Universidad
Rey Juan Carlos

GRADO EN INGENIERÍA EN SISTEMAS AUDIOVISUALES Y MULTIMEDIA

Curso Académico 2021/2022

Trabajo Fin de Grado

ANÁLISIS DE CLONADO Y ABSTRACCIÓN EN SCRATCH

Autor : Felipe Enmanuel Sandoval Sibada

Tutor : Dr. Gregorio Robles

Trabajo Fin de Grado

Análisis de Clonado y Abstracción en Scratch

Autor : Felipe Enmanuel Sandoval Sibada

Tutor : Dr. Gregorio Robles

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2022, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2022

Dedicado a

mi madre, por su apoyo incondicional.

II

Agradecimientos

No ha sido fácil...

Salir de mi zona de confort ha sido una de las decisiones más difíciles y, a la vez, gratificantes que he tomado en mi vida. En el año 2014 decidí embarcarme hacía una de las mejores ciudades en las que he podido vivir, mi querida Madrid. Evidentemente no ha sido fácil dejar atrás a familiares, amigos, costumbres y un largo etcétera, pero cambiar de rutina hizo que me volviese una persona independiente, curiosa y resiliente.

Quiero agradecer a mis padres por su apoyo, comprensión, entendimiento y soporte. A mi madre, Flor Marjorie, por siempre estar ahí en cada momento, gracias a ti me convertí en la persona que soy hoy en día. No son suficientes las palabras para expresar mi gratitud, te quiero un mundo.

Gracias a ti, Nathalie, por escucharme y entenderme, por tu paciencia en momentos difíciles, por ayudarme a desconectar y por nunca dejar que tirase la toalla. Me demuestras que la motivación es importante pero que sin disciplina y sin hábitos de estudio, de poco vale. Estoy seguro que el futuro nos deparará momentos y vivencias memorables que podremos compartir. Te adoro.

A mis amigos, gracias por su comprensión y por seguir contando conmigo para todos los planes. Son incontables las experiencias vividas y los *"tours"* que hicimos en casi todas las bibliotecas de Madrid. Gracias por enseñarme que en equipo la universidad es más fácil.

Por último quiero agradecer a mi tutor, Gregorio, por brindarme la oportunidad de trabajar junto a él y demostrarme que la enseñanza es un tema de vocación. Con este proyecto he ratificado la importancia de motivar y dotar de herramientas a las generaciones más jóvenes, ya que solo así será posible desarrollar su potencial profesional y humano. Muchas gracias por tu paciencia, amabilidad y tiempo, especialmente porque, a pesar de haber atravesado una pandemia

mundial, al final hemos logrado salir adelante con este proyecto.

He sufrido y disfrutado con la experiencia universitaria, conocí personas muy especiales durante esta etapa que han y siguen siendo muy importantes en mi vida.

Gracias a este viaje cuento con una caja de herramientas que me permitirán ejercer y desarrollarme en el apasionante oficio de la ingeniería y el mundo de las TIC. A cualquiera que lea este proyecto, ¡Muchas Gracias!

Y sí, no ha sido fácil... pero volvería a repetirlo.

Resumen

En este trabajo se busca desarrollar una herramienta capaz de extraer y detectar duplicidad de código en proyectos de Scratch. Durante el proceso, se analizan los distintos bloques que conforman cada proyecto y se obtiene como resultado una lista con los conjuntos que mayor duplicidad presenten. Para lograr este objetivo se realiza un estudio estadístico tanto a nivel *intra-sprite* como a nivel *project-wide* haciendo uso del algoritmo de clustering de propagación por afinidad.

Esta idea surge a partir de otras herramientas que evalúan aspectos del pensamiento computacional, el cual abarca habilidades como el racionamiento lógico, la sincronización, el paralelismo, la representación de la información y la abstracción. Es el impacto de la duplicidad en esta última habilidad la que motiva la ejecución de este trabajo.

La herramienta se desarrolla usando Python como principal tecnología. Durante el trabajo se realizarán tests unitarios y guías de estilo para mejorar la calidad del código. Por último, se explicarán los resultados y analizarán, concluyendo si se consiguen o no, los objetivos planteados.

Summary

This work aims to develop a tool capable of extracting and detecting code duplicity in Scratch projects. The different blocks that make up each project are analyzed to obtain a list of the most duplicated ones. In order to achieve this, a statistical study is carried out at both *intra-sprite* and *project-wide* levels, using the affinity propagation clustering algorithm.

The project idea arises from other tools that evaluate aspects such as computational thinking, which encompasses skills like logical reasoning, synchronization, parallelism, information representation, and abstraction. Impact of duplication on the last ability motivates the execution of this work.

The tool is developed using Python as the main technology. Through the work, unit tests and style guides are carried to improve the code quality. At last, results will be explained and analyzed, concluding whether or not the objectives are achieved.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En este trabajo se busca crear una herramienta que, a partir del análisis de bloques de código en Scratch, pueda dar una visión general sobre la duplicidad de código en un proyecto. Se entenderá por duplicidad de código aquellos bloques que se repitan tanto a nivel *intra-sprite* como *project-wide*, considerando que los baremos de penalización son distintos según cada caso. Esto es así por consecuencia de los límites de la propia aplicación que restringe la re-utilización de código entre objetos de forma eficiente.

Antes que nada, es importante explicar una serie de conceptos del glosario de Scratch[1].

- Un bloque o *block* es una pieza de código que ejecuta una acción. A nivel visual en la interfaz de Scratch, tienen formas de pieza de puzle con distintos colores y funcionalidades. En la imagen 1.1 se pueden apreciar varias estructuras de bloques. Más adelante se detallan los distintos tipos de bloques y las consideraciones especiales a tener en cuenta.

- Un *sprite* son los objetos de mi proyecto. A nivel visual, un sprite puede ser el fondo de tu escenario o una imagen, a modo de personaje. En la imagen 1.2 se puede apreciar un ejemplo de un *sprite*.

- Una función o *script* es una colección de bloques. Su orden no es trivial, ya que determina cómo interactúan los *sprites* entre sí, con el escenario y con el usuario. A nivel visual es una pila de varias piezas de puzle agrupadas.

- Un bloque personalizado o *custom block* son las funciones definidas por el usuario. Este

[1] `https://en.scratch-wiki.info/wiki`

tipo de bloques permite crear funciones sin pensar en los tipos de bloques que contiene, por eso constituye una de las principales muestras de abstracción presentes en Scratch. También ayudan a reducir el tamaño de bloques del proyecto, ya que si un usuario tiene una larga cadena de código que sabe que usará mucho, tiene sentido usar un bloque personalizado. De esta manera, no se tiene que repetir código varias veces [1]. En la imagen 1.3 se puede apreciar un *custom block*, caracterizados por su color rosa, y un *script*.

.

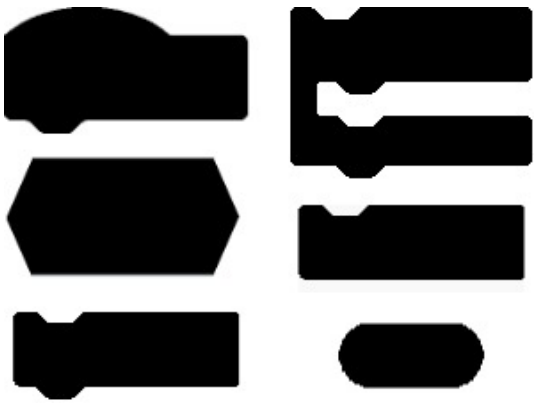

Figura 1.1: Estructura de varios bloques y sus distintas formas

Es importante destacar que los *custom blocks*, debido al funcionamiento y diseño de Scratch, solo tienen utilidad dentro del *sprite* dónde se definan. Es por esta razón que se realizan estudios *intra-sprite* y *project-wide*, ya que la única manera de reutilizar código entre en objetos es a través del copia y pega.

Para poder analizar la duplicidad de bloques de código en Scratch es necesario extraer la información del proyecto. Todos los datos de variables, objetos y bloques están contenidos en el fichero JSON dentro de la extensión *.sb3* que usa Scratch. Una vez se obtiene esta información es necesario procesar, según una serie de parámetros, los bloques que más se repiten. Luego, el análisis de dichos datos se realiza mediante la asignación de familias de grupos entre puntos de datos, mejor conocido como algoritmo de propagación por afinidad. Se emplea este algoritmo ya que considera cada punto de datos como un nodo en una red que recursivamente itera hasta tener un buen conjunto de ejemplares para clasificar grupos [2].

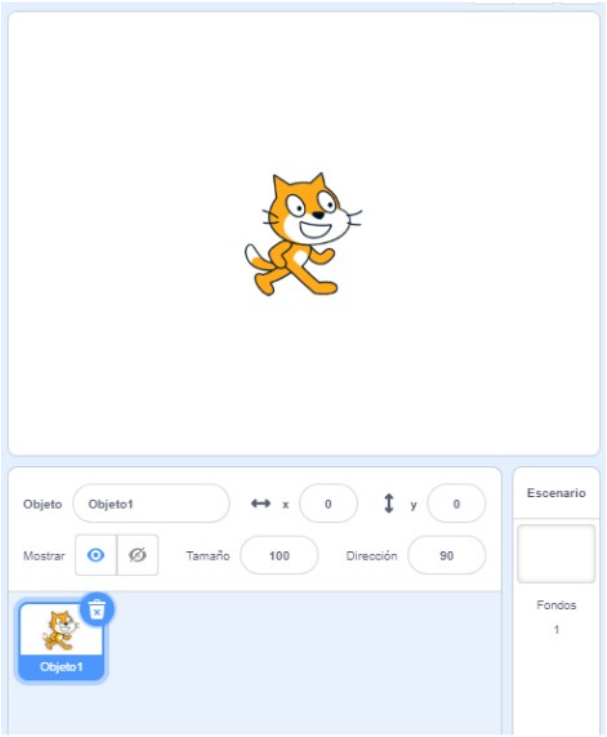

Figura 1.2: Sprite u objeto

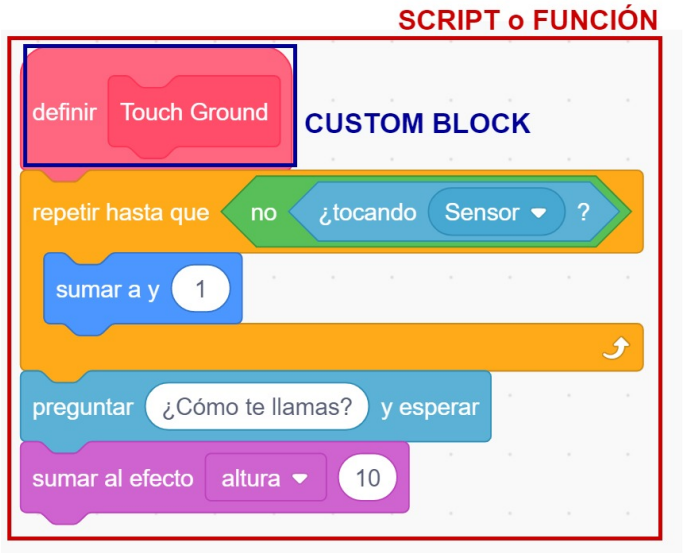

Figura 1.3: Script de un custom block

Ejemplos de código duplicado

A continuación, a modo ilustrativo, se muestran varios casos de duplicidad. Las imágenes que ilustran esta sección se obtienen del fichero *test.sb3*.

- En la imagen 1.4 se observa un ejemplo de duplicidad de tipo *intra-sprite*. En este caso ambos scripts realizan las mismas acciones con la diferencia que afectan a variables distintas.

- En la imagen 1.5 se observa un ejemplo de duplicidad a nivel *project-wide*. La función mostrada realiza exactamente lo mismo pero en dos objetos diferentes.

