django 16

3.7. XML 17

3.8. Bad Smells 18

3.9. LaTeX 18

4. Diseño e implementación **21**

4.1. Arquitectura general 21

4.2. Parseo del XML 23

4.3. Funciones de evaluación del PC 23

4.4. Modo Personalizado 28
4.5. Análisis de Bad Smells 29
4.6. Modo Asistente 30
4.7. Certificado del proyecto 31

5. Ejemplos de uso **33**

6. Conclusiones **43**
6.1. Consecución de objetivos 43
6.2. Aplicación de lo aprendido 44
6.3. Lecciones aprendidas 44
6.4. Trabajos futuros 45

Bibliografía **47**

Índice de figuras

1.1. Portada de Berkeley Snap! 2
1.2. Entorno de programación de Snap! 3

2.1. Diagrama de Gantt 8

3.1. Bloques de tipo motion, looks y sound 10
3.2. Bloques de tipo pen, control y sensing 10
3.3. Bloques de tipo variables y operators 11
3.4. Ejemplo de script 12
3.5. Ejemplo de sprite y stage 12
3.6. Ejemplo de personal costumes 13
3.7. Ejemplo de personal sound 13
3.8. Estructura básica de un documento HTML 14
3.9. Ejemplo de uso de LaTeX 19

4.1. Estructura de la aplicación 22
4.2. Estructura XML para parsear 24
4.3. Estructura General de los Bloques 24
4.4. Estructura del código de búsqueda de bucles 26
4.5. Estructura del código de búsqueda de ifs 27
4.6. Primer ejemplo de rúbrica 28
4.7. Segundo ejemplo de rúbrica 28
4.8. Ejemplo de URL personalizado 29
4.9. Vista del análisis realizado 29
4.10. Vista del stage 30
4.11. Vista del sprite 30

4.12. Vista del código muerto 31
4.13. Jerarquía de Bad Smells 31
4.14. Ejemplo de certificado 32

5.1. main.html 34
5.2. main.html 34
5.3. Pantalla de carga 35
5.4. dashboard-default.html 35
5.5. dashboard-default.html 36
5.6. dashboard-personal.html 37
5.7. dashboard-recommender.html 37
5.8. Vista general de los proyectos 38
5.9. Análisis del proyecto Basic 39
5.10. Rúbrica del proyecto Winterrest en modo Extended 39
5.11. Rúbrica del proyecto Winterrest en modo Vanilla 41

Capítulo 1

Introducción

Este Trabajo Fin de Grado se centra en Berkeley Snap! [11], una plataforma de programación interactiva que permite a los usuarios desarrollar programas mediante bloques predefinidos o creando sus propios bloques personalizados.

Snap! ofrece la posibilidad de subir proyectos a la plataforma, acceder a tutoriales, consultar su propio manual de referencia y participar en un foro comunitario.

Se desarrollará una aplicación web en la que los usuarios podrán subir sus proyectos creados en Snap! para ser analizados y obtener una calificación basada en el desarrollo de pensamiento computacional.

El pensamiento computacional es una habilidad clave en la educación actual, ya que permite a los estudiantes abordar problemas de manera lógica y estructurada [19]. Se basa en cuatro ejes fundamentales: descomposición, que consiste en dividir un problema en partes más pequeñas; abstracción, que permite centrarse en los aspectos esenciales ignorando detalles irrelevantes; reconocimiento de patrones, identificar similitudes y regularidades en los problemas; y diseño de algoritmos, que implica la creación de soluciones paso a paso para resolver problemas [17].

Dicha aplicación web se llamará Dr. Snap!, la cual nace a partir de la aplicación web Dr. Scratch! [18] donde se analizan proyectos creados mediante el lenguaje de programación Scratch [8].

Con el objetivo de convertirse en una herramienta de carácter académico, la aplicación web estará diseñada tanto para estudiantes como para profesores. Su propósito es proporcionar a los estudiantes un entorno de aprendizaje óptimo, claro y conciso para desarrollar habilidades de programación, mientras que a los profesores les facilitará la tarea de corrección y evaluación de proyectos.

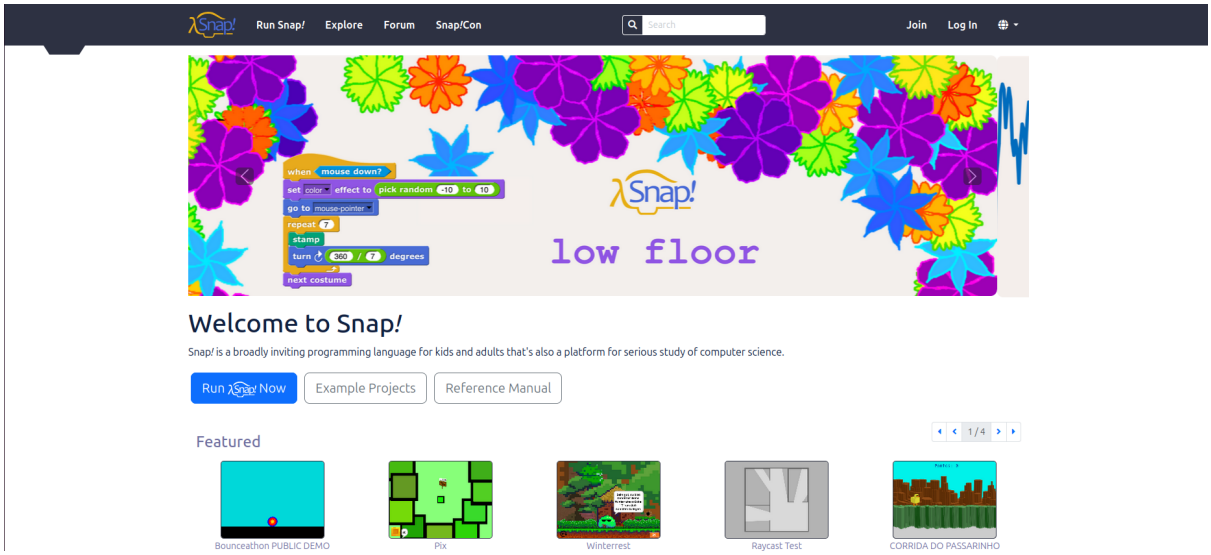


Figura 1.1: Portada de Berkeley Snap!

1.1. Bases de la Programación y Desarrollo

1.1.1. Berkeley Snap!

Snap! es una reconfiguración extendida de Scratch, la cual permite construir tus propios bloques [11]. También cuenta con listas, procedimientos, sprites, sonidos y disfraces. La página principal de Berkeley Snap! se observa en la Figura 1.1 y se puede encontrar en el siguiente enlace: `https://snap.berkeley.edu/`

Tal y como se muestra en la Figura 1.1, la aplicación web está formada por una barra superior donde se pueden encontrar las siguientes opciones: correr un programa, realizar una búsqueda de los proyectos que se han creado, acceder a un foro e incluso crear un usuario. Además, ofrece una breve definición de Snap! y acceso directo tanto al entorno de programación de Snap!, como a proyectos de ejemplo y al manual de referencia.

La función más relevante es el entorno de programación de Snap!, es decir, donde se crean los programas. Como se observa en la Figura 1.2 se encuentra dividido en tres zonas al observarlo de izquierda a derecha.

En la primera zona se sitúan los distintos bloques de programación que se pueden usar, los cuales están divididos en ocho categorías distintas: Motion, Looks, Sound, Pen, Control, Sensing, Operators, Variables.

La zona central es el propio entorno de programación, es decir, donde se va a establecer el código que se va a usar para la realización del programa.

Figura 1.2: Entorno de programación de Snap!

Por último, se encuentra un entorno visual donde se representará lo obtenido en el programa creado.

1.1.2. Pensamiento computacional

El pensamiento computacional se define como los procesos involucrados en la formulación de problemas para que sus soluciones puedan representarse como pasos computacionales y algoritmos [19, 6].

Una parte importante de este proceso es encontrar modelos apropiados de computación con los que formular el problema y derivar sus soluciones.

Sin embargo, a medida que los sistemas informáticos se vuelven más complejos y se aplican abstracciones de la informática a nuevos dominios de problemas, no siempre se tienen los modelos adecuados para idear soluciones. En estos casos, el pensamiento computacional se convierte en una actividad de investigación que incluye la invención de nuevos modelos apropiados de computación.

Para este proyecto se ha creado el modelo de computación representado en el Cuadro 1.1, basado en el de Dr. Scratch! [18].

- **Abstracción y Descomposición de problemas**: División del problema general en subproblemas más pequeños.

- **Paralelismo**: Capacidad de ejecutar varias secuencias de instrucciones de manera si-

multánea.

- **Pensamiento Lógico**: Uso de operadores condicionales y operadores lógicos para evaluar condiciones, estructuras de control que dirigen el flujo del programa y, por último, comprobar si en el caso de los operadores condicionales estos se encuentran anidados dentro de otras estructuras.

- **Sincronización**: Coordinación del tiempo en la ejecución de diferentes procesos cuando para realizar una instrucción es necesario que haya ocurrido otra anteriormente.

- **Control de flujo**: Uso de bucles para evaluar condiciones, estructuras de control que dirijan el flujo del programa y, por último, comprobar si estos se encuentran anidados dentro de otras estructuras.

- **Interactividad del Usuario**: Manera de medir el nivel de participación del usuario con el proyecto creado.

- **Representación de Datos**: Uso de los diferentes tipos de datos, por ejemplo, variables y listas, entre otros.

- **Operaciones Matemáticas y de Cadenas**: Uso de diferentes operaciones aritméticas, fórmulas, fórmulas trigonométricas, entre otras.

- **Operaciones de Movimiento**: Uso de bloques que conlleven algún desplazamiento o modificación del movimiento de un objeto.

Cada apartado se divide en cuatro niveles:

1. **Basic**

2. **Developing**

3. **Proficient**

4. **Advanced**

Concepto CT	Basic	Developing	Proficient	Advanced
Abstracción y Descomposición de problemas	Más de un script y más de un sprite	Uso de clones	Definición de bloques y variables	Uso avanzado de clones
Paralelismo	Ejecutar dos scripts al presionar la bandera verde	Dos scripts al presionar una tecla, dos scripts al hacer clic en un sprite en el mismo sprite	Dos scripts al recibir un mensaje, crear un clon, dos scripts cuando cambia el fondo	Sincronización avanzada de eventos
Pensamiento Lógico	If	If-else	Operaciones lógicas (AND, OR)	Condicionales anidados
Sincronización	Esperar	Enviar(Broadcast), al recibir un mensaje, detener todo, detener el programa, detener el sprite	Esperar hasta, cuando el fondo cambie, enviar y esperar	Manejo dinámico de mensajes
Control de Flujo	Secuencia de bloques	Repeat, Forever	Repeat until	Estructuras de flujo anidadas
Interactividad del Usuario	Bandera verde	Pulsar una tecla, clicar sobre un sprite, preguntar y esperar, bloques de ratón	Vídeo, audio	Entradas de usuario complejas
Representación de Datos	Modificadores de propiedades de sprites	Operaciones con variables	Operaciones con listas	Lógica booleana
Operaciones Matemáticas y de Cadenas	Aritmética	Fórmulas	Operaciones con cadenas	Trigonometría
Operaciones de Movimiento	Desplazamiento en los ejes X e Y	Rotación angular	Velocidad	Secuencias de movimiento complejas

Cuadro 1.1: Tabla sobre Conceptos de Pensamiento Computacional. Basado en [18].

1.2. Estructura de la memoria

En esta sección se explica la estructura de la memoria, cómo está organizada la memoria y una breve descripción sobre cada apartado.