- En la imagen 1.6 se observa un ejemplo de duplicidad de dos *custom blocks*. Como se comentó previamente, por limitaciones de la plataforma no es posible reutilizar bloques de código personalizados en varios *sprites*. Este caso lo ilustra perfectamente.

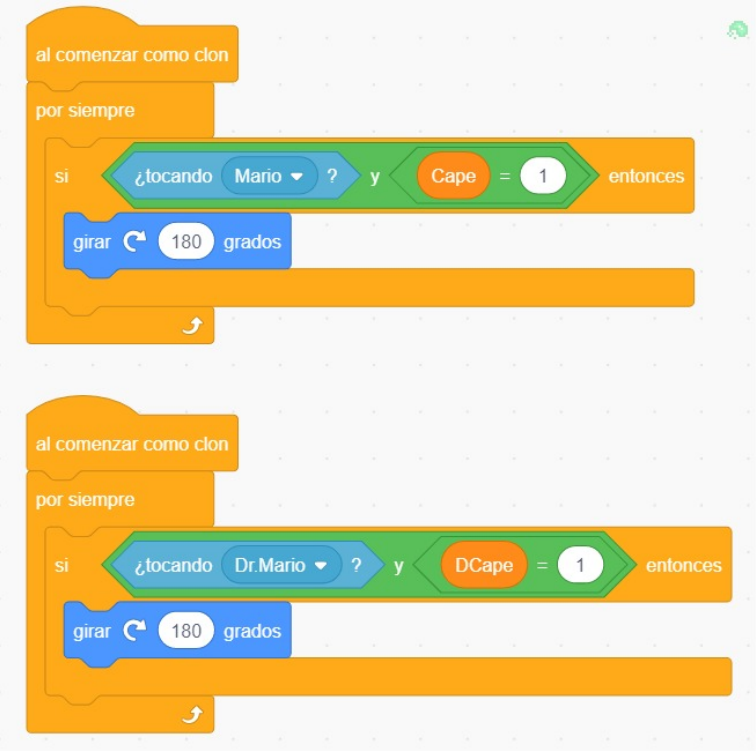

Figura 1.4: Scripts duplicados a nivel intra-sprite

1.1. Contexto Personal

Como alumno del Grado de Ingeniería en Sistemas Audiovisuales y Multimedia (GISAM) he adquirido distintas competencias durante los años de carrera. Las asignaturas de programación son las que mayor interés suscitaron en mí. Después de unos años en los que por motivos

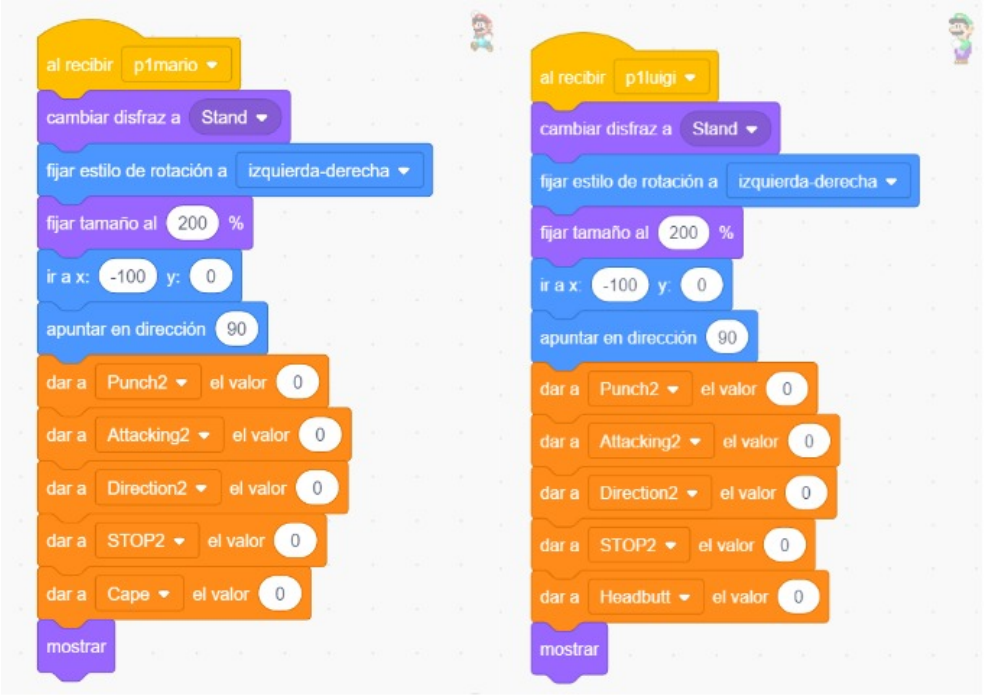

Figura 1.5: Ejemplo de duplicidad project-wide

profesionales me alejé de este campo, se me presentó la oportunidad de desarrollar una herramienta de software libre enfocado en aplicaciones de carácter pedagógico. La propuesta suponía un interesante reto que decidí afrontar.

1.2. Motivación

Según se ha podido comprobar en la literatura científica, a mayor existencia de duplicidad de código menos uso se hace de la abstracción, lo que ocasiona una mala síntesis en la descomposición de problemas [3]. Los proyectos de Scratch hacen uso extensivo del clonado (p.ej. copia y pega) lo que puede suponer una limitación a la hora de desarrollar distintas habilidades del pensamiento computacional.

Entendemos la abstracción como un proceso científico que busca, primero, definir un modelo que permita pensar y descomponer un problema fundamental en problemas mas sencillos, y segundo, facilitar la proyección de técnicas que solucionen dichos problemas. Los mecanismos de abstracción, según René Zuñiga [4], son definidos en representación, descomposición y ensamble, utilizados durante la resolución de problemas computacionales en el contexto de programación orientada a niños y adolescentes. Sin embargo, se advierte la necesidad de pen-

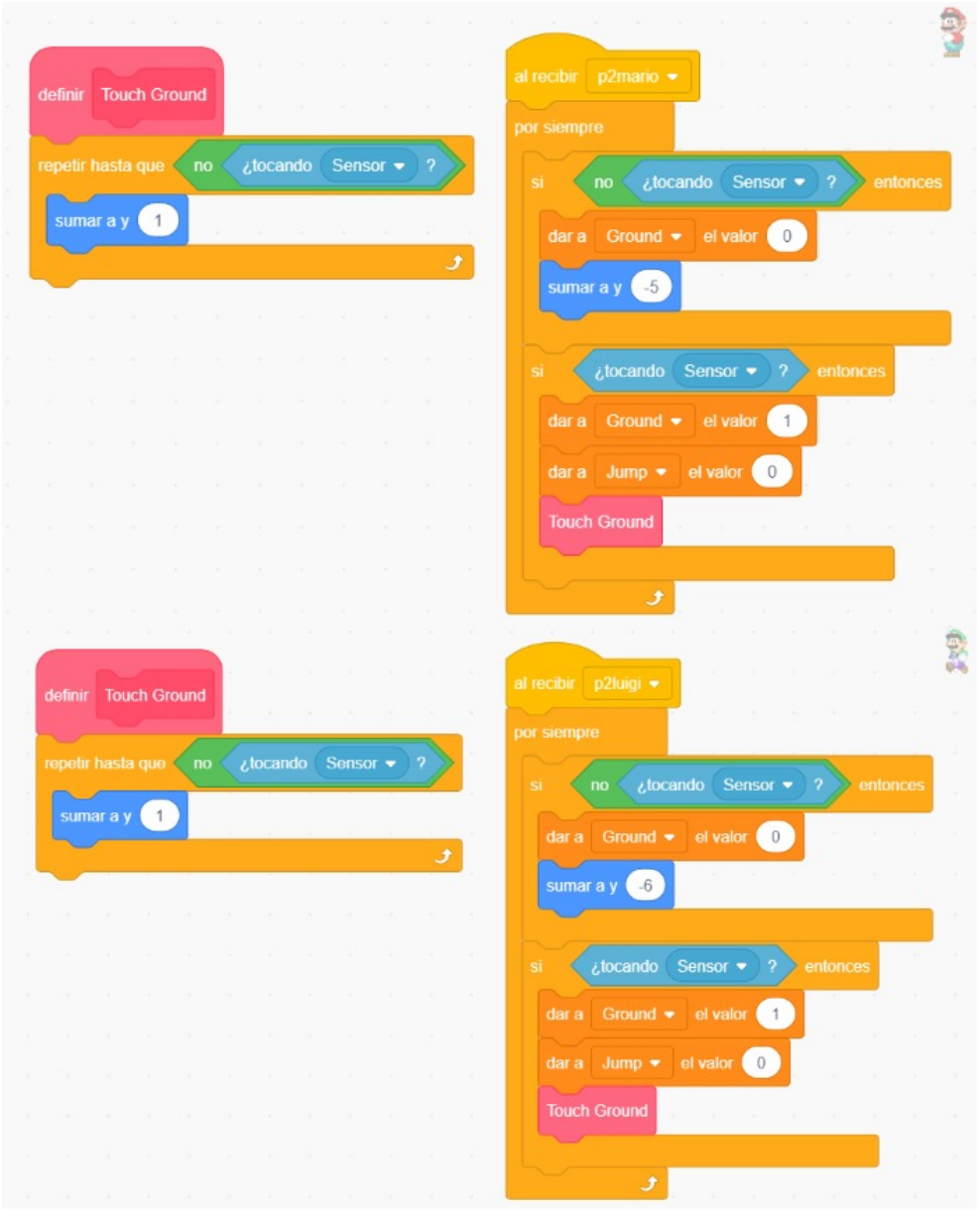

Figura 1.6: Duplicidad project-wide por limitaciones de Scratch

sar en dos tipos de soluciones como si se tratase de dos procesos de abstracción distintos, las centradas en el usuario y las centradas en las máquinas.

Cuando un programa es pequeño (p.ej. cientos de líneas de código), se puede obtener una solución basada en un único componente o función. Sin embargo, cuando el tamaño del problema aumenta el ser humano es incapaz de manejar tal cantidad de detalles y es necesaria una descomposición en pequeñas partes independientes. La evolución de las herramientas de programación ha ido acompañada de un creciente uso de este conjunto de conceptos, lo que

confirma a la abstracción como una aproximación efectiva para enfrentarse a problemas de gran complejidad [5].

La motivación principal de este proyecto nace de los conocimientos planteados en el artículo *Software Clones in Scratch Projects: On the Presence of Copy-and-Paste in Computational Thinking Learning* [6] del cual se obtienen como resultados los siguientes puntos:

- La clonación de software se puede encontrar comúnmente en proyectos de Scratch.
- La duplicidad se vuelve más frecuente a medida que los alumnos trabajan en proyectos que requieran habilidades avanzadas.
- No se encuentra ninguna dimensión del pensamiento computacional más relacionada con la ausencia de software clones que otros.
- Los alumnos, incluso si saben cómo evitar la clonación, todavía copian y pegan con frecuencia.

Los conocimientos y conclusiones del citado documento podrían ser utilizados por educadores y estudiantes para determinar cuándo es pedagógicamente más efectivo abordar la clonación de software, por desarrolladores de plataformas de programación educativa para adaptar sus sistemas y por herramientas de evaluación del aprendizaje para proporcionar mejores evaluaciones.

1.3. Estructura de la memoria

Este trabajo se divide de la siguiente manera:

1. **Introducción:** En este capítulo se hace una breve introducción al concepto del proyecto.
2. **Objetivos:** En este capítulo se describen los objetivos generales y específicos, así como la planificación temporal empleada.
3. **Estado del arte:** En este capítulo se describen las tecnologías que se implementan en el trabajo y se detallan conceptos para entender la estructura.
4. **Diseño e implementación:** En este capítulo se explica el funcionamiento del programa, desde la arquitectura que sigue hasta la explicación de las funciones y el hilo de ejecución.

5. **Experimentos, validaciones y resultados:** En este capítulo se profundiza en las pruebas realizadas, en los métodos de validación para el funcionamiento del código y en los resultados obtenidos.

6. **Conclusiones:** En este capítulo se muestran los objetivos conseguidos y se reflexiona sobre el futuro del proyecto.

7. **Apéndice:** En este capítulo se detalla el manual del usuario así como los requisitos para ejecutar el programa.

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo de este trabajo es crear una herramienta de análisis capaz de detectar duplicidad de código, tanto por objeto *(intra-sprite)* como en todo los objetos del proyecto *(project-wide)* en el lenguaje de programación Scratch.

El planteamiento inicial se enfoca en obtener datos para ficheros individuales pero en realidad el objetivo principal es aplicarlo a múltiples proyectos, por lo que es muy importante que el código a desarrollar sea escalable. En aspectos generales, el código ha de cumplir los siguientes pasos:

- Obtener la información del código fuente del lenguaje Scratch.
- Analizar y filtrar los datos para generar nuevos ficheros con la información relevante de cada bloque.
- Procesar la información mediante un modelo estadístico usando técnicas de clustering.
- Interpretar y mostrar los resultados obtenidos.

2.2. Objetivos específicos

Para alcanzar el objetivo principal se han perseguido los siguientes objetivos específicos:

- Estudiar la estructura visual y funcional del lenguaje de programación Scratch, especialmente el fichero con formato JSON[1] de cada proyecto.
- Diferenciar los bloques definidos por el programa de los definidos por el usuario.
- Contabilizar los bloques, sprites y scripts que tiene el proyecto.
- Contabilizar los bloques personalizados y las llamadas que se hacen a lo largo de la ejecución.
- Contabilizar los bloques ignorados según lo indique el usuario por línea de comandos.
- Obtener y representar el nivel de duplicidad de bloques.
- Incluir test unitarios para revisar la calidad del código y minimizar la aparición de errores no no controlados.
- Verificar y experimentar, en gran escala, con muchos códigos de Scratch.
- Definir patrones para la detección de duplicidad de código.

2.3. Planificación temporal

La planificación de este trabajo ha sido bastante atípica si tomamos en cuenta la situación excepcional vivida desde el año 2020, producto de la pandemia causada por la COVID-19. En Marzo del 2021 contacté con Gregorio; mi tutor, y me presentó la idea de un proyecto relacionado con Scratch, sin embargo, no es hasta el mes de Abril que se da inicio con el trabajo de investigación y con el desarrollo de código en Python. Durante los meses de Abril a Mayo nos reunimos de forma telemática una vez a la semana para conversar sobre el estado del proyecto. Se vuelve a retomar el trabajo el siguiente curso escolar 2.021/2.022, comenzando nuevamente desde el mes de septiembre.

A continuación, se puede visualizar en la figura 2.1 el itinerario de planificación y desarrollo que refleja las fases y duración del proyecto en un diagrama de Gantt.

[1] `https://en.scratch-wiki.info/wiki/Scratch_File_Format`

	2021				2022						
DURACIÓN / *FASES DEL PROYECTO*	SEPTIEMBRE	OCTUBRE	NOVIEMBRE	DICIEMBRE	ENERO	FEBRERO	MARZO	ABRIL	MAYO	JUNIO	JULIO
Planteamiento inicial											
Definición de objetivos											
Investigación y desarrollo de ideas											
Desarrollo y depuración del código											
Pruebas de entorno											
Desarrollo de memoria											
Modificaciones, revisión y maquetación final											

Figura 2.1: Diagrama de Gantt

Para facilitar la organización, estructuración y consecución de objetivos se hace uso de la herramienta Trello[2], allí se dividen las tareas en tres bloques: **Investigativo** (Amarillo), **Desarrollo de código** (Verde) y **Desarrollo de memoria** (Rojo). En las imágenes 2.2, 2.3 y 2.4 se pueden observar los distintos estados de los objetivos durante la elaboración del trabajo.

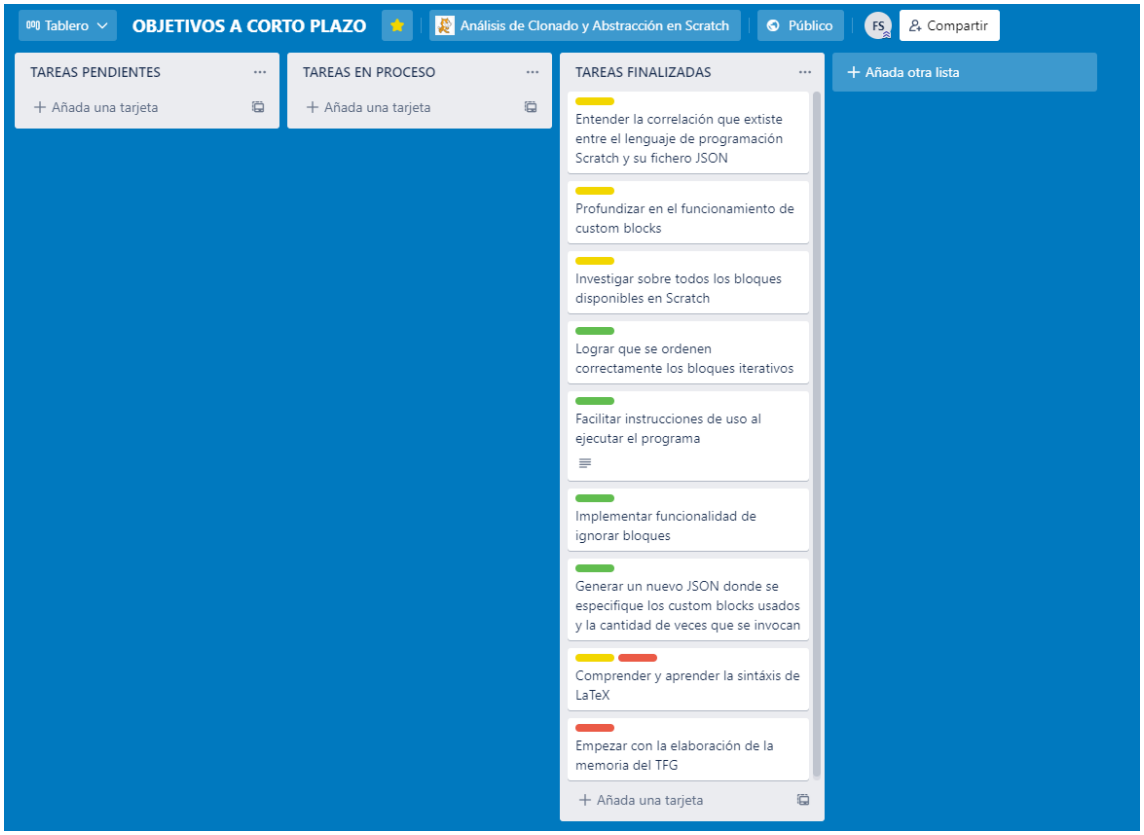

Figura 2.2: Tablero de Trello con objetivos a corto plazo

[2] `https://trello.com/tfg_felipesandoval/boards`

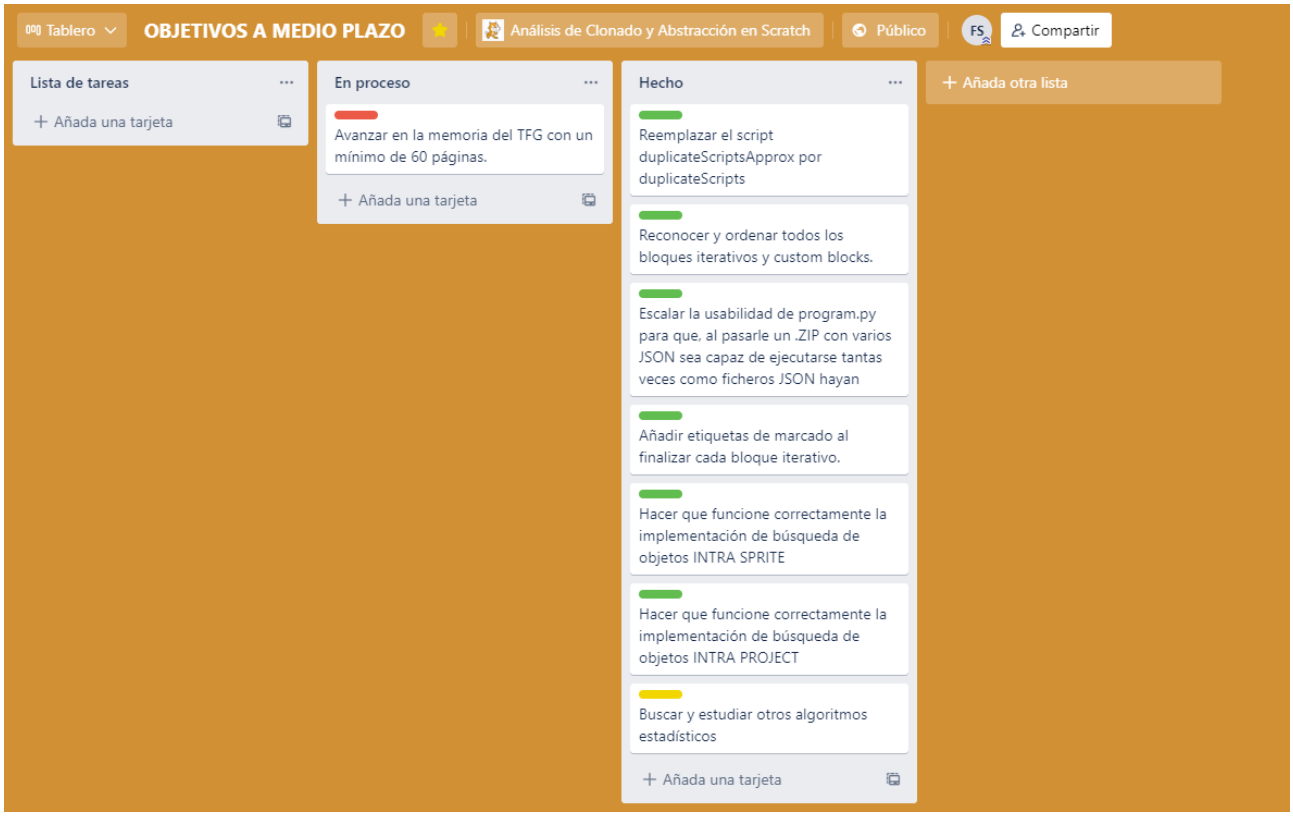

Figura 2.3: Tablero de Trello con objetivos a medio plazo

Figura 2.4: Tablero de Trello con objetivos a largo plazo

Capítulo 3

Estado del arte

En este capítulo se introducen las bases tecnológicas del trabajo.

3.1. Scratch

Scratch es un lenguaje de programación creado por el Grupo Lifelong Kindergarten del Laboratorio de Medios del MIT. En Scratch se permite programar historias interactivas, videojuegos, felicitaciones, animaciones, entre otros; todo esto a través de un entorno completamente visual que se fundamenta en el uso de personajes, escenarios y bloques gráficos que permiten la interacción entre ellos.

Entre sus cualidades desatacan que es totalmente gratuito, de uso libre, multilenguaje y altamente recomendado para la iniciación de niños y adolescentes en el mundo de la programación. Scratch es, además, una gran comunidad de usuarios de los que se puede aprender y compartir proyectos. Los mayores consumidores de Scratch en el mundo son Estados Unidos y Reino Unido, donde los niños que más lo usan son los que tienen una edad media entre los 13 y los 16 años.