- **Capítulo 1, Introducción.** Se realiza una breve explicación sobre el origen de Dr. Snap! y sobre el nivel de análisis que se creará a partir del pensamiento computacional.
- **Capítulo 2, Objetivos.** Se especifican los diferentes objetivos propuestos en el proyecto.
- **Capítulo 3, Estado del arte.** Se describen las distintas herramientas que se han utilizado para crear las funcionalidades de Dr. Snap!.
- **Capítulo 4, Diseño e implementación.** Se detalla de forma extendida y técnica la manera de crear cada aspecto del proyecto.
- **Capítulo 5, Ejemplos de uso.** Se realiza una explicación sobre determinados proyectos analizados.
- **Capítulo 6, Conclusiones.** Se estudia si los objetivos del proyecto se han cumplido. Además, se ponen en práctica los conocimientos adquiridos durante el Grado de Ingeniería Telemática, se analiza la mejora en las habilidades y se exponen posibles mejoras a realizar.

Capítulo 2

Objetivos

2.1. Objetivo general

Este Trabajo Fin de Grado tiene como objetivo crear una herramienta de análisis de desarrollo de pensamiento computacional de proyectos creados en Berkeley Snap!.

2.2. Objetivos específicos

A continuación, se presentan los distintos objetivos que se propusieron a la hora de crear el proyecto.

1. **Analizar proyectos de Snap!**, mediante dos opciones:
    - **Mediante URL**: Se obtendrán los datos de los proyectos directamente desde el servidor de Berkeley Snap!, realizando una solicitud a la plataforma para acceder a la información del código.
    - **Mediante la subida de un fichero XML**: Los usuarios podrán cargar manualmente el archivo XML del proyecto, permitiendo su análisis sin necesidad de conexión directa con el servidor de Snap!.
2. **Analizar "Bad Smells"**, donde se produce un continuo flujo de ideas para resolver dichos "Bad Smells".
3. **Crear un modo personalizado**, donde se creará una rúbrica según el nivel de análisis que se necesite.

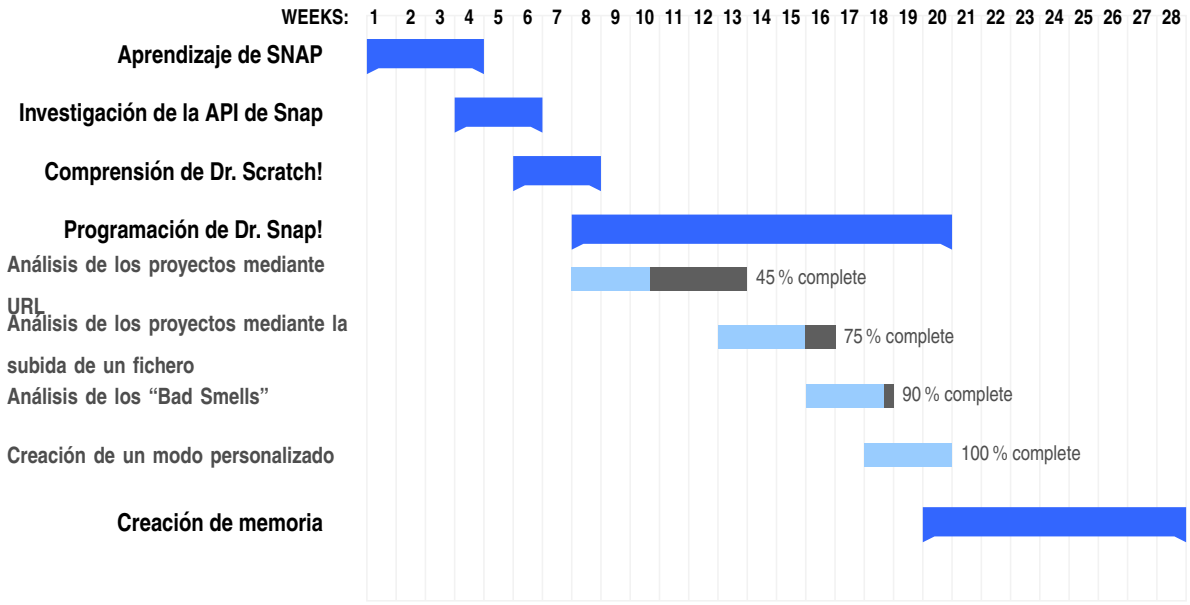


Figura 2.1: Diagrama de Gantt

2.3. Planificación temporal

En la Figura 2.1, se observa el Diagrama de Gantt, donde se indica el tiempo empleado en cada tarea del Trabajo Fin de Grado, entre los meses de septiembre y marzo. En este, se muestra el apartado Programación de Dr. Snap! dividido en cuatro fases y el porcentaje de cada fase respecto a lo global.

Capítulo 3

Estado del arte

En este capítulo se realizará una breve descripción de las tecnologías utilizadas para la creación del proyecto.

3.1. Berkeley Snap!

Anteriormente, se ha explicado genéricamente el funcionamiento de Berkeley Snap!. Este apartado se centra en los diferentes conceptos del entorno de programación.

- **Block**: Elemento básico del lenguaje de programación de Snap!, es decir, las diferentes órdenes que se pueden usar. A continuación, se desarrollan los bloques, los cuales están divididos según el valor que aporten al programa.

    1. **Motion**: Formado por las órdenes que supondrán un desplazamiento tanto en el eje X como Y o un cambio de ángulo.
    2. **Looks**: Permite modificar los componentes de la apariencia respecto al nombre, ancho, alto y píxeles e incluso el uso de disfraces.
    3. **Sound**: Permite reproducir sonidos variando el volumen y la frecuencia, entre otros.
    4. **Pen**: Permite modificar las características de los colores como la saturación, transparencia...
    5. **Control**: Formado por los bloques responsables de establecer un control del programa, incluyendo ifs y bucles.
    6. **Sensing**: Proporciona un control sobre las interacciones externas.

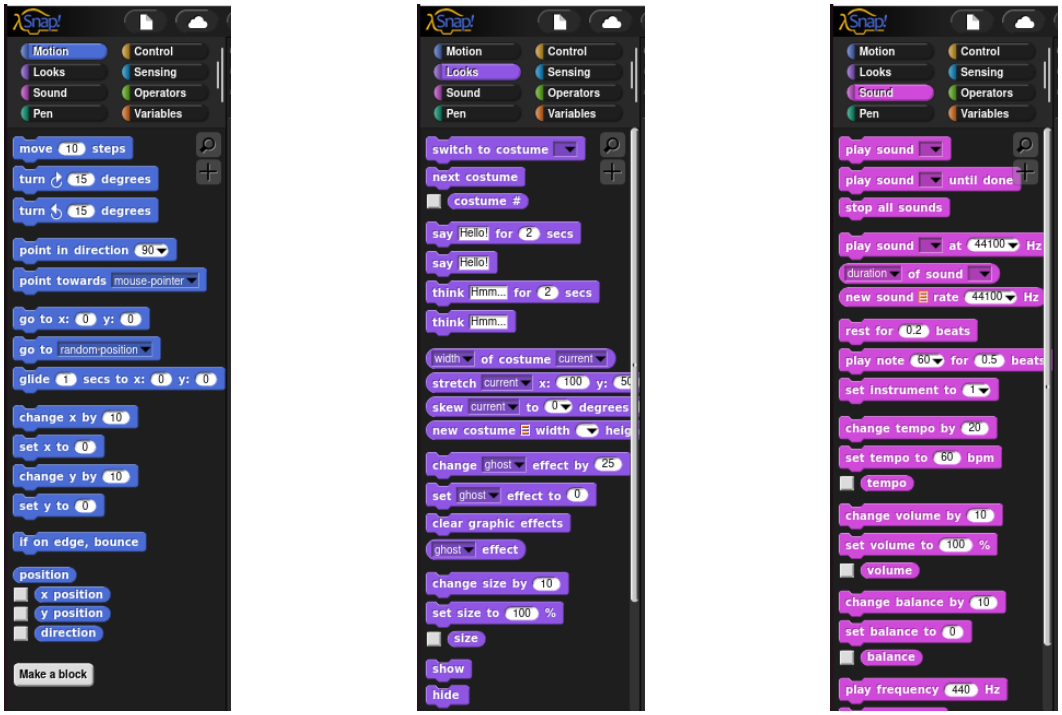


Figura 3.1: Bloques de tipo motion, looks y sound

Figura 3.2: Bloques de tipo pen, control y sensing

Figura 3.3: Bloques de tipo variables y operators

7. **Operators**: Formado por distintas operaciones matemáticas, aritméticas y geométricas.

8. **Variables**: Donde se pueden establecer variables de diferentes tipos de datos.

- **Script**: Agrupación de varios bloques, es decir, un conjunto de instrucciones, tal y como se observa en la Figura 3.4.

- **Sprite**: Diferentes elementos que aparecerán en la realización del proyecto. Generalmente, un proyecto está integrado por más de un Sprite y, a su vez, cada Sprite está formado por Scripts.

- **Stage**: Escenario en el que se reproducirá el proyecto.

- **Personalizar**: El entorno de programación de Snap! permite personalizar los distintos Sprites y Stages usados en el proyecto.

  1. **Costumes**: Permiten dibujar, añadir elementos geométricos e incluso subir imágenes.

  2. **Sound**: Permite grabar un sonido y añadirlo al proyecto.