Como se comentó previamente, en Scratch, los objetos se denominan *sprites* y la parte visual es construida por piezas de puzle llamadas *blocks* que al encajar entre sí forman funciones, conocidas como *scripts*. En la imagen 3.1 se puede apreciar la pantalla principal de la interfaz gráfica de Scratch. Existen varias categorías de bloques, siendo de especial interés las siguientes:

- Movimiento: Usados para mover y girar un objeto por la pantalla.

- Apariencia: Usados para cambiar la visualización de un objeto.
- Sonido: Usados para reproducir o detener secuencias de audio.
- Eventos: Usados para controlar eventos que ejecuten determinadas acciones.
- Datos: Usados para para crear y asignar variables.
- Control: Usados para crear bucles o condicionales de ejecución.
- Custom blocks: Bloques definidos por el usuario. A modo de comparativa, son funciones personalizadas que podrán ser invocadas a lo largo del código.

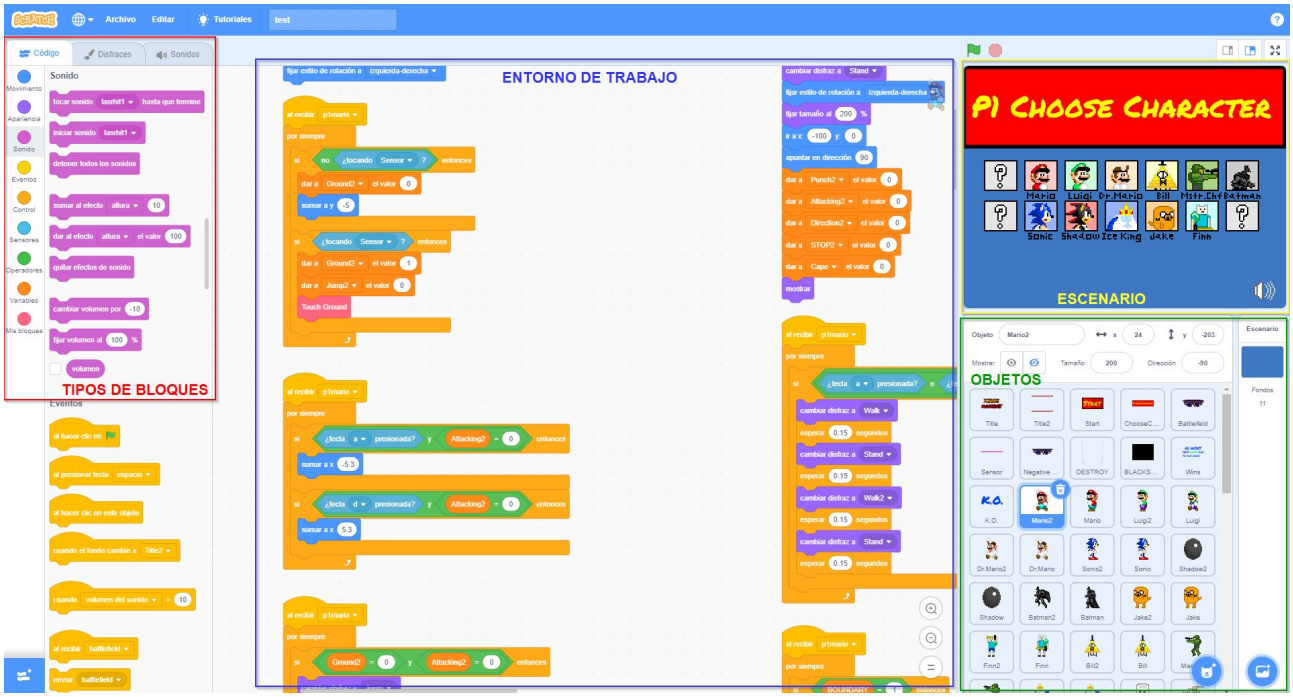

Figura 3.1: Interfaz gráfica de Scratch

La forma de programar en Scratch permite al usuario aprender rápidamente sin tener que comprender la sintaxis del lenguaje, centrándose en la lógica del programa, lo que mejora la habilidad de abstracción del usuario [7].

3.2. JSON

JavaScript Object Notation, mejor conocido como JSON, es un formato ligero de intercambio de datos que puede ser leído por cualquier persona. Este tipo de ficheros almacena infor-

mación estructurada que es utilizada principalmente para transferir datos entre un servidor y un cliente [8].

Teniendo en cuenta la sintaxis, existen dos elementos centrales en un fichero JSON *Keys y Values*. Las *Keys* o llaves contienen una secuencia de caracteres rodeados de comillas y los *Values* o claves contienen un tipo de datos JSON válido, igualmente entre comillas. Puede tener la forma de un array, un diccionario, un string, un boolean, un objeto, un número o de tipo nulo. A modo ilustrativo, en el siguiente texto se puede apreciar un ejemplo de la estructura del formato JSON:

```
{
  "datos":{
    "objeto1": {
      "opcode": "loop",
      "blocks": [
        "bloque1",
        "bloque2",
        "bloque3"
      ]
    },
    "objeto2": {
      "opcode": "custom",
      "blocks": [
        "customA",
        "customB"
      ]
    }
  }
}
```

Es importante mencionar la estructura de un fichero JSON ya que es donde estará el código fuente de mi proyecto en Scratch.

3.3. Python

Es un lenguaje de programación interpretado cuya filosofía enfatiza el tener una sintaxis sencilla favorecedora al código legible. Se trata así de un lenguaje de programación multiparadigma, ya que soporta la programación orientada a objetos, la programación imperativa y la programación funcional [9]. Python fue creado por Guido van Rossum a finales de los ochenta y su nombre se lo debe a los humoristas británicos *" Monty Python"*[1].

Existen tres versiones principales de Python pero las más extendidas son las versiones 2 y 3, siendo la versión 3.10 la última actualización disponible. En este proyecto se hace un uso íntegro de la versión Python 3.9. Cabe recalcar que las versiones 2 y 3 son incompatibles entre sí por el gran número de diferencias que existen entre ellas.

De todos los lenguajes de programación que aprendí a lo largo de la carrera he elegido Python por ser un lenguaje con una sintaxis sencilla, por ser multiplataforma, por su código abierto y, especialmente, por ser fuertemente tipado, ya que facilita la depuración de código y minimiza errores de compilación y ejecución.

Este proyecto se apoya en una serie de bibliotecas y módulos para Python. A continuación se enumeran y detallan el funcionamiento de cada uno:

3.3.1. Sys

Módulo que permite interactuar con parámetros y funciones específicas del sistema. Es usado para solicitar argumentos por consola y para detener la ejecución del programa mostrando un mensaje por pantalla. También es usado para registrar la información de los mensajes de error en el fichero de log de eventos.

3.3.2. Os

Módulo que permite trabajar, de forma portátil, con las funcionalidades del sistema operativo. En este proyecto se usa para eliminar ficheros JSON una vez han sido analizados, esto a fin de evitar crear excesivos documentos por cada ejecución.

[1] https://platzi.com/blog/historia-python/

3.3.3. Shutil

Módulo que ofrece varias operaciones de alto nivel en archivos y colecciones de archivos. Es usado para copiar ficheros dentro de la ruta donde se ejecute el script.

3.3.4. ZipFile

Módulo que permite el manejo de archivos con extensión de tipo ZIP. Se usa para abrir, descomprimir y extraer ficheros dentro de este tipo de contenedores.

3.3.5. JSON

Módulo que ofrece la posibilidad de manipular ficheros de tipo JSON. Es usado para *parsear*[2] los elementos del fichero JSON de cada proyecto Scratch.

3.3.6. Logging

Módulo que define funciones y clases para implementar un registro de eventos de log para aplicaciones y bibliotecas. Se usa para generar el fichero **program_logs.txt** que mostrará información relevante de la ejecución del programa. Es en este fichero donde se escribirán errores de ejecución y se detallará sobre la causa del fallo.

3.3.7. Traceback

Módulo que copia el comportamiento del intérprete de Python cuando muestra una traza de pila. Es usado para obtener información de fallos relevantes en el fichero de logging.

3.3.8. difflib

Módulo que implementa clases y funciones que ayudan a computar y trabajar con diferencias entre secuencias. Es especialmente útil para comparar texto e incluyen funciones que usan diversos algoritmos para lograr obtener esta información.

[2]Proceso de analizar una secuencia de símbolos a fin de determinar su estructura gramatical definida. También llamado análisis de sintaxis.

SequenceMatcher

Clase flexible que se usa para comparar pares de secuencias de cualquier tipo, siempre y cuando los elementos de la secuencia sean separables. Esta clase implementa un método heurístico que identifica automáticamente ciertos elementos como no deseados, para ello cuenta cuantas veces aparece cada elemento en la secuencia. Se usa para comparar secuencias en cadenas de caracteres [10].

3.3.9. collections

Este módulo implementa tipos de datos que proporcionan alternativas a los contenedores integrados de uso general de Python, como diccionarios, listas, sets, y tuplas [11].

counter

Es una subclase del tipo de datos *dict* para contar objetos separables. Los elementos se almacenan como llaves y sus conteos se almacenan como valores de diccionario. Se permite que los conteos sean cualquier valor entero, incluidos los conteos a cero o negativos.

defaultdict

Es una subclase del tipo de datos *dict* que anula un método y añade una variable de instancia editable a través de una función para suministrar valores faltantes. Funciona de forma parecida a los diccionarios clásicos de Python.

ordereddict

Es una subclase del tipo de datos *dict* que tiene métodos especializados para reorganizar el orden del diccionario ya que recuerda las entradas en el orden que se agregaron.

3.3.10. sklearn

Este módulo es usado para ejecutar funciones y clases de Machine Learning construidas sobre SciPy [12]. Cuenta con algoritmos de clasificación, regresión, clustering y reducción de

dimensionalidad. Además, es compatible con otras librerías de Python como NumPy y matplotlib.

cluster

Es una subclase que reúne diferentes algoritmos de agrupación no supervisada *(unsupervised clustering)*. Dichos algoritmos ejecutan la tarea de agrupar conjunto de objetos con datos similares en grupos semejantes entre sí. Juega un papel primordial en el análisis de datos exploratorios y ofrece técnicas infalibles para el reconocimiento de patrones.