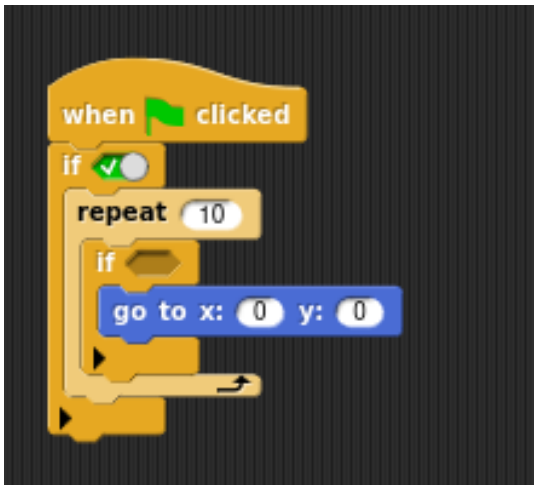


Figura 3.4: Ejemplo de script

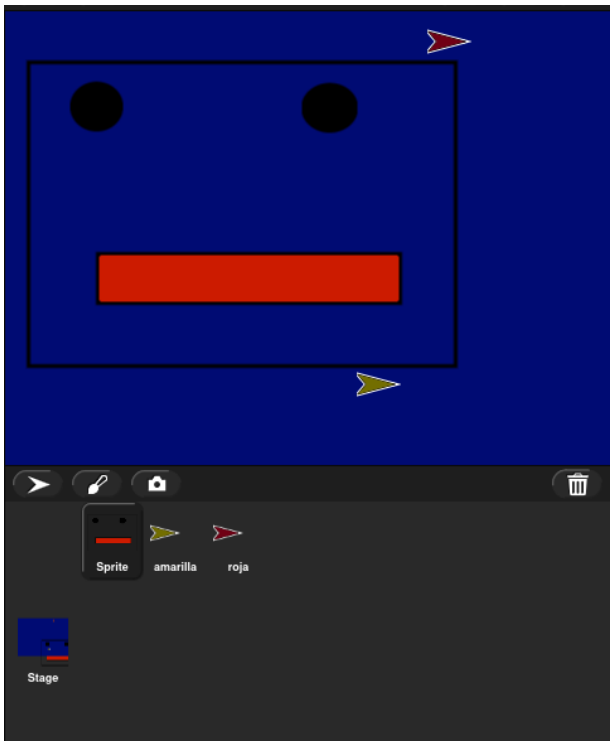


Figura 3.5: Ejemplo de sprite y stage

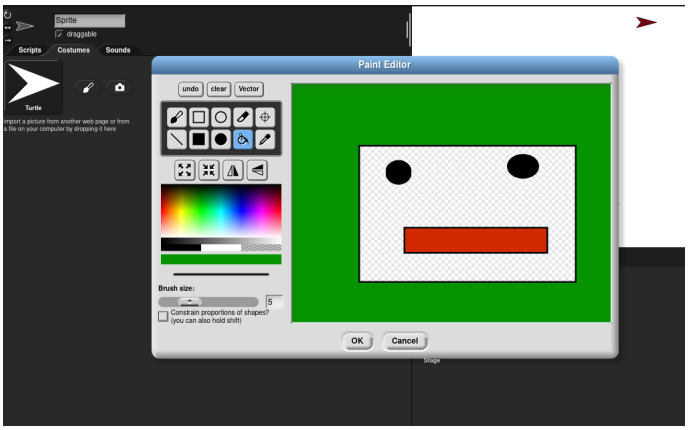


Figura 3.6: Ejemplo de personal costumes

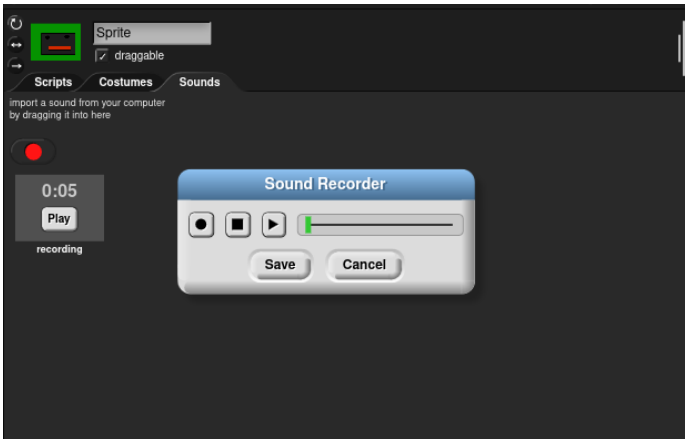


Figura 3.7: Ejemplo de personal sound

`<!DOCTYPE html>`	Indica la versión de HTML
`<html>`	Engloba todo el contenido de la página web
`<head>`	Define el encabezado
`<meta>`	Define meta-datos
`<title>`	Establece el título que aparecerá en la pestaña del navegador
`<body>`	Contiene todo el contenido visible de una página web
`<a>`	Insertar una URL
`<img>`	Insertar una imagen
`<!-- -->`	Añadir comentarios en el código

Figura 3.8: Estructura básica de un documento HTML

3.2. HTML

El lenguaje de marcado de hipertexto, Hypertext Markup Language o HTML, es un lenguaje de marcado estándar para la creación de páginas web y aplicaciones en línea. Considerado como un lenguaje de estructura y no como uno de programación, en él se define la organización del contenido en la web mediante etiquetas y atributos [12].

Un hipertexto se define como un texto utilizado para enlazar con otros. Por el contrario, un lenguaje de marcado es una serie de marcas que indican a los servidores web la estructura y el estilo de un documento (Figura 3.8).

Con HTML los usuarios web pueden crear y estructurar secciones, párrafos y enlaces mediante elementos, etiquetas y atributos. Además, se suele usar junto con CSS (para estilos) y JavaScript (para interactividad).

3.3. CSS

CSS (Cascading Style Sheets o Hojas de Estilo en Cascada) es un lenguaje de diseño que se emplea para definir la apariencia y presentación de documentos HTML y cuyo propósito principal es mejorar la estética y accesibilidad de las páginas web, separando el contenido de su diseño [7].

HTML y CSS son herramientas complementarias e indispensables para el diseño web.

CSS funciona mediante reglas de estilo que se aplicarán a los elementos HTML, sin em-

bargo, no todos los elementos se comportan igual. En este contexto, los elementos pueden clasificarse en dos grandes categorías:

- **Elementos no reemplazados (non-replaced)**: Aquellos cuya representación visual es generada directamente por el navegador a partir del contenido que contienen.
- **Elementos reemplazados (replaced)**: Aquellos cuyo contenido no es parte del HTML, sino que es sustituido por una representación externa, como una imagen o un vídeo.

La sintaxis de CSS se basa en reglas de estilo compuestas por:

- **Selector**: Indica a qué elemento o elementos HTML se aplican.
- **Bloque de declaración**: Formado por una o varias declaraciones, donde cada declaración especifica los estilos que se aplican.
- **Atributo**: Característica que se modifica.
- **Valores**: Definen el nuevo valor de la característica modificada.

3.4. JavaScript

JavaScript fue creado por Netscape en los primeros días del desarrollo de la web con el objetivo de dotar a los navegadores de capacidades interactivas del lado del cliente. Técnicamente, el nombre JavaScript es una marca registrada originalmente por Sun Microsystems (actualmente parte de Oracle) que describía la implementación del lenguaje desarrollada por Netscape (hoy parte de Mozilla). Para formalizar y estandarizar el lenguaje, Netscape lo sometió al proceso de normalización a través de ECMA International (Asociación Europea de Fabricantes de Ordenadores). Como resultado, la versión estandarizada del lenguaje se denomina oficialmente ECMAScript, aunque el nombre JavaScript se ha mantenido como el término más comúnmente utilizado en la industria [10].

Anteriormente, las páginas web eran de naturaleza estática. Una página estática presentaba información en un diseño fijo, sin ofrecer la interactividad ni las características avanzadas que se esperan de un sitio web moderno. En este contexto, JavaScript emergió como una tecnología del lado del cliente destinada a hacer que las aplicaciones web fueran más dinámicas. Gracias a este, los navegadores adquirieron la capacidad de responder a las interacciones de los usuarios,

permitiendo la modificación en tiempo real de la disposición del contenido dentro de la página web.

Asimismo, JavaScript es un lenguaje de programación ligero donde la gran mayoría de las herramientas necesarias para su uso son de software libre. Además, es un lenguaje dinámico debido a que los objetos se crean en el momento de su uso, es multiparadigma y puede ser interpretado o compilado justo-a-tiempo (just-in-time) con funciones de primera clase.

3.5. Python

A principios de la década de los 90, Python fue creado por Guido van Rossum en el Stichting Mathematisch Centrum (CWI) (Países Bajos) como sucesor de un lenguaje denominado ABC. A pesar de que Guido es el principal autor de Python, se incluyen contribuciones de otras muchas personas [3].

Python es un lenguaje de programación potente y fácil de aprender [15], y que dispone de estructuras de datos eficientes de alto nivel y un enfoque simple pero efectivo para la programación orientada a objetos. La elegante sintaxis y la escritura dinámica de Python, junto con su naturaleza interpretada, lo convierten en un lenguaje ideal para secuencias de comandos y desarrollo rápido de aplicaciones en la mayoría de las plataformas [2].

Comparado con otros lenguajes de programación populares, Python se caracteriza por tener una curva de aprendizaje progresiva. Así pues, permite a los desarrolladores principiantes alcanzar rápidamente un mejor nivel de productividad gracias a su sintaxis sencilla y legibilidad. Sin embargo, a medida que el usuario gana experiencia, el lenguaje ofrece una gran profundidad [14].

3.6. Django

Django es un framework web de alto nivel escrito en Python, diseñado para fomentar el desarrollo rápido de aplicaciones y establecer un diseño pragmático y limpio [5]. Es un proyecto libre y de código abierto que se encuentra disponible en el repositorio `https://www.djangoproject.com/download`.

Además, incluye una potente línea de comandos llamada “django-admin” que permitirá crear nuevos proyectos y gestionar tareas básicas.

Sus características principales son las siguientes:

- **Rápido desarrollo**: Sigue el principio "DRY" (Don't Repeat Yourself), evitando la repetición de código.
- **Seguro**: Protege contra ataques comunes como inyección SQL, CSRF y XSS.
- **Escalable**: Utilizado por grandes empresas como Instagram, Pinterest y Spotify.
- **Basado en MVC/MVT**: Organiza el código en Modelos, Vistas y Plantillas (MVT).
- **Base de datos integrada**: Incluye un ORM (Object-Relational Mapper) que facilita la gestión de bases de datos como PostgreSQL, MySQL y SQLite.
- **Panel de administración automático**: Genera un panel de administración sin necesidad de escribir mucho código.

3.7. XML

XML, acrónimo de Extensible Markup Language (Lenguaje de Marcado Extensible), es un lenguaje de marcado desarrollado en 1996 por el W3C (World Wide Web Consortium) y en el que existen dos versiones principales: XML 1.0 y XML 1.1.

XML proviene de SGML (Standard Generalized Markup Language), un estándar definido por la norma ISO 8879:1986. A su vez, SGML es un metalenguaje, es decir, un lenguaje diseñado para definir otros lenguajes de marcado.

Un lenguaje de marcado se determina como el conjunto de reglas que definen la estructura y la codificación de un documento. A pesar de que XML no constituye en sí mismo un lenguaje de programación debido a que no permite la inclusión de condicionales, variables, bucles u otros elementos propios de la computación, es ampliamente utilizado por los programadores para acceder a los datos y transmitirlos dentro de sus aplicaciones [9].

Además, XML es ideal para intercambiar datos entre distintas plataformas y aplicaciones sin depender del sistema que lo genera al poder configurarse cualquier estructura de datos y ser esta independiente de la fuente de extracción.

3.8. Bad Smells

"Bad Smells", también llamados "Code Smells", es el término que hace referencia a malos hábitos frecuentes entre los programadores. La existencia de "Bad Smells" no implica que los programas no funcionen correctamente, sino que el código no está escrito de forma adecuada [16].

En la mayoría de los casos, los errores no son graves, son patrones que tienen un impacto negativo en la calidad del software. Normalmente, suelen ser comunes en programadores principiantes, aunque también pueden aparecer en proyectos más avanzados.

Existen numerosos "Bad Smells" en la programación. A continuación, se explican los más comunes:

- **Variables sin inicializar:** en determinados lenguajes se pueden declarar variables sin asignarles un valor inicial, lo que puede provocar errores inesperados en tiempo de ejecución.
- **Tabulación incorrecta:** dificulta la lectura y comprensión del código.
- **Código duplicado:** este término indica la repetición del mismo código en distintos lugares. Existe la posibilidad de resolverlo creando funciones.
- **Exceso de comentarios**
- **Dead code:** fragmentos de código que nunca se ejecutan.
- **No seguir los estándares de programación,** lo que conlleva una gran dificultad a la hora de compartir el código o incluso a la hora de trabajar en equipo.
- **Uso de "números mágicos":** cada número debe ser sustituido por una constante que haga referencia a dicho valor.

3.9. LaTeX

LaTeX es un sistema de composición tipográfica de alta calidad que incluye características diseñadas para la producción de contenido técnico o científico. Asimismo, es el estándar de facto para la comunicación y publicación de documentos científicos gracias a su capacidad para

```
\title{DR. SNAP! UNA APLICACIÓN WEB DE ANÁLISIS EN EL LENGUAJE DE PROGRAMACIÓN SNAP!}
\author{Miguel del Prado Ventura}

% Guarda el título, el autor y la fecha en variables
\makeatletter
\let\thetitle\@title
\let\theauthor\@author

\let\thedate\@date
\makeatother

\renewcommand{\baselinestretch}{1.5}  %% Interlineado

\begin{document}

\renewcommand{\refname}{Bibliografía}  %% Renombrando
\renewcommand{\appendixname}{Apéndice}
```

Figura 3.9: Ejemplo de uso de LaTeX

manejar fórmulas matemáticas complejas, referencias bibliográficas y estructuración avanzada de documentos.

Escrito inicialmente por Leslie Lamport, se basa en el motor de composición tipográfica TeX, desarrollado por Donald Knuth [13]. A lo largo del tiempo, desarrolladores y autores han contribuido con un gran número de extensiones que amplían sus capacidades. Además, LaTeX está disponible como software libre.

A diferencia de los editores de texto tradicionales, LaTeX separa el contenido de la presentación. Esto permite mantener una consistencia tipográfica impecable y un control preciso sobre la estructura del documento.