Aunque existen muchos algoritmos en esta subclase los más utilizados son K-medias y el algoritmo de propagación de afinidad **Affinity Propagation**, esto debido a su alta escalabilidad y tolerancia al ruido. En la imagen 3.2 se pueden observar los clasificadores de distintos algoritmos incluidos en la subclase cluster. A efectos de este trabajo se describen los siguientes:

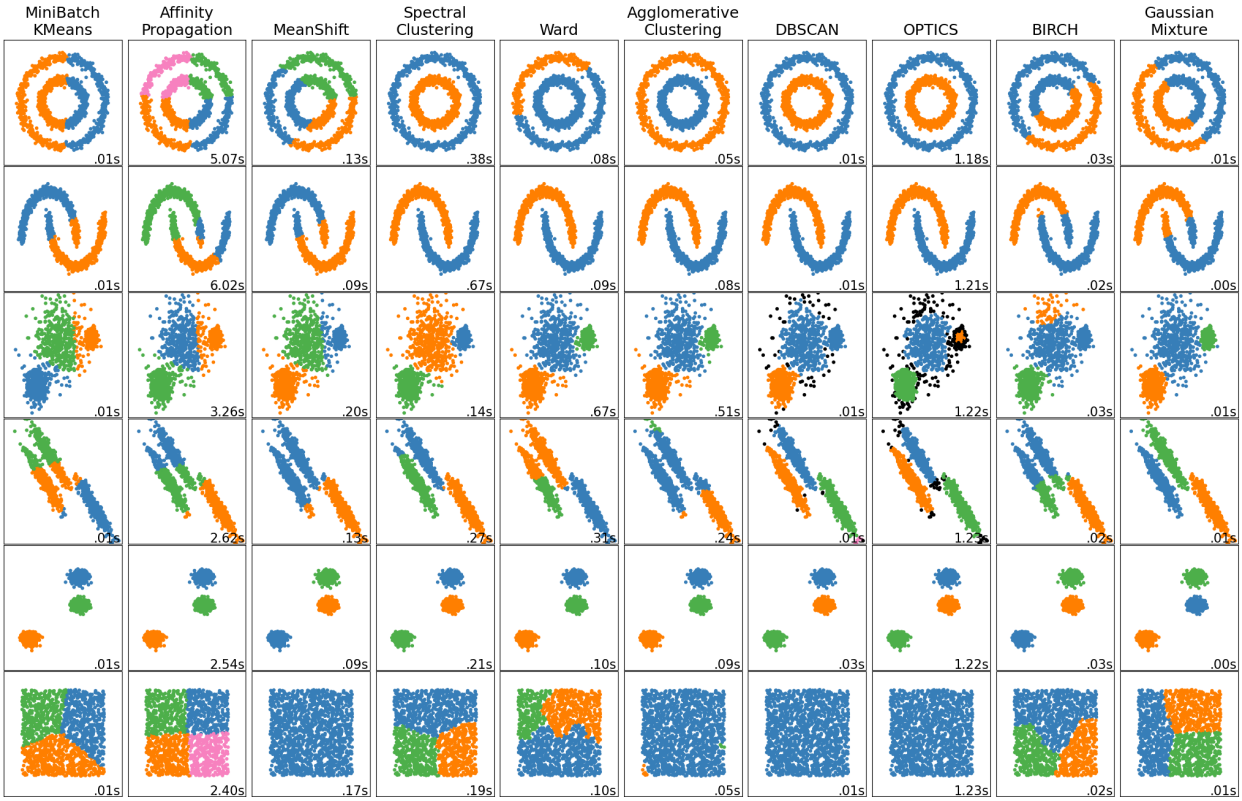

Figura 3.2: Comparativa de algoritmos de clustering

- *Algoritmo de propagación por afinidad (Affinity Propagation)*: Se usa para encontrar grupos de objetos que son similares entre sí. El algoritmo toma como input un conjunto de

objetos y un característica de similitud para devolver como output una lista de objetos que están agrupados por coincidencia. En la imagen 3.3 se puede apreciar un ejemplo del las iteraciones que realiza para clasificar el centro de los clústeres.

- *Algoritmo de K-medias (K-means)*: Se trata de un método de tipo divisivo, lo que significa que los datos se dividen en grupos más pequeños a medida que avanza el algoritmo. Su objetivo es asignar cada dato a un grupo de tal forma que los elementos dentro de un conjunto sean similares entre sí, mientras que los datos de otros grupos sean lo más diferentes posible.

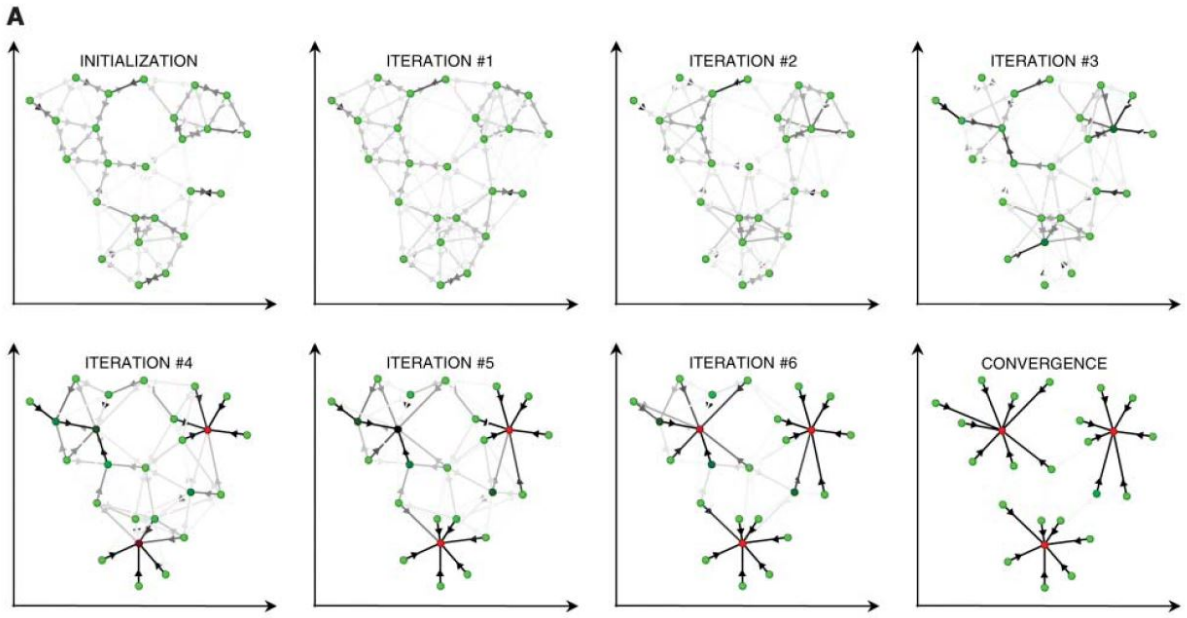

Figura 3.3: Funcionamiento ilustrativo del algoritmo de propagación por afinidad

En este trabajo se hará un uso exclusivo del algoritmo **Affinity Propagation** ya que, en teoría, es el más óptimo para la homogeneidad, integridad y similitud en conjuntos de cadenas de textos [13].

3.4. Hairball

Hairball en un plugin de Python que analiza proyectos de Scratch, ofreciendo a su salida información sobre el nivel de habilidades del pensamiento computacional, como abstracción,

paralelismo, lógica, sincronización, control de flujo, interactividad con el usuario y representación de los datos.

El modulo hairball permite generar una representación gráfica sobre los datos de un proyecto Scratch. Esto es útil para visualizar el flujo de datos y el funcionamiento del programa. También otorga información sobre malos hábitos de programación cómo código duplicado, código muerto y mal nombramiento e inicialización de variables [14].

3.5. Pip

Pip es un gestor de paquetes para el ecosistema de Python que es capaz de instalar y administrar paquetes mediante línea de comandos [15]. Pip es el gestor con el que se han instalado todos los módulos y bibliotecas que no trae por defecto Python. Pip suele estar incluido al instalar Python, sin embargo, podría no ser así por lo que sería necesario su descarga.

3.6. Unittest

Unittest es una herramienta empleada para realizar tests en Python, tanto para clases y módulos enteros como para scripts. Creada por Kent Beck, permite la automatización de tests y la opción de crear colecciones. Además, no está ligada a ningún marco de reportes por lo que nos permite utilizar el mas conveniente [16]. Para todo lo mencionado Unittest se sirve de tests fixtures, test cases, test suites y test runner, propios de la programación orientada a objetos, conceptos explicados a continuación.

- **Test fixture**: es la preparación del entorno de todas las características del test que queremos ejecutar.
- **Test case**: es la unidad individual del test, nos permite generar un test unitario para testear cada función.
- **Test suite**: es la colección de los test cases que se ejecutan y agrupan juntos, también puede existir una coleccion de test suites. Ejecutar una suite es lo mismo que ejecutar una serie de test cases individualmente.

- **Test runner**: es el componente encargado de ejecutar los tests y proporcionar un resultado. Puede disponer de una interfaz gráfica, texto o devolver un valor especial.

Para ejecutar el conjunto de test desarrollados en este proyecto es necesario ejecutar la siguiente orden en línea de comandos:

```
$ python3 -m unittest discover
```

3.7. Guía de estilo PEP8

PEP8 (Python Enhancement Proposal 8) es una guía de estilos definida para Python. Se trata de un conjunto de recomendaciones cuyo objetivo es ayudar a escribir código de forma estándar y legible [17]. Es la guía de estilo elegida porque se usó durante la elaboración de prácticas en la asignatura Protocolos de Transmisión de Audio y Vídeos por Internet (PTAVI). Algunas de las reglas más importantes de esta guía son:

- Indentación en cuatro espacios.
- Espacios antes de cada tabulación.
- Líneas limitadas a 79 caracteres.
- Saltos de línea con barra invertida.

Actualmente, la herramienta ha sido renombrada como **pycodestyle** y el comando de instalación necesario es el siguiente:

```
$ pip install pycodestyle
```

Capítulo 4

Diseño e implementación

En este capítulo se realiza una descripción detallada de la estructura del software desarrollado y la funcionalidad a nivel de código. También se indica el hilo de ejecución, el entorno de trabajo y las herramientas usadas para la consecución de este proyecto.

4.1. Entorno y herramientas de trabajo

Para poder obtener resultados confiables en múltiples entornos, se compila, ejecuta y depura el código en los sistemas operativos Windows e UNIX. Se hace uso de tres aplicaciones para mantener sincronizados ficheros, historial de versiones y tener un control sobre la consecución de objetivos:

- **Trello:** Aplicación usada para gestionar y organizar, a modo de tableros, las fases de las tareas a corto, medio y largo plazo.
- **Dropbox:** Herramienta empleada como servicio de almacenamiento en la nube para mantener sincronizados ficheros, imágenes y referencias de la memoria.
- **GitHub:** Entorno usado para alojar el código y mantener un control de versiones. Resulta muy útil para dar marcha atrás en caso de errores o para crear nuevos hilos con modificaciones en el código.

En la imagen 4.1 se puede apreciar un diagrama del entorno y la comunicación entre las herramientas usadas.

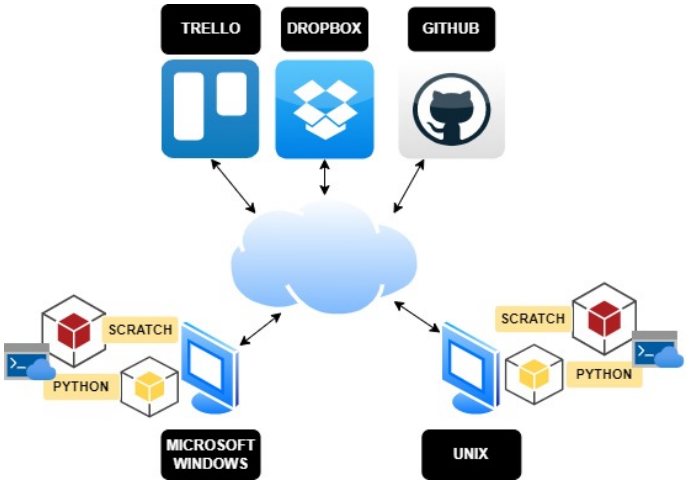

Figura 4.1: Diagrama de entorno y aplicaciones

4.2. Arquitectura general

A nivel general, el flujo del funcionamiento del programa es el que se puede apreciar en la imagen 4.2. El script a ejecutar es *program.py* y de allí se irán concatenando llamadas a *duplicatescripts.py*, *most_frequent_blocks.py*, *statistics.py* y paralelamente, *cluster.py*

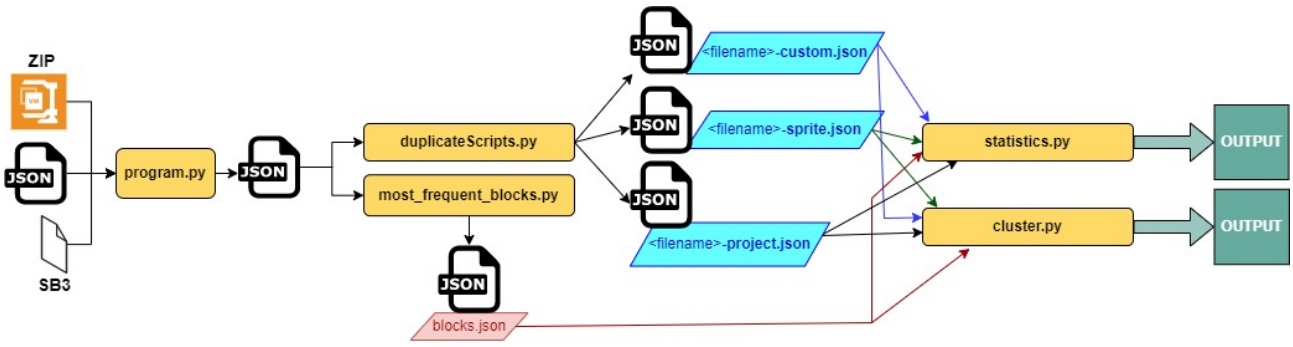

Figura 4.2: Hilo de ejecución del programa

Para ejecutar el software es necesario que el usuario pase por línea de comandos hasta un máximo de dos argumentos. Primero y obligatoriamente, el nombre del archivo a examinar, segundo y de forma opcional, el argumento **-i** que, de indicarlo, ignorará los bloques especificados en el fichero *IgnoreBlocks.txt*. En caso de pasar más argumentos, el programa mostrará un mensaje de error y creará un fichero de registros en el que escribirá los eventos que ocurran durante la ejecución.

Los ficheros a analizar pueden estar en formato SB3 (extensión que emplea Scratch) en

formato JSON o un fichero comprimido en formato ZIP. El software se ha desarrollado para ser escalable, por lo que, para realizar un análisis extensivo de muchos proyectos, es necesario comprimir los ficheros JSON en un contenedor ZIP.

A continuación, se detallan los scripts, funciones y clases relevantes del software.

4.2.1. program.py

Sirve como script principal. Su tarea es controlar el hilo de ejecución de todo el programa, para ello extrae y recorre el contenido de ficheros JSON. Esto es posible a través de las funciones **sb3_json_extraction** y **obtaining_json**. En este script se declara la forma como se escribirán los registros de control de ejecuciones en el archivo de logs, *program_logs.txt*. Este fichero resulta muy útil para depurar y encontrar excepciones durante la ejecución.

4.2.2. duplicateScripts.py

Este script toma como argumentos el nombre del fichero con su extensión correspondiente, el contenido del archivo JSON y el argumento opcional que indica si se ignoraran, o no, ciertos tipos de bloques.

El contenido del JSON se recorrerá como un de diccionario, empezando por la clave *targets* que contiene todos los objetos. Seguidamente se extrae la información relevante de la clave *blocks*, según el tipo de bloque que sea. En el código se irá almacenando en una variable de tipo diccionario, la clave/valor: *block_id:opcode*, la cual será importante para encontrar y añadir en su posición correcta a los bloques de control y *custom blocks*. En la imagen 4.3 se puede apreciar una vista de árbol generalizada sobre las clave/valor del fichero JSON, que según cada tipo de bloque contendrá más o menos información.

Por otra parte, en la imagen 4.4 se puede apreciar un diagrama con la estructura general de un bloque genérico. Las claves de color amarillo siempre están presentes por lo que son importantes para organizar la estructura de mis bloques. Las claves de color verde no son importantes para la organización aunque aparezcan en todos los tipos de bloque. Las claves de color azul no siempre están presentes pero son igualmente importantes. Por último, las claves de color rojo no aparecen siempre y tampoco son relevantes.

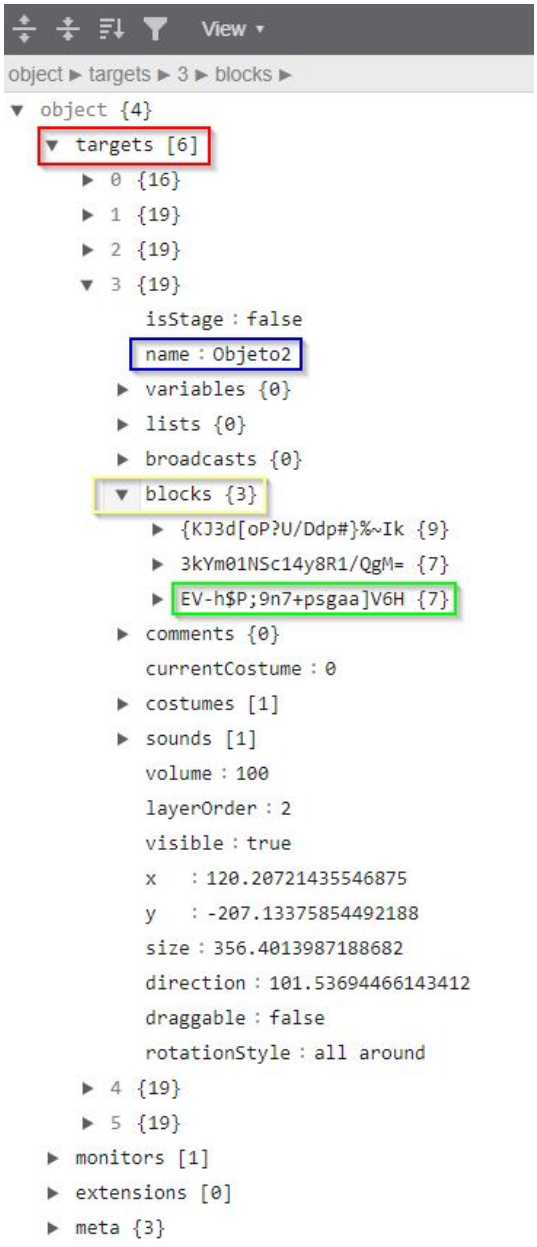

Figura 4.3: Vista de árbol del contenido de un fichero JSON

Como indicamos en el apartado anterior, todos los tipos de bloques suelen tener una estructura similar pero no todos sus campos son relevantes. Las claves como **opcode**, **parent**, **next** y **topLevel** son esenciales para la organización y extracción de datos. Otras claves, como **inputs** y **mutation** no aparecen en todos los tipos de bloques, sin embargo, son igualmente imprescindibles para obtener el orden correcto del conjunto de scripts. Es importante enfatizar que el *Block_ID* es un valor **único por bloque**, totalmente independiente del *opcode*. A continuación, se enumeran los casos genéricos y particulares considerados:

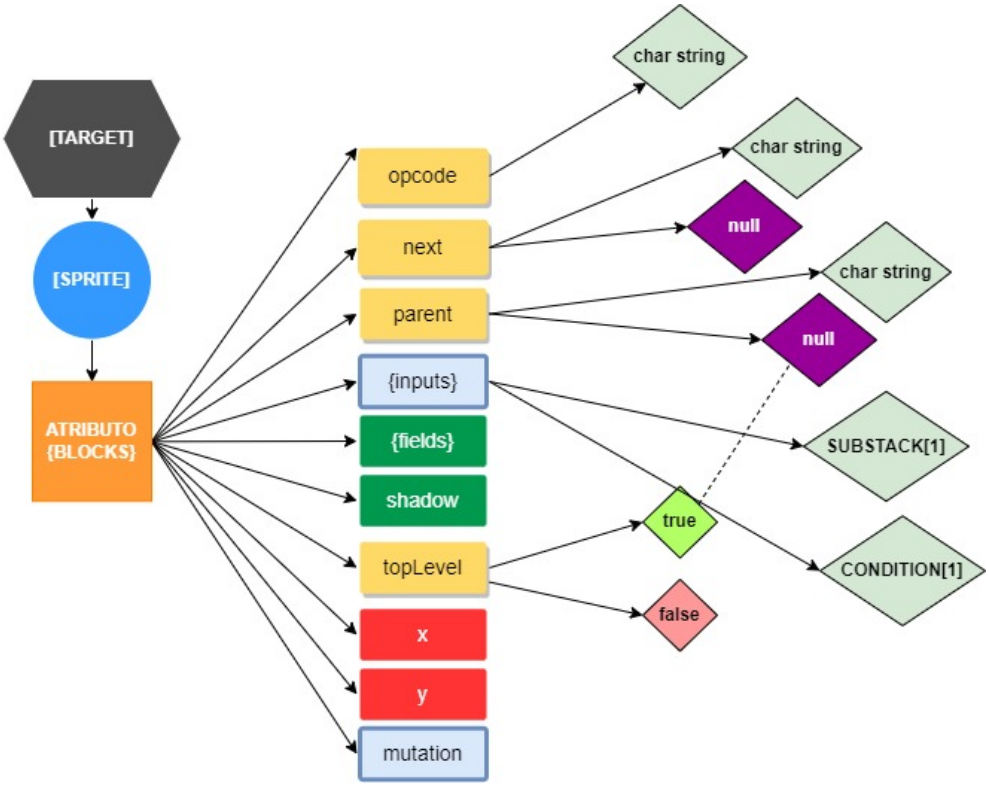

Figura 4.4: Estructura general de un bloque genérico

1. Si el valor de **topLevel** es True significa que el elemento es el primer bloque de mi script. En este caso el valor de la clave **parent** es null.

2. Si el valor de **topLevel** es False significa que el elemento no es el primer bloque. En este caso el valor de la clave **parent** contiene el *Block_ID* del bloque superior.

3. La clave de **opcode** siempre tendrá como valor un string con el nombre de ese bloque.

4. El valor de **next** puede ser un string con el *Block_ID* del siquiente bloque o puede ser null. En este último caso, puede ser porque es el último elemento del script, porque es un bloque de control *(control_repeat, control_forever)* o porque es un bloque definido por el usuario *(custom block)*. En la imagen 4.5 se puede apreciar un diagrama con las claves y valores relevantes para este tipo de bloques.

5. Dentro del valor **inputs** se encuentra otra clave llamada **substack** que contiene los elementos internos de los bloques de control (*control_repeat, control_forever, control_if, control_repeat_until y control_if_else)*. Este valor es exclúsivo de este tipo de bloques. En la

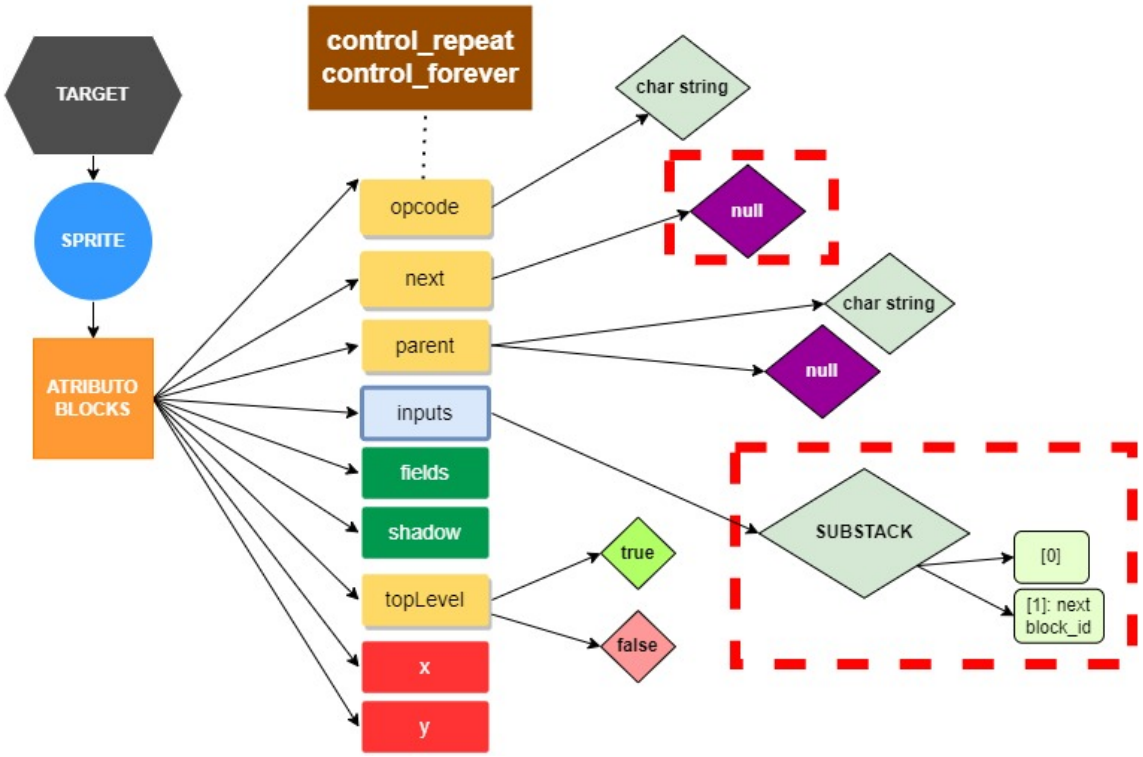

Figura 4.5: Estructura general de un bloque de tipo control_repeat o control_forever

imagen 4.6 se ejemplifica este caso y en la imagen 4.7 se muestra la interfaz de Scratch, a modo de puzle, sobre el script y sus referencias en el JSON.