Capítulo 4

Diseño e implementación

4.1. Arquitectura general

La arquitectura de la aplicación se representa en la Figura 4.1, donde se muestra la ruta a seguir para su funcionamiento.

Teóricamente se encuentra dividida en dos partes: el Frontend y el Backend.

El Frontend consta de diferentes estructuras HTML, diferentes estilos CSS y el lenguaje de programación JavaScript.

El Backend está formado por el uso de la API de Berkeley Snap!, Django, los diferentes scripts que se han creado para realizar las funcionalidades de la aplicación y, por último, el servidor local, por el que se comprueba el correcto funcionamiento de la aplicación.

La arquitectura de la aplicación sigue el modelo cliente-servidor, un modelo de comunicación que permite la distribución de tareas dentro de una red de ordenadores.

Asimismo, un servidor se define como un hardware o programa informático que proporciona recursos, datos o servicios a otros dispositivos llamados clientes. A su vez, se define cliente como un ordenador o programa que envía solicitudes al servidor y que recibe respuestas. Finalmente, el servidor procesará las peticiones del cliente y proporcionará la respuesta solicitada.

Este modelo representa la interacción entre cliente y servidor mediante peticiones y respuestas y es utilizado en muchos entornos tecnológicos, como páginas web, aplicaciones móviles, videojuegos en línea y sistemas empresariales [1].

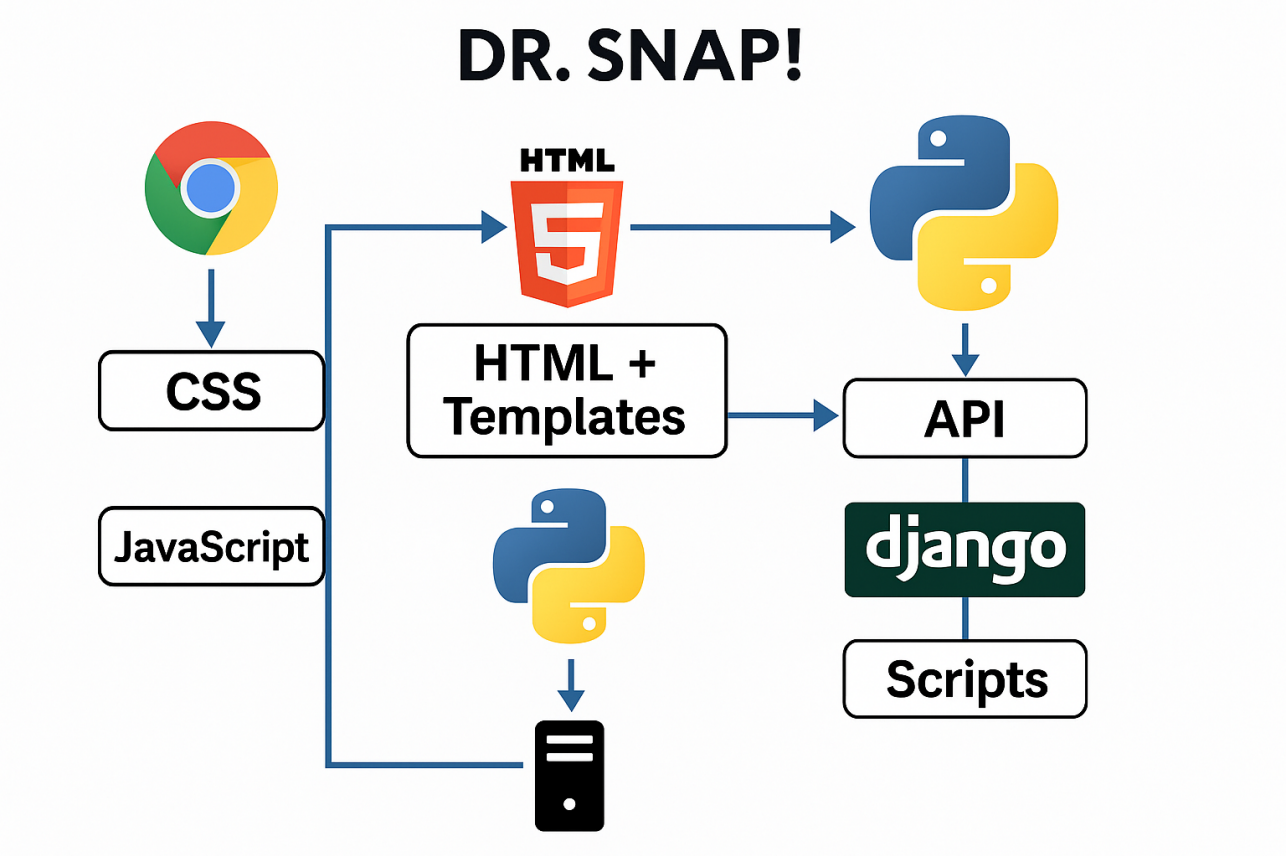


Figura 4.1: Estructura de la aplicación

4.2. Parseo del XML

En esta sección, se explicará el procedimiento llevado a cabo para dividir el archivo XML.

En primer lugar, se procede a la búsqueda de información sobre la API de Snap! localizable en el siguiente enlace: `https://snap.berkeley.edu/static/API`. Además de no ser de fácil acceso, se puede deducir a través de su apariencia que no es muy elaborada.

Identificado qué método GET es el más apto para este proyecto, se detecta que dicho GET devuelve el archivo XML del proyecto a analizar.

A continuación, se realizan multitud de solicitudes con el objetivo de comprobar cómo es exactamente la estructura XML que crea Snap! para cada proyecto y surge la necesidad de crear una función que recorra dicho XML. Para llevarlo a cabo, se hace uso de la librería xml.etree.ElementTree [4]. Gracias a ella se trabaja con la opción findall, adentrándose en el XML y accediendo a los datos necesarios (Figura 4.2). En este caso, dentro del bloque "block s".

A partir del bloque obtenido, se diseña una estructura con el objetivo de ser utilizada más adelante. Esta estructura está formada por el nombre del bloque, un identificador único y distintos valores opcionales.

En caso de que el bloque posea subbloques, se crea un campo "Next", donde se encontrará el nombre del siguiente bloque que se va a ejecutar.

Posteriormente, se elabora un campo "Option", que se introduce en los bloques con diferentes opciones de uso, y, para concluir, se añade el campo "Costume", formado por el nombre de los disfraces utilizados en cada apartado (Figura 4.3).

4.3. Funciones de evaluación del PC

En esta sección se explican las funciones creadas para poder llevar a cabo el análisis de los diferentes proyectos según el desarrollo de pensamiento computacional.

Con el objetivo de analizar los diferentes bloques de instrucciones, en primer lugar, se realiza una búsqueda del formato interno de cada orden relevante en Berkeley Snap!, debido, principalmente, a que en la mayoría de los casos no coincide con el nombre del entorno de programación de Snap.

Una vez traducidos los bloques de interés, se trabaja con una misma función. Esta función recibe un diccionario con los valores a buscar. De esta forma, mediante un bucle se recorre la

Estructura XML para parsear

```
<Project>
  <Scenes>
    <Scene>
      <Stage>
        <Sprites>
          <Sprite>
            <Scripts>
              <Script>
                <Block>
                  <Block s>
```

Figura 4.2: Estructura XML para parsear

Estructura General

Nombre del Proyecto:

*** Stage:**

Blocks

- Nombre del Bloque

> id

- Nombre del Bloque

> id

> next

> option

Costume

*** Sprite:**

Blocks

- Nombre del Bloque

> id

- Nombre del Bloque

> id

> next

> option

Costume

*** Sprite(2):**

Blocks

- Nombre del Bloque

> id

Costume

Figura 4.3: Estructura General de los Bloques

estructura de datos creada y, en caso de encontrar algún bloque indicado en el parámetro de entrada, dicha función devuelve True.

Para determinadas funciones, fue necesario crear diccionarios personalizados. Por ejemplo, cuando los bloques posean el campo “Costume” o “Option”, es necesario analizar cada uno de ellos individualmente antes de introducirlo en el nuevo diccionario. De esta forma, se evita que se repitan. En funciones posteriores, se analizarán dichos diccionarios para proceder con la evaluación de los distintos niveles.

Las dos funciones que conllevaron una mayor dificultad a la hora de su creación se localizan en el nivel Advanced y fueron las siguientes:

Para comprobar la existencia de bucles anidados, se sigue el esquema de la Imagen 4.4: Primero se verifica si existe algún bucle en el proyecto. Si se encuentra, se guarda el valor del campo “Next”, es decir, el nombre del bloque que se sitúa en el interior del bucle. Una vez se ha obtenido el nombre del bloque, se realiza su búsqueda en el diccionario, se guarda su identificador y, en caso de existir y comprobar que realmente es un bucle, se concluye que hay bucles anidados.

Otra posibilidad es que el bloque “Next” sea un if. En este caso, se continúa accediendo dentro de este bloque y se comprueba si dicho if posee a su vez su propio campo “Next”. Una vez verificada la existencia del campo “Next”, se realizarán los mismos pasos descritos en el párrafo anterior.

En la última opción el campo “Next” no es ni un if ni un bucle y por lo tanto habrá que comprobar si dicho bucle en su interior tiene más de una orden. En caso de poseer más de una orden, se analizará cada una. Para ello, se aumenta el identificador del bloque a encontrar y, una vez aumentado, se realizan los mismos pasos sucesivamente hasta haber analizado todas las órdenes que se encuentran en su interior.

Para comprobar la existencia de ifs anidados, se sigue el esquema de la Imagen 4.5. La función es bastante similar a la anterior puesto que hay que acceder al interior de los ifs e ir analizando los distintos bloques que se pueden encontrar. No obstante, en esta ocasión, se buscará un if en su interior en lugar de un bucle.

Para finalizar se crean dos diccionarios denominados “Vanilla” y “Extended” que representan dos formas de evaluación y que están formados por el concepto de evaluación y la puntuación obtenida. Las diferencias entre ambos son las siguientes: el “Vanilla” posee una puntuación máxima de tres (Proficient) y menos valores a analizar mientras que el “Extended”