6. Para el caso especial del bloque *control_if_else* el valor **substack2** contiene una lista con la secuencia de bloques internas en el hilo condicional. En la imagen 4.8 se ejemplifica la estructura especial de este tipo de bloques y en la imagen 4.9 se aprecia la interfaz de Scratch.

7. Se añaden las siguientes marcas de control al finalizar los siguientes bloques:

   - Marca "END_IF" al final de la primera condición de los bloques *control_if, control_repeat_until* y *control_if_else*.
   - Marca "END_ELSE" al final de la segunda condición del bloque *control_if_else*.
   - Marca "END_LOOP" al final de todos los bloques de control.

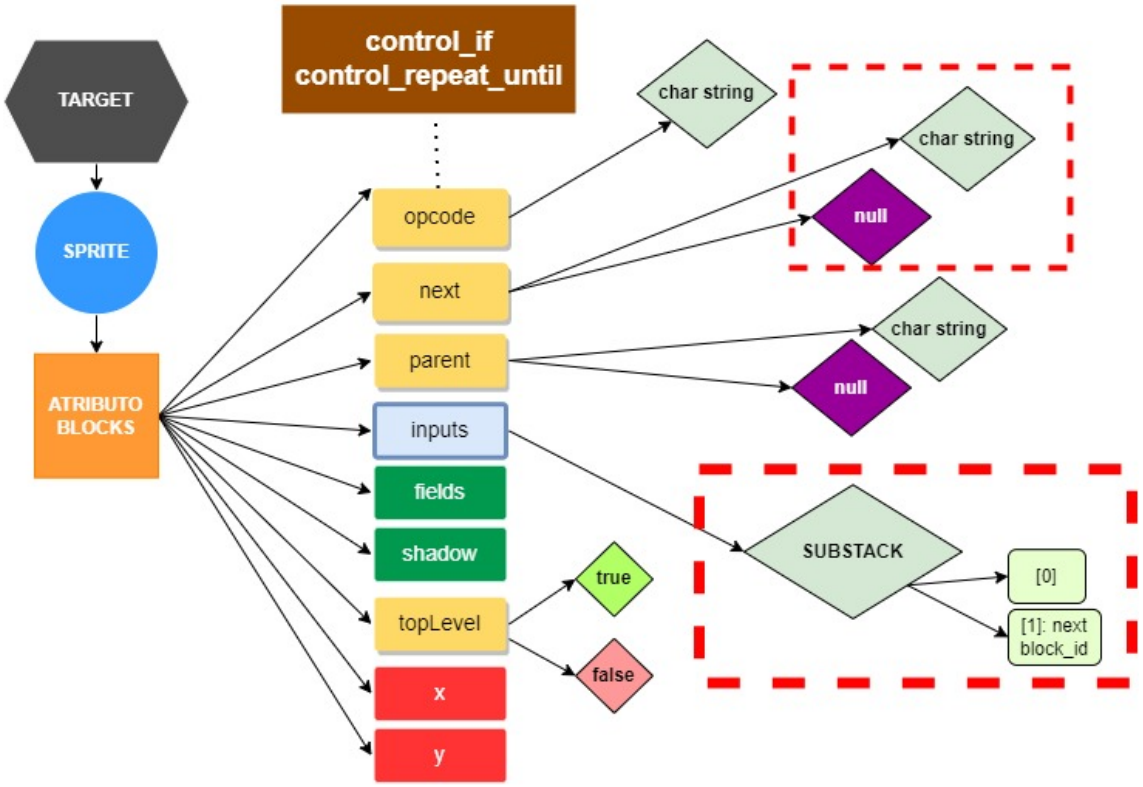

Figura 4.6: Estructura de un bloque condicional de tipo control_if o control_repeat_until

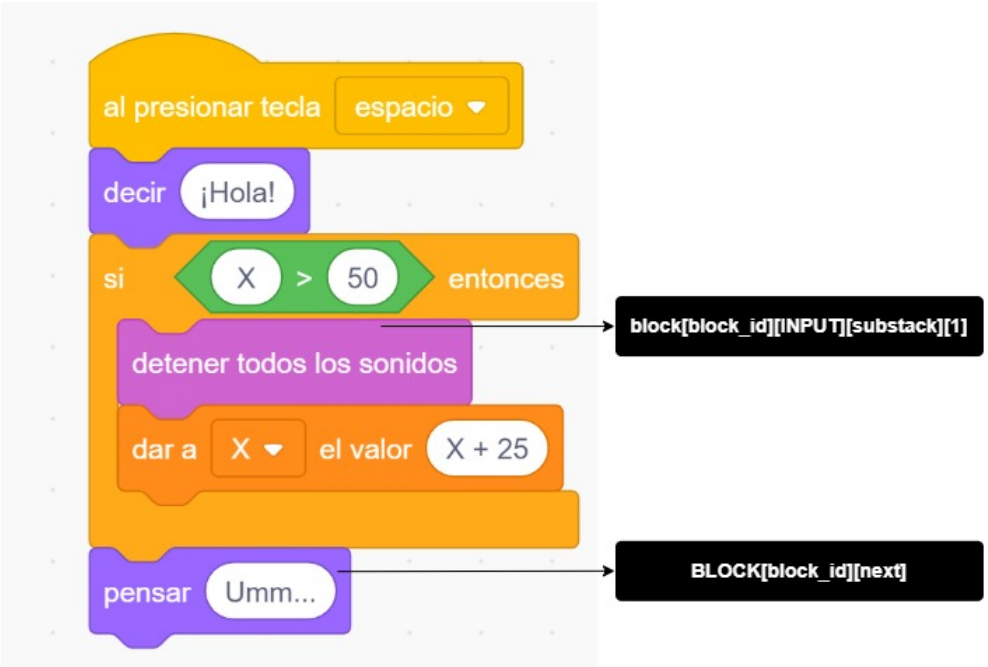

Figura 4.7: Script con bloque condicional de tipo control_if

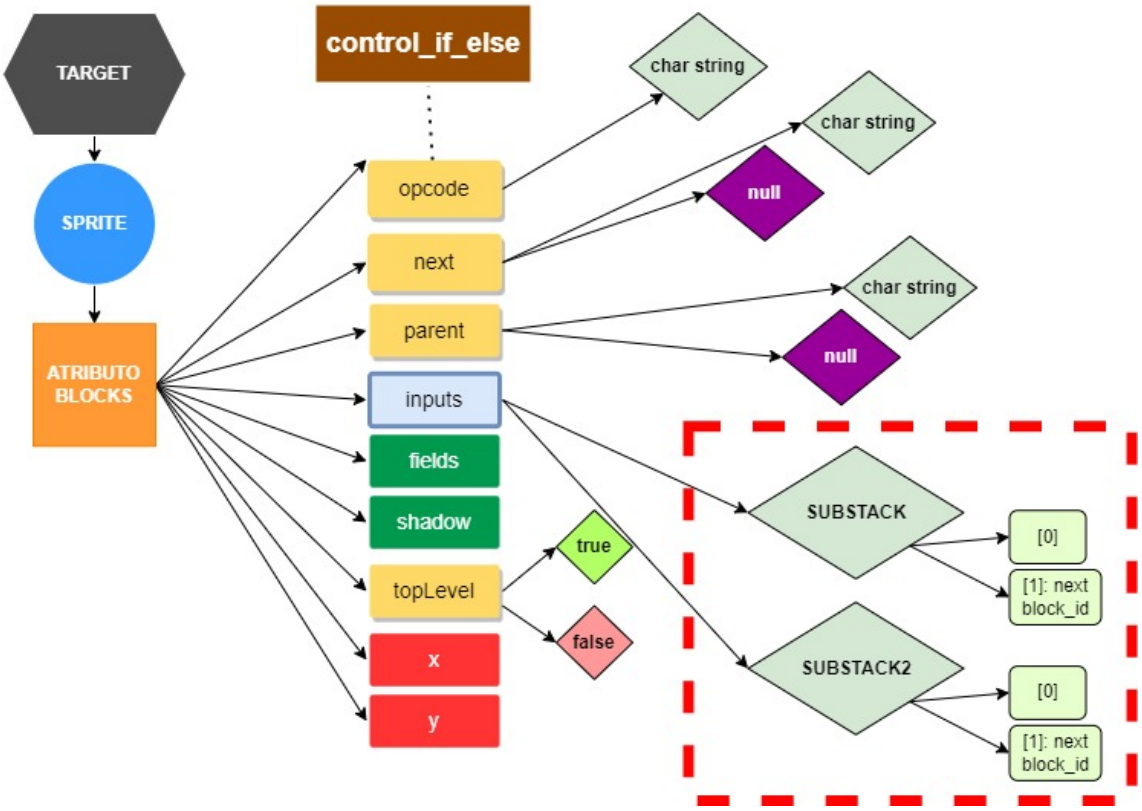

Figura 4.8: Estructura general de un bloque condicional de tipo control_if_else

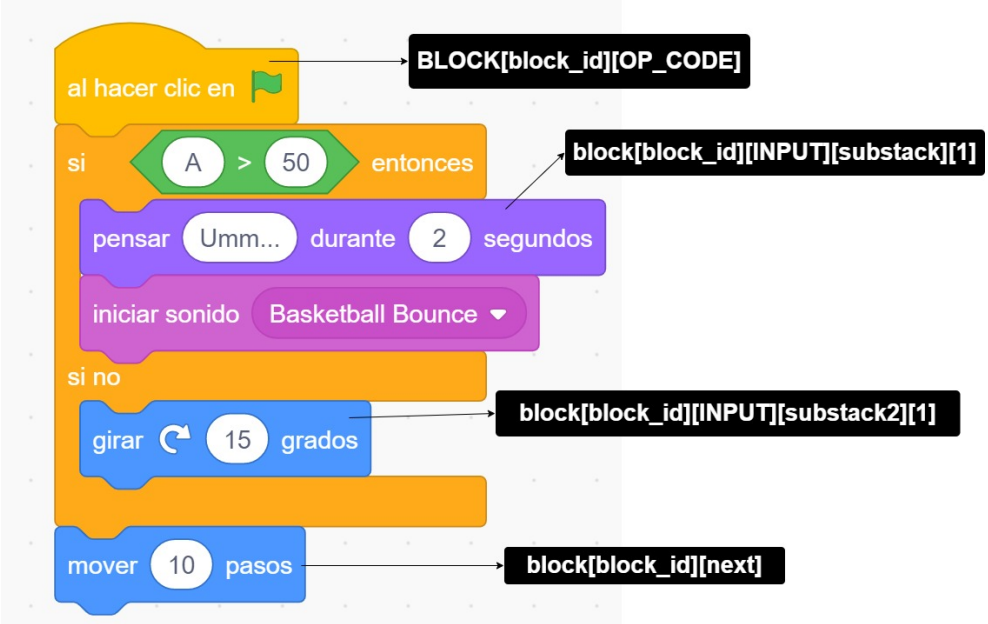

Figura 4.9: Script con bloque condicional de tipo control_if_else

Para los bloques personalizados por el usuario, los *custom blocks*, se tienen en cuenta los tres siguientes casos:

1. Para elementos con **opcode** cuyo valor sea *procedures_prototype*:
   - Su clave **next** tendrá siempre null como valor.
   - Su clave **parent** tiene como valor el block_ID de *procedures_definition*.
   - Su clave **topLevel** tendrá siempre false como valor.
   - En su clave **mutation** se encuentra tanto el nombre de la función, como los argumentos y sus respectivos valores.
2. Para elementos con **opcode** cuyo valor sea *procedures_definition*:
   - Su clave **parent** tendrá siempre null como valor.
   - Su clave **topLevel** tendrá siempre true como valor.
   - En su clave **inputs** se encuentra la sucesión de bloques que conforman mi custom block.
3. Para elementos con **opcode** cuyo valor sea *procedures_call*:
   - En su clave **mutation** se encuentra el nombre de la función lo que permitirá contabilizar el número de llamadas.