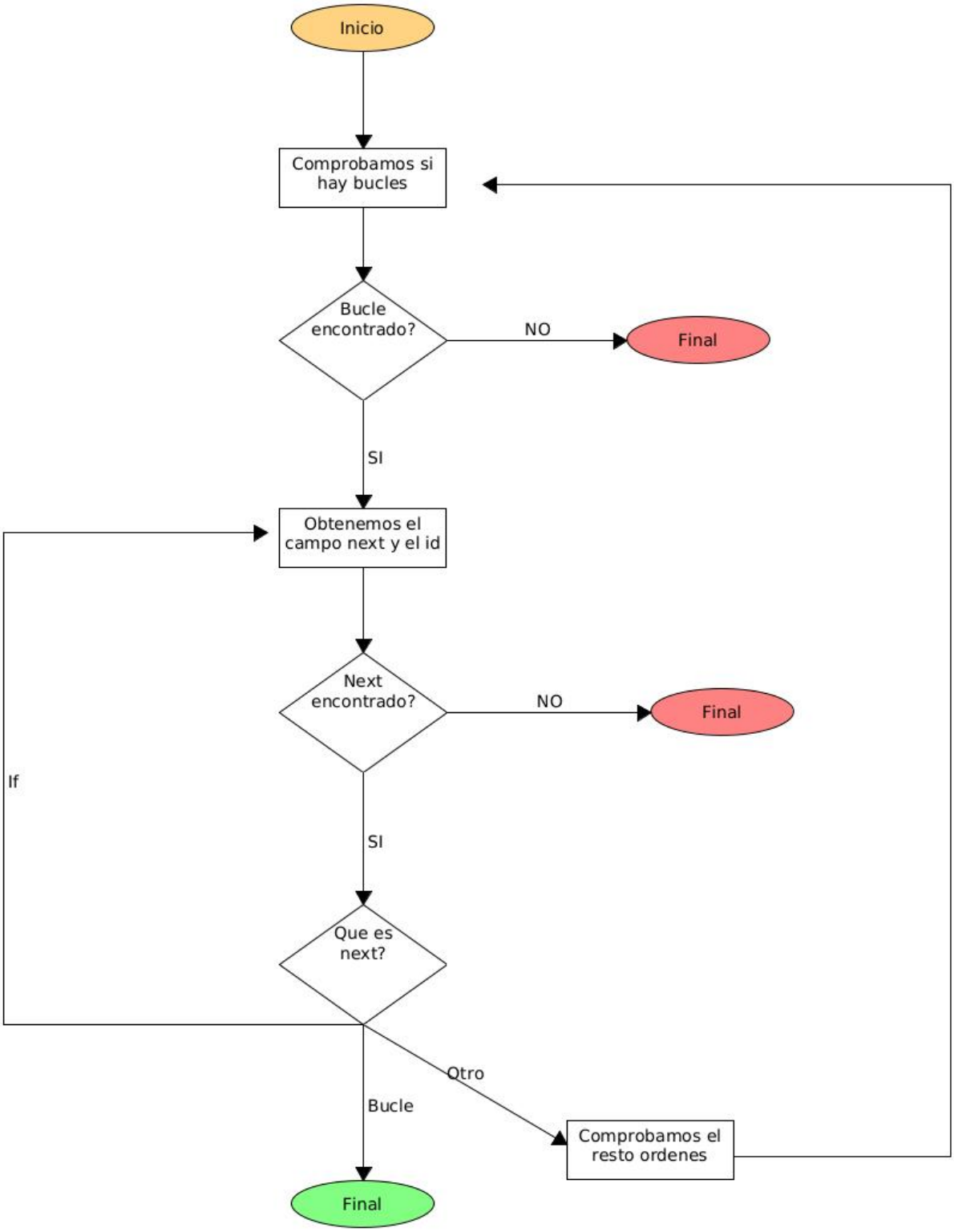


Figura 4.4: Estructura del código de búsqueda de bucles

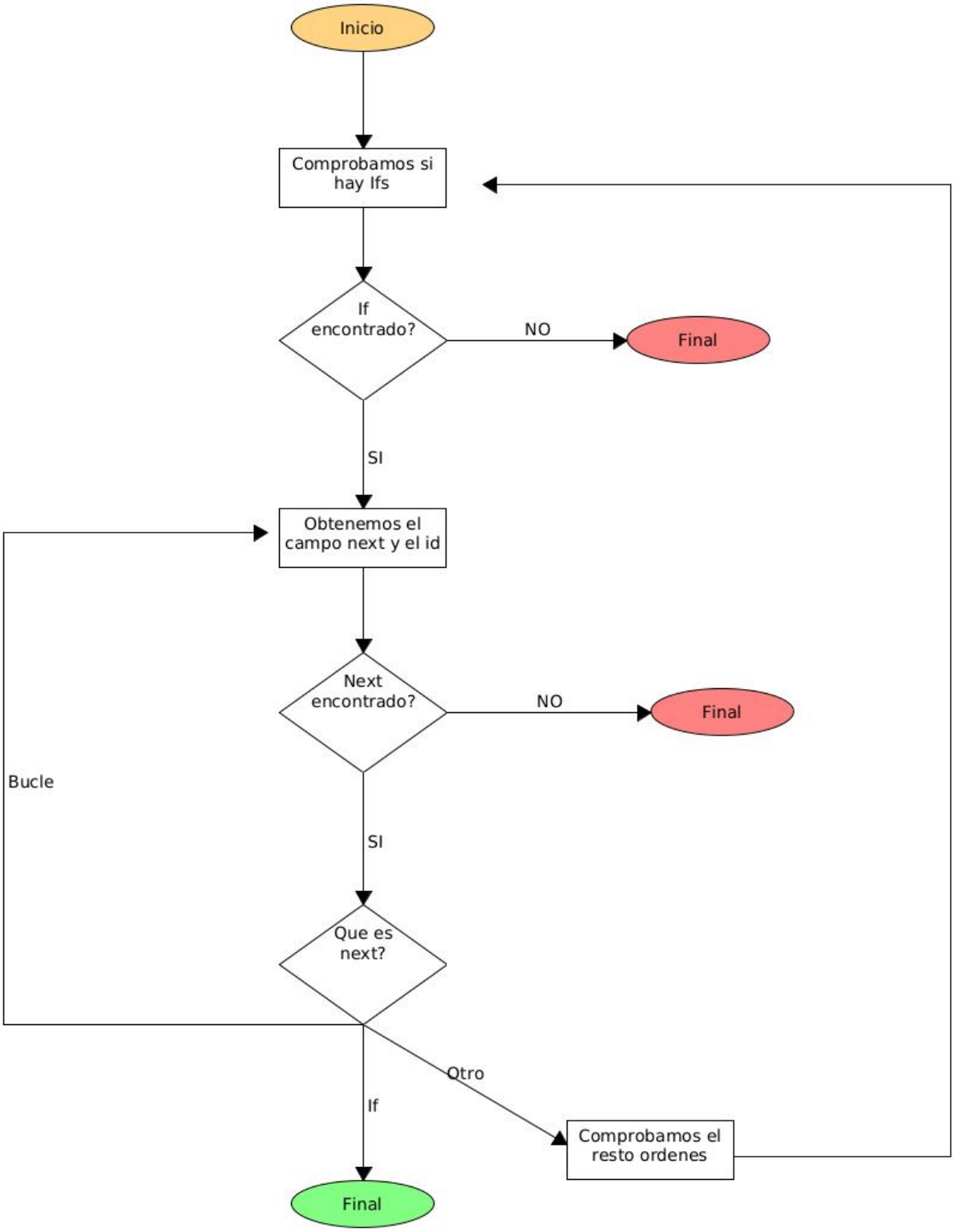


Figura 4.5: Estructura del código de búsqueda de ifs

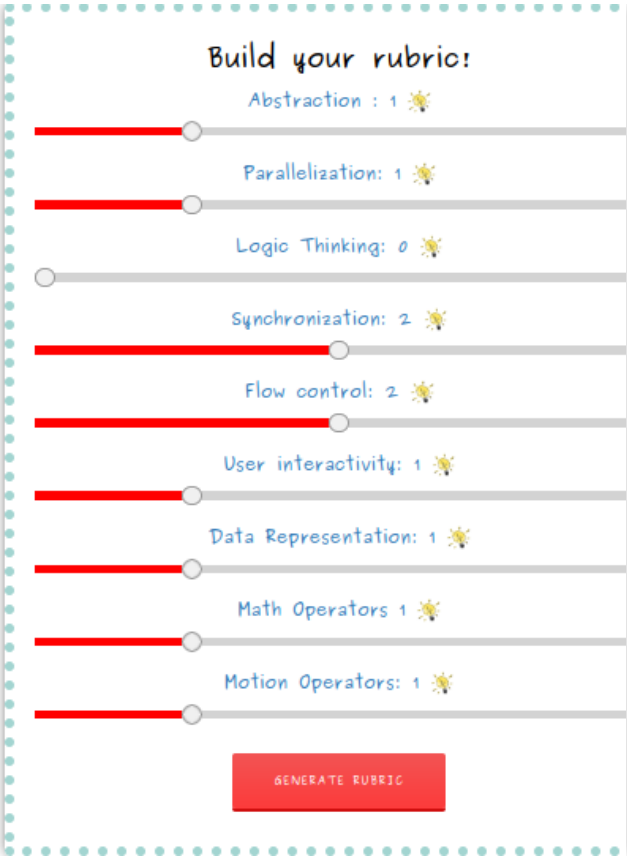


Figura 4.6: Primer ejemplo de rúbrica

Figura 4.7: Segundo ejemplo de rúbrica

dispone de una puntuación máxima de cuatro (Advanced) y se realiza un análisis completo de los conceptos de pensamiento computacional.

4.4. Modo Personalizado

En este modo se puede establecer los valores de medidas personalizados según el criterio de evaluación que se desee. El funcionamiento consta de un diccionario predeterminado con los valores de medida a evaluar, sin embargo, se modificará dicho diccionario con los valores elegidos.

Con el objetivo de evaluar determinados aspectos de los proyectos, véase “Arte” o “Música”, se han creado rúbricas personalizadas, tal y como se observa en la Figura 4.6 y en la Figura 4.7.

Una vez generada la rúbrica, se creará una URL a partir del servidor local. A continuación, se podrá indicar la URL del proyecto que se desea analizar o, directamente, subir el proyecto mediante el archivo XML. Finalmente, se lleva a cabo el análisis según los valores establecidos.

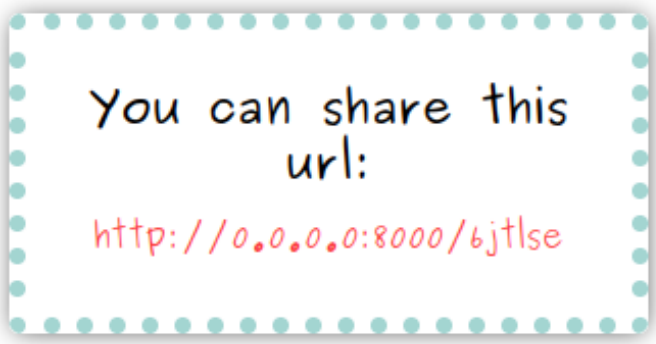


Figura 4.8: Ejemplo de URL personalizado

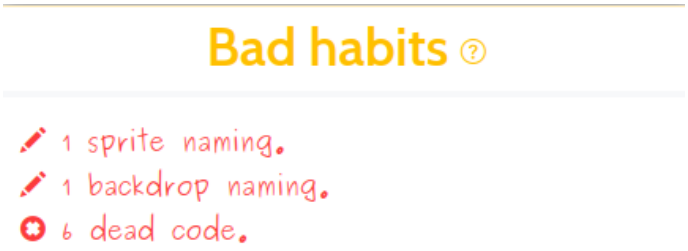


Figura 4.9: Vista del análisis realizado

4.5. Análisis de Bad Smells

Una vez realizado el análisis de cada proyecto, se procede a evaluar la presencia de "Bad Smells", término que hace referencia a malos hábitos frecuentes entre los programadores en los proyectos en el lenguaje de programación de Snap!.

Se identifican los tres "Bad Smells" más frecuentes y, para cada uno, se crea un script en Python que permita detectarlos (Figura 4.9).

En Snap!, el entorno de programación nombra por defecto al escenario como Stage y, al crear un nuevo escenario, este recibe el nombre Stage(1), y así sucesivamente. El uso de nombres predeterminados para los escenarios es una mala práctica y, por tanto, lo más óptimo es establecer un nombre personalizado a cada escenario. Así pues, se establece el análisis del Stage como el primer "Bad Smell".

Para realizar este análisis, se crea una función en Python, donde se usará la estructura que se ha creado anteriormente para cada proyecto. En primer lugar, se recorre dicha estructura comprobando que no se encuentran nombres por defecto y, en caso de encontrar alguno, se incrementa una variable que contabilizará el número de nombres por defecto y se guardarán los nombres para posteriormente mostrarlos.

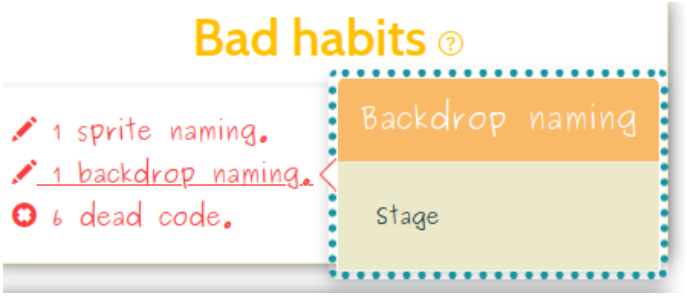


Figura 4.10: Vista del stage

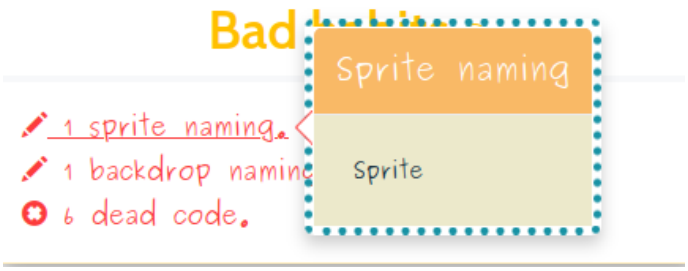


Figura 4.11: Vista del sprite

El segundo "Bad Smell" es el análisis de Sprite. Snap! sigue un patrón similar para crear los nombres, por defecto usa el nombre Sprite y, al crear uno nuevo, este recibe el nombre Sprite(1), y así sucesivamente. De esta forma, se crea una función similar a la anterior pero modificando los valores a buscar.

Por último, el análisis de código muerto. En este caso, se comprueba si determinados eventos, bucles o ifs, están formados por órdenes anidadas puesto que son términos que para ejecutarse necesitan una segunda orden que se encuentre en su interior (Figura 4.12).

4.6. Modo Asistente

En este modo, se analiza la URL de un proyecto. Sin embargo, en este caso solo es necesario comprobar los "Bad Smells" presentes, es decir, no se obtendrá una calificación como en anteriores modos. A partir de los "Bad Smells" identificados se proporcionará una posible solución.

Pueden surgir tres tipos de "Bad Smells":

- **Código muerto**
- **Sprite Naming**

Figura 4.12: Vista del código muerto

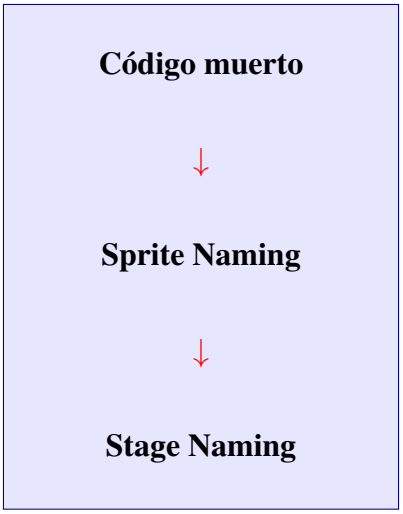


Figura 4.13: Jerarquía de Bad Smells

- **Stage Naming**

Se proporcionará una posible solución para cada uno estos problemas.

Para trabajar de acuerdo a este modo, se ha creado una jerarquía de “Bad Smells” a través de la cual se solucionan primero los problemas más importantes: se presentará la solución para los “Bad Smells” de nivel superior y, a continuación, los de nivel inferior.

4.7. Certificado del proyecto

Una vez finalizado el análisis del proyecto, existe la posibilidad de generar un certificado y descargarlo en tu propio ordenador. Este certificado se crea utilizando la herramienta LaTeX y

Figura 4.14: Ejemplo de certificado

emplea los siguientes datos: la URL analizada y la puntuación obtenida.

Al comprobar la estructura general de las URLs de Snap, se verifica que hacen uso de caracteres especiales. Estos son conflictivos, no siendo soportados por LaTeX y por este motivo, se crea una función que los sustituye por caracteres legibles. Una vez realizado este cambio, se genera el nombre del certificado con extensión .pdf.

Con el fin de crear un certificado personalizado (Figura 4.14) para cada proyecto, se utiliza la imagen base generada como parte del Trabajo Fin de Grado y, a través de LaTeX se completa el certificado añadiendo el siguiente texto: la URL y la puntuación.

Capítulo 5

Ejemplos de uso

A continuación se muestran las distintas vistas creadas mediante HTML y las plantillas utilizadas.

- Vista del inicio de la aplicación web (Figura 5.1 y Figura 5.2). Se observa una breve descripción, los distintos modos que se pueden emplear, un cuadro donde se indica la URL a analizar y otro donde se realiza la subida del archivo XML. Respecto a los modos Personalizado y Asistente, aparece una explicación sobre su forma de uso, acompañada de una imagen de ejemplo.

- Vista de la pantalla de carga entre diferentes funcionalidades de la aplicación web (Figura 5.3). Aparece mientras se realiza el análisis solicitado, en cualquiera de los distintos modos.

  En ella se muestra un mensaje de espera y una barra de carga, junto con el logo de Dr. Snap!.

- Vista de la pantalla donde se muestran los resultados obtenidos en modo "Vanilla" (Figura 5.4).

- Vista de la pantalla donde se muestran los resultados obtenidos en modo "Extended" (Figura 5.5).

  En las Figuras 5.4 y 5.5 se muestran los resultados del análisis: la puntuación general, la puntuación obtenida en cada apartado, los "Bad Smells" encontrados y, por último, la

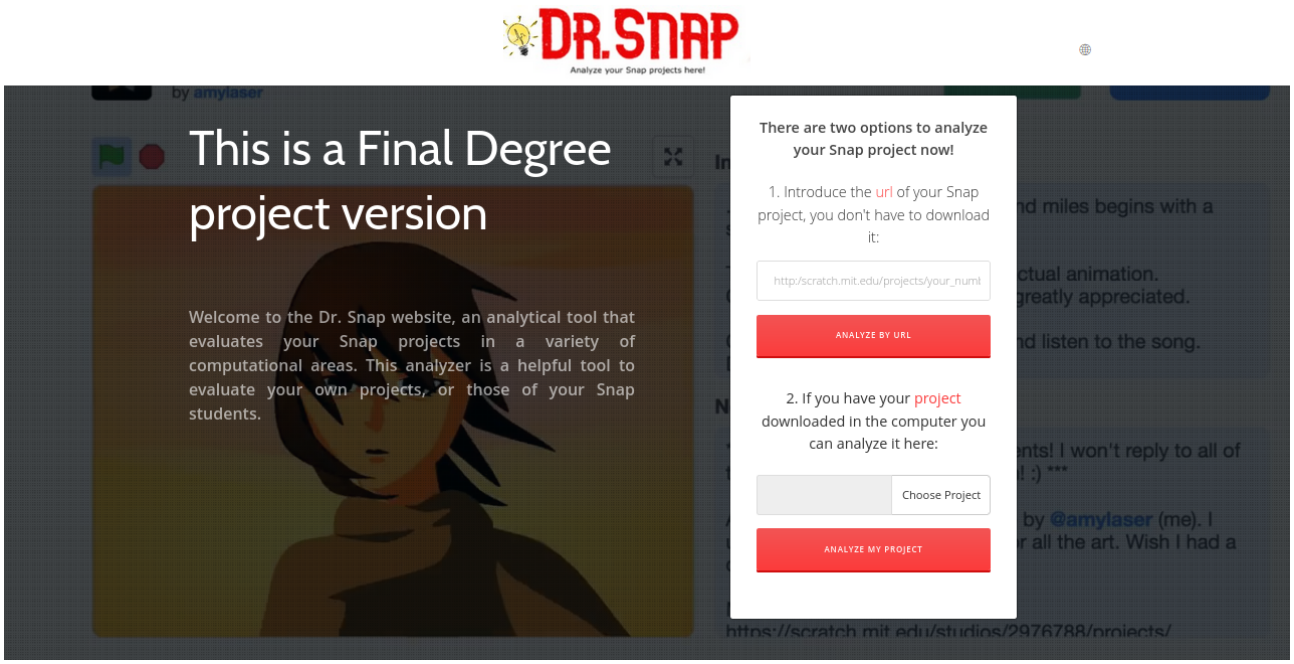


Figura 5.1: main.html

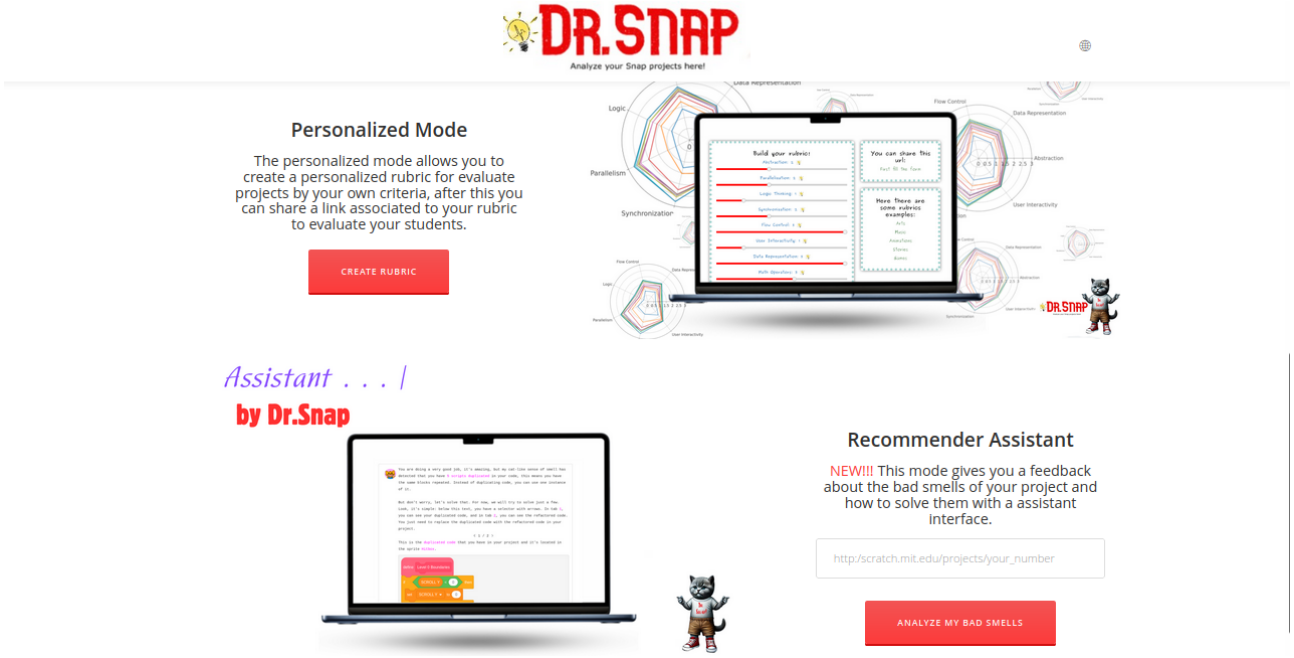


Figura 5.2: main.html

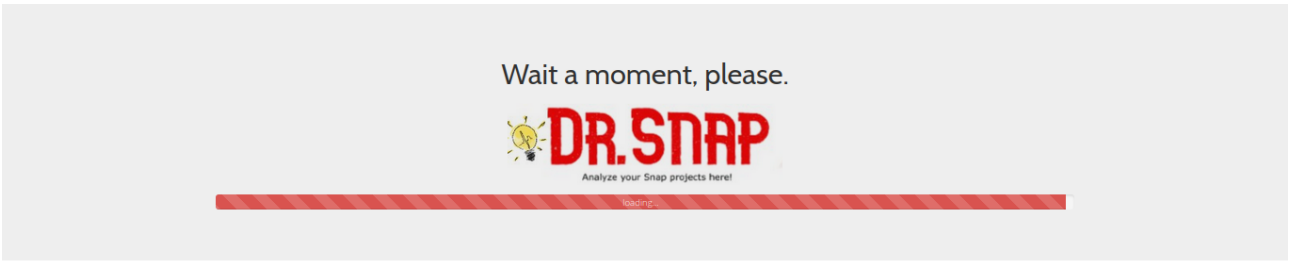


Figura 5.3: Pantalla de carga

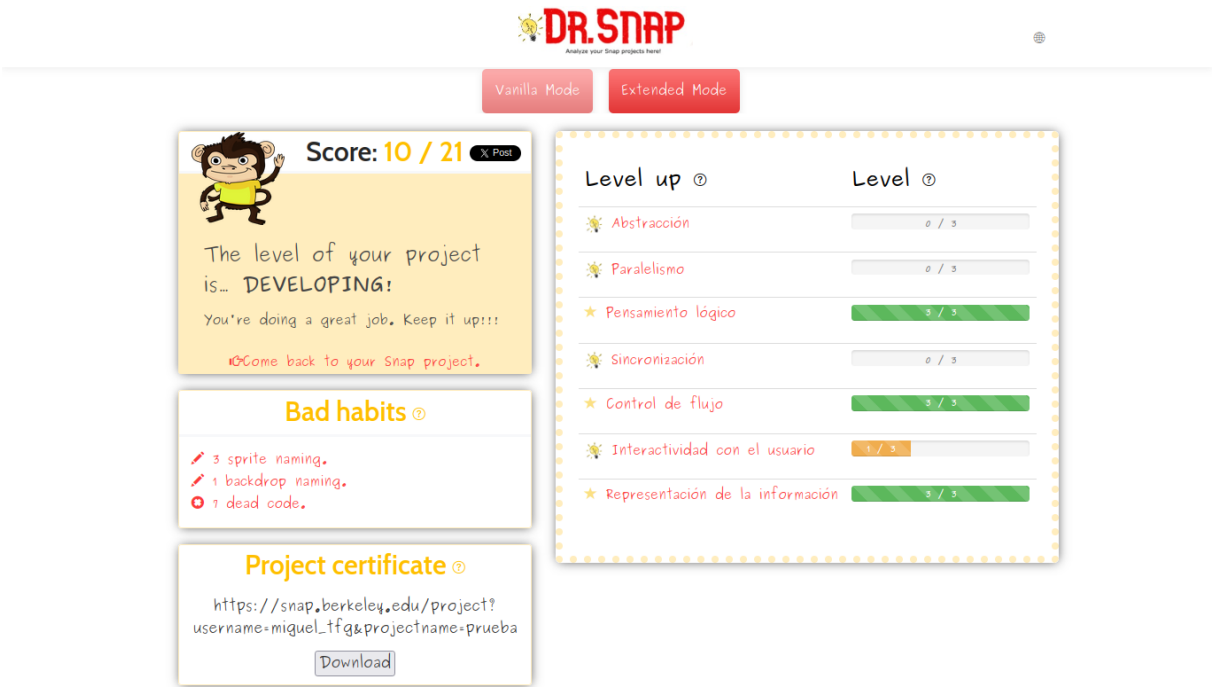


Figura 5.4: dashboard-default.html

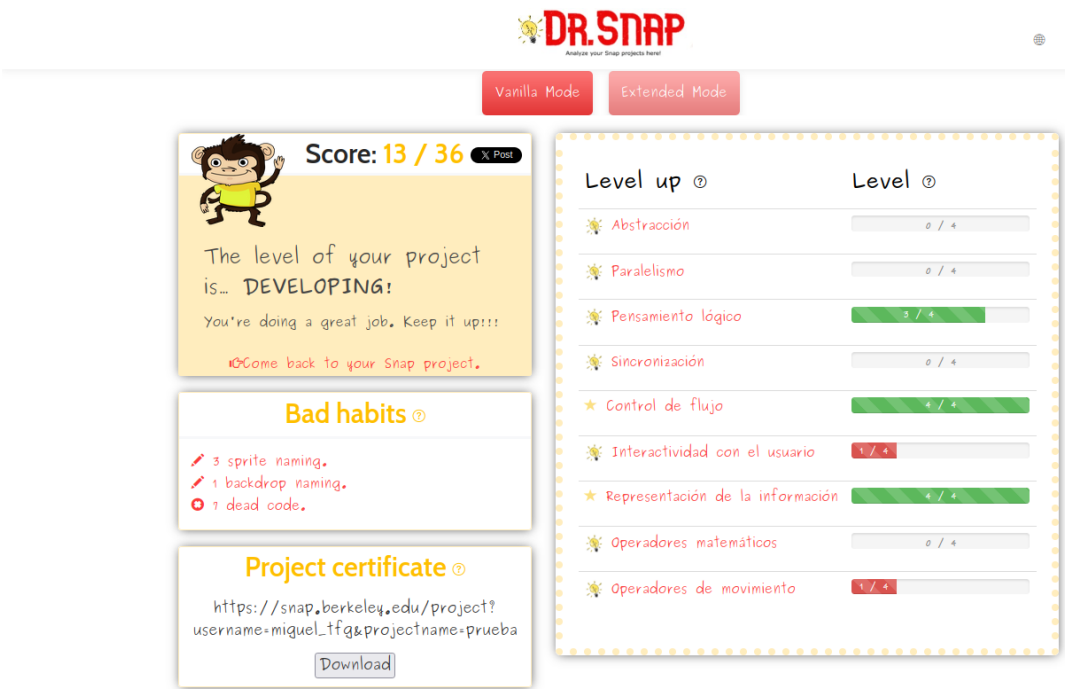


Figura 5.5: dashboard-default.html

opción de descargar el certificado.

- Vista del modo personalizado (Figura 5.6). Se encontrará una plantilla personalizable según los valores a los que se les desee dar una mayor o menor importancia. En la esquina superior derecha aparecerá el enlace una vez se haya generado la rúbrica. Por último, se muestran rúbricas de ejemplo ya creadas y de las cuales se puede hacer uso.

  Este modo está dirigido a los profesores para evaluar proyectos.

- Vista del modo asistente (Figura 5.7).

  Se observa el logo de Dr. Snap!, junto a un mensaje que describe el error encontrado, mostrando el bloque que lo ha generado y una explicación detallada.

  Este modo, enfocado en los estudiantes, simula un profesor virtual.

  Para comprobar el correcto comportamiento de la aplicación, se han creado varios proyectos en Berkeley Snap! que se pueden observar en la Figura 5.8.

  En primer lugar, se crean archivos como prueba, p2, option, cuya funcionalidad es comprobar el correcto análisis de los distintos bloques que se ejecutan en cada proyecto.

Figura 5.6: dashboard-personal.html

Figura 5.7: dashboard-recommender.html

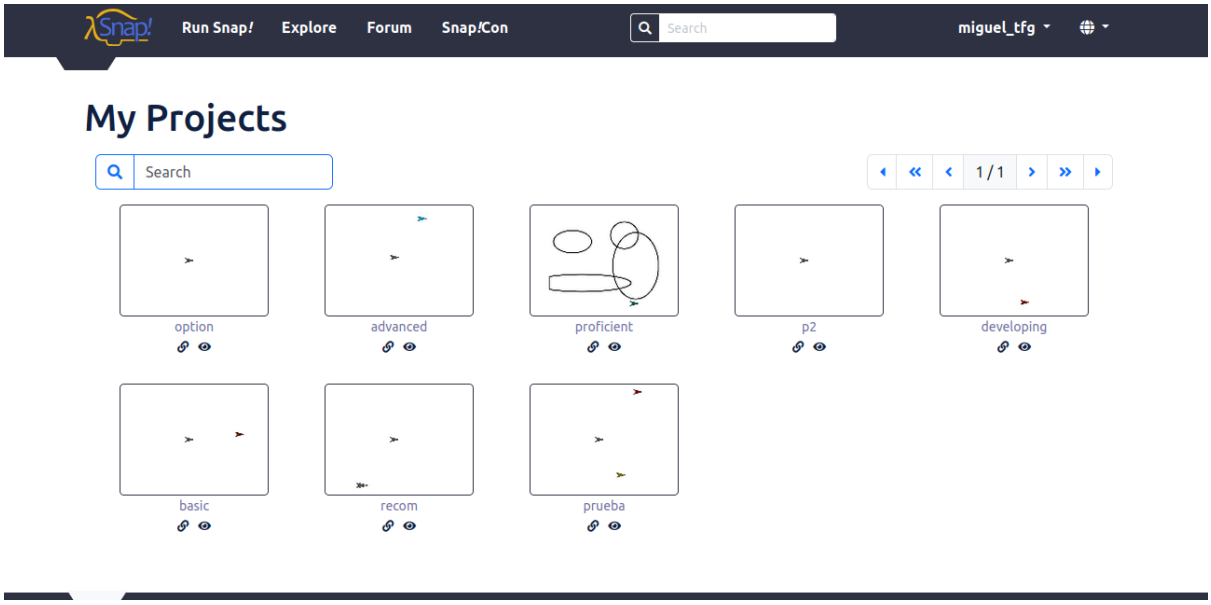


Figura 5.8: Vista general de los proyectos

Dichos proyectos, debido a su simpleza, se han modificado en multitud de ocasiones con el objetivo de poder verificar que se analizan correctamente los cambios introducidos.

A continuación, se llevan a cabo distintos proyectos (Basic, Developing, Proficient y Advanced) según el nivel de análisis. El objetivo de estos es obtener una puntuación en cada apartado, la cual indicará el nombre del proyecto. De esta forma, en el nivel Basic se obtendrá en todos los aspectos del proyecto analizado, una calificación de Basic (Nivel 1), tal y como se muestra en la Figura 5.9.

Finalmente, se analizan decenas de proyectos de la página oficial de Berkeley Snap!.

El proyecto que obtuvo mayor puntuación se identifica como “Winterrest” y se puede encontrar en el siguiente enlace: `https://snap.berkeley.edu/project?username=lagsonly&projectname=Winterrest`. Este es un juego donde se controla a una ardilla mediante el teclado y cuyo objetivo es ir obteniendo bellotas e ir avanzando a los siguientes niveles.

A continuación, tal y como se representa en la Figura 5.10, se desglosará la puntuación obtenida en el modo “Extended”, siendo esta la que presenta una mayor dificultad. Para ello, se accede al código que ha creado este proyecto y que está formado por ocho Sprites y un único Stage:

- **Abstracción,** uso avanzado de clones. Recibe el bloque “receiveOneClone”, donde

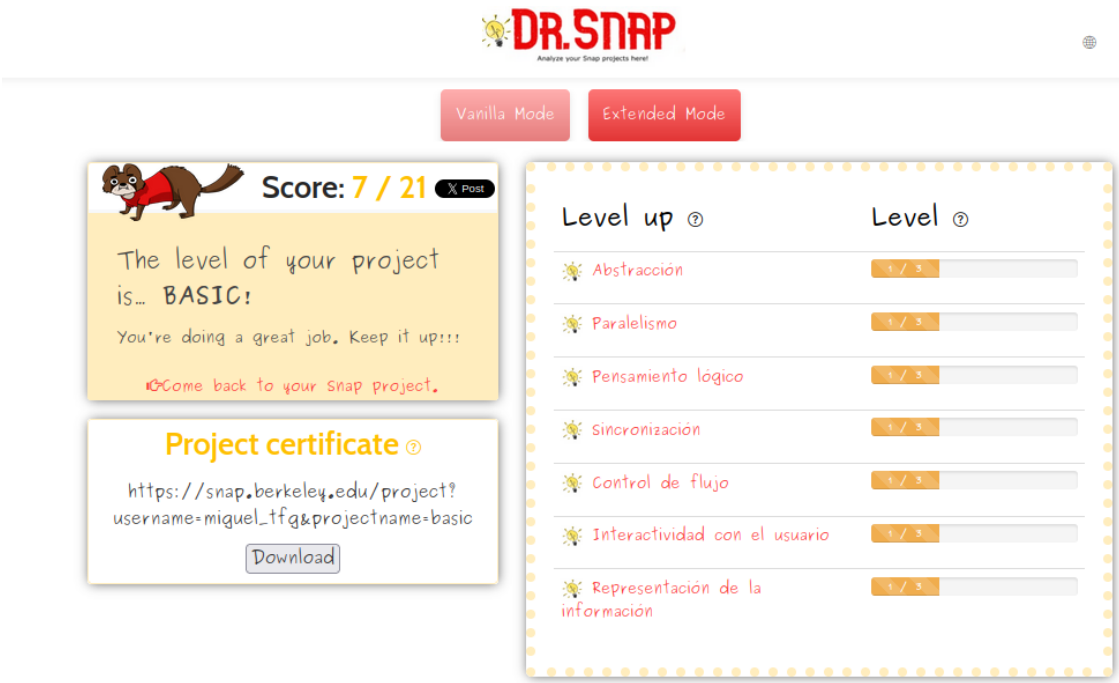


Figura 5.9: Análisis del proyecto Basic

Figura 5.10: Rúbrica del proyecto Winterrest en modo Extended

se comprueba si el campo “Next” es un bloque “broadcast”, un bucle o ifs. En caso positivo recibe los cuatro puntos.

- **Paralelismo,** uso de una gran cantidad de disfraces. Este aspecto se puede comprobar de manera visual, accediendo en cada Sprite al apartado “Costume”. En los Sprites llamados Level y Stage es donde se puede observar el mayor número de Costumes.
- **Lógica,** uso de ifs anidados. Se puede observar en el Sprite Props.
- **Sincronización,** uso de alguno de estos bloques: “doWaiUntil”, “doBroadcast”, “receiveOneclone” o “receiveCondition”. Al no realizar un manejo dinámico de los mensajes no obtiene la mayor puntuación. Además, debido a que supera el mínimo establecido de uso de estos bloques (“doBroadcast” o “doBroadcastAndWait”), obtiene tres puntos.
- **Control de flujo,** hace uso de bucles pero no obtiene la máxima puntuación al no crear bucles anidados.
- **Interactividad con el usuario,** solo es necesario el uso del teclado y en consecuencia, alcanza una puntuación de dos.
- **Representación de la información,** uso de variables booleanas: “=”, “and” o “or”. Consigue la máxima puntuación (cuatro puntos).
- **Operadores matemáticos,** uso de operaciones en cadena utilizando los siguientes bloques: “reportJoinWords”, “reportLetter” o “reportTextAttributate”. Al no emplear fórmulas trigonométricas no obtiene la mayor puntuación (tres puntos sobre cuatro).
- **Operadores de movimiento,** ejecuta una gran cantidad de movimientos logrando la máxima puntuación (cuatro puntos).

Por otro lado, en la Figura 5.11 se representa la puntuación que ha obtenido este proyecto, según el modo “Vanilla”.

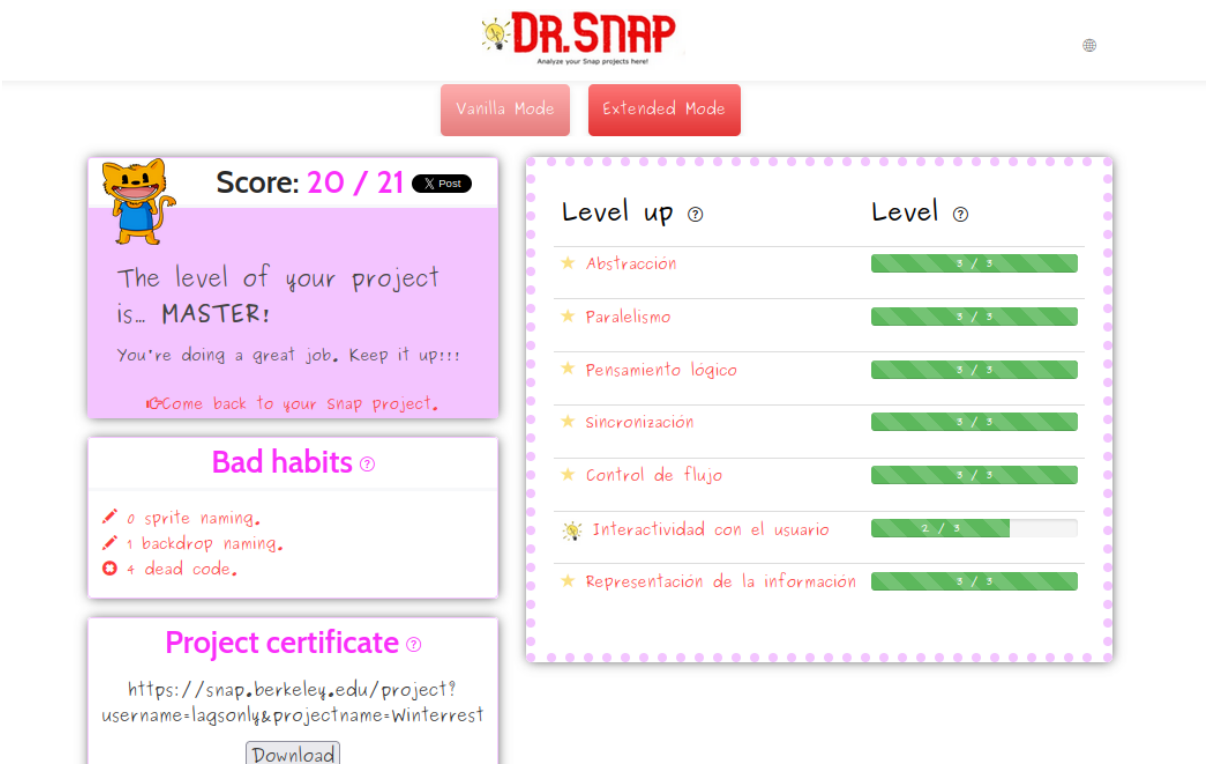


Figura 5.11: Rúbrica del proyecto Winterrest en modo Vanilla

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

Debido a que la aplicación web desarrollada en este Trabajo Fin de Grado realiza un correcto análisis de los proyectos mediante el uso de URL o mediante la subida de un archivo XML, se puede concluir que se han cumplido los distintos objetivos establecidos.

La aplicación web ofrece diversas funcionalidades que pueden llegar a ser muy útiles para el ámbito académico tanto por el propio análisis como por la creación de una rúbrica personalizada. Asimismo, ofrece al estudiante un análisis de "Bad Smells" y un modo asistente que puede mejorar sus proyectos.

Cabe destacar que a lo largo del proyecto han surgido una gran cantidad de problemas. A pesar de que, generalmente han sido problemas de programación, han surgido otros como el descrito a continuación:

En el desarrollo de la aplicación web se trabaja con multitud de archivos HTML. Estos archivos poseen grandes cantidades de líneas de código y por lo tanto, era tedioso buscar línea a línea los fragmentos a modificar. Como consecuencia, se crea un script de Shell para Linux, llamado "searchstring.sh" con el objetivo de llevar a cabo una búsqueda avanzada y rápida. Este script recibe como parámetro obligatorio un string y busca todos los ficheros HTML del directorio actual y de los hijos. Cuando encuentra un archivo HTML, busca el parámetro de entrada y, en caso de encontrarlo, presenta por la salida del terminal la ruta donde se sitúa el archivo que se está analizando, la línea en la que

se encuentra y la frase donde se está usando dicho string. Trabajará así recursivamente hasta que se analicen todos los archivos de los directorios.

6.2. Aplicación de lo aprendido

A lo largo del Grado se han adquirido una serie de conocimientos y habilidades que han sido fundamentales para el desarrollo de este Trabajo Fin de Grado, en particular:

1. **Fundamentos de la programación:** es la primera asignatura de programación que se cursa en el Grado, y se trabajó con el lenguaje Ada, donde se aprende a pensar como un buen programador.
2. **Programación de sistemas de telecomunicación:** primer contacto con Python, con la que se asimila la sintaxis del lenguaje y con la que se crean proyectos de Cliente-Servidor.
3. **Servicios Telemáticos:** con ella, se comprende cómo usar Django para diferentes proyectos web.
4. **Aplicaciones telemáticas:** se aprende la forma de crear páginas web mediante el uso de HTML, CSS y JavaScript.

6.3. Lecciones aprendidas

Gracias a la realización del Trabajo Fin de Grado, a nivel personal, he mejorado mi nivel en diferentes aspectos:

1. Creación de páginas web mediante el uso de HTML y CSS de manera más profesional.
2. Comprensión del modo de analizar un archivo XML y, de manera más óptima, el funcionamiento de las aplicaciones web, gracias al uso de Python.

6.4. Trabajos futuros

A pesar de que la aplicación web se ha desarrollado satisfactoriamente, se podrían añadir distintas funcionalidades.

En primer lugar, sería posible implementar la comparación de varios proyectos con el objetivo de identificar en qué difieren o, incluso, para comparar varias versiones del mismo proyecto. Mediante la puntuación de ambos se comprobará cuál es el más óptimo.

Otra alternativa sería posibilitar la subida de un archivo .txt donde aparezcan múltiples URLs. De esta forma, se analizaría cada una de ellas y se mostraría un resumen de los resultados obtenidos.

Por último, se podría crear un manual de referencia en el que se detallase las funciones que posee la aplicación web, incluyendo una breve explicación de los elementos que se analizan y ejemplos visuales.

Bibliografía

[1] Ionos web. Technical report, Ionos. https://www.ionos.es/digitalguide/servidores/know-how/modelo-cliente-servidor/.

[2] Documentación oficial de python 3. Technical report, Python Software Foundation, 2001–2025. https://docs.python.org/es/3/tutorial/.

[3] Historia del software python 3. Technical report, Python Software Foundation, 2001–2025. https://docs.python.org/es/3.13/license.html.

[4] Tutorial de uso de la librería xml.etree.elementtree. Technical report, Python Software Foundation, 2001–2025. https://docs.python.org/es/dev/library/xml.etree.elementtree.html.

[5] Página oficial de django. Technical report, Django Software Foundation, 2005–2025. https://www.djangoproject.com/.

[6] *Simposio de Ubicuidad: Computación y Pensamiento Computacional Aho, Alfred V.* 2nd ed. (international ed.). edition, 2011.

[7] E. W. Eric Meyer. *CSS: La Guía Definitiva, 5ª Edición*. O'Reilly Media, Inc, 2024.

[8] J. Fagerlund, P. Häkkinen, M. Vesisenaho, and J. Viiri. Computational thinking in programming with scratch in primary schools: A systematic review. *Computer Applications in Engineering Education*, 29(1):12–28, 2021.

[9] J. Fawcett, D. Ayers, and L. R. Quin. *Beginning XML, 5th Edition*. Wrox, 2012.

[10] D. Flanagan. *JavaScript: La Guía Definitiva, 7ª Edición*. O'Reilly Media, Inc, 2024.

[11] B. Harvey, D. D. Garcia, T. Barnes, N. Titterton, D. Armendariz, L. Segars, E. Lemon, S. Morris, and J. Paley. Snap! (build your own blocks). In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 759–759, 2013.

[12] B. Hyslop. *The HTML.* 1st edition. edition, 2010.

[13] S. Kottwitz. Getting started with latex. In *LaTeX Beginner's Guide*. Packt Publishing, Limited, United Kingdom, 2021.

[14] B. Lubanovic. *Introducción a Python, 2ª Edición*. O'Reilly Media, Inc, 2024.

[15] M. Lutz. *Aprender Python, 5ª Edición*. O'Reilly Media, Inc, 2024.

[16] R. C. Martin. Clean code / robert martin., 2016.

[17] J. Moreno, G. Robles, M. Román, and J. D. Rodríguez. Not the same: a text network analysis on computational thinking definitions to study its relationship with computer programming. *RiiTE Revista interuniversitaria de investigación en tecnología educativa*, 2019.

[18] J. Moreno-León, G. Robles, and M. Román-González. Dr. scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, (46):1–23, 2015.

[19] J. M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.