En las imágenes 4.10 y 4.11 podemos apreciar un ejemplo sobre la estructura de las clave:valor que contiene el JSON. Por último, se realiza una comparación en la imagen 4.12 sobre los distintos valores de cada bloque. Se aprecia que la información relevante para continuar con la organización secuencial está en *procedures_prototype* mientras que el contenido del custom block la tiene *procedures_definition*.

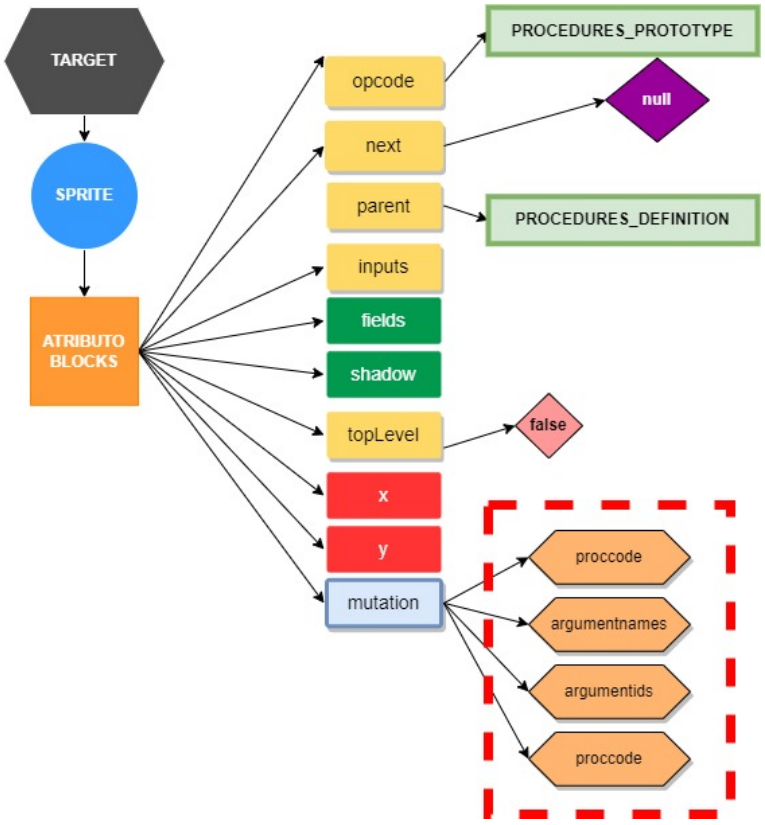

Figura 4.10: Estructura general de un bloque de tipo block_prototype

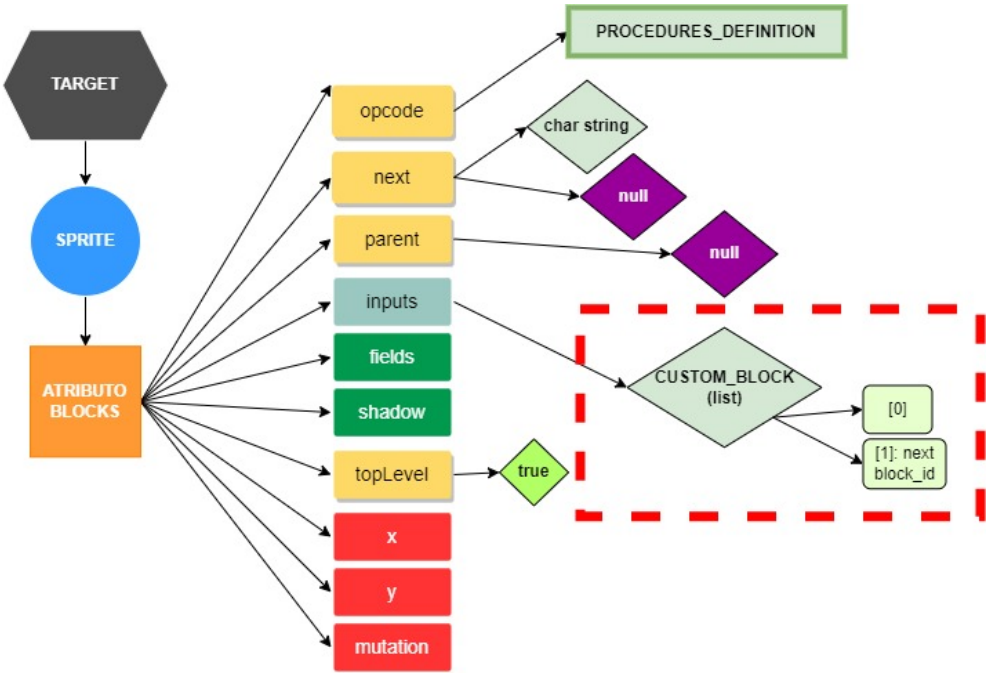

Figura 4.11: Estructura general de un bloque de tipo block_definition

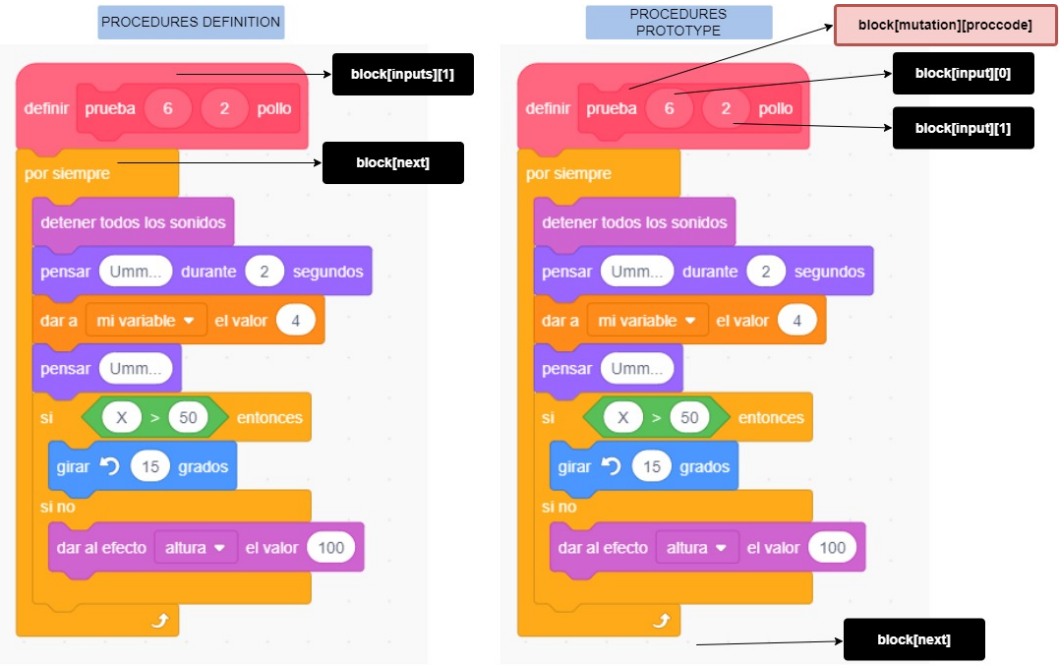

Figura 4.12: Diferencias entre tipos de bloques custom block

Uno de los principales problemas al momento de analizar el JSON ha sido ordenar de forma correcta la estructura de bloques, ya que por defecto no se muestran de forma ordenada. Para lograrlo se crean dos diccionarios: *scripts_dict* que almacena como valor los bloques de cada sprite y el diccionario *blocks_dict* que almacena como clave el block_id de cada bloque y como valor su opcode. Además, este último es esencial para insertar bloques según el block_id del parent que le corresponda.

Una vez se tiene todos los bloques de cada sprite ordenados se procede a obtener los bloques duplicados. Esto se logra gracias a la función **find_dups**, que toma como argumento una lista de bloques. Puede ser, exclusiva para los bloques de cada objeto (*intra-sprite*)o para todos los bloques del proyecto (*project-wide*). La función hace uso de la clase **sequenceMatcher** para comparar, palabra a palabra, cada bloque entre sí y devolver una lista con los bloque que más se repitan en cada iteración.

4.3. Obtención de datos

Para finalizar **duplicateScripts.py**, se ejecuta la función **finalize** que genera tres ficheros de tipo JSON. En la imagen 4.13 se puede apreciar un diagrama acerca esta última fase de

ejecución.

- "filename"-sprite.json: Contendrá las cadena de bloques mas repetidos por objeto.
- "filename"-project.json: Contendrá las cadena de bloques mas repetidos en todo el programa.
- "filename"-custom.json: Contendrá información relevante sobre los bloques personalizadoss presente en cada script, además del recuento de llamadas.

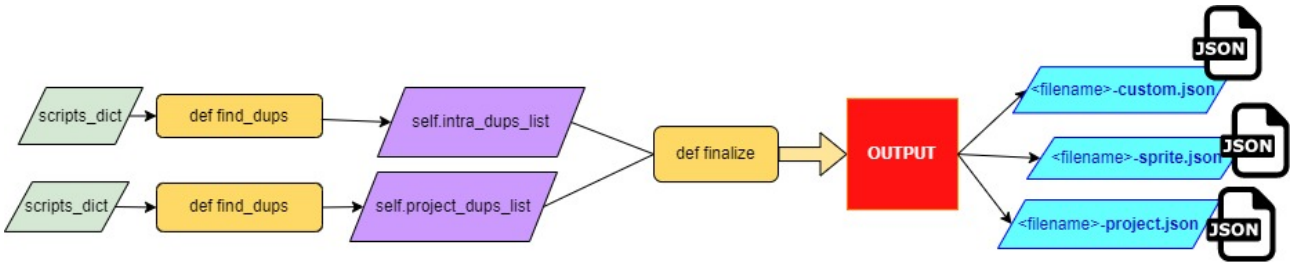

Figura 4.13: Diagrama de la última fase de ejecución duplicateScripts.py

4.3.1. most_frequent_blocks.py

Este script toma como argumento un fichero JSON e itera sobre sus elementos para buscar y contabilizar los *opcodes* de los bloques que más se repiten. Luego, ordena los bloques por frecuencia de uso y los guarda en un nuevo archivo JSON: **blocks.json** donde le asigna un caracter a cada bloque según su frecuencia de repetición. Al bloque mas repetido le corresponde la primera letra del abecedario y así sucesivamente completando la cadena de caracteres definida en la variable *characters*. Se puede apreciar el diagrama del funcionamiento del script en la imagen 4.14.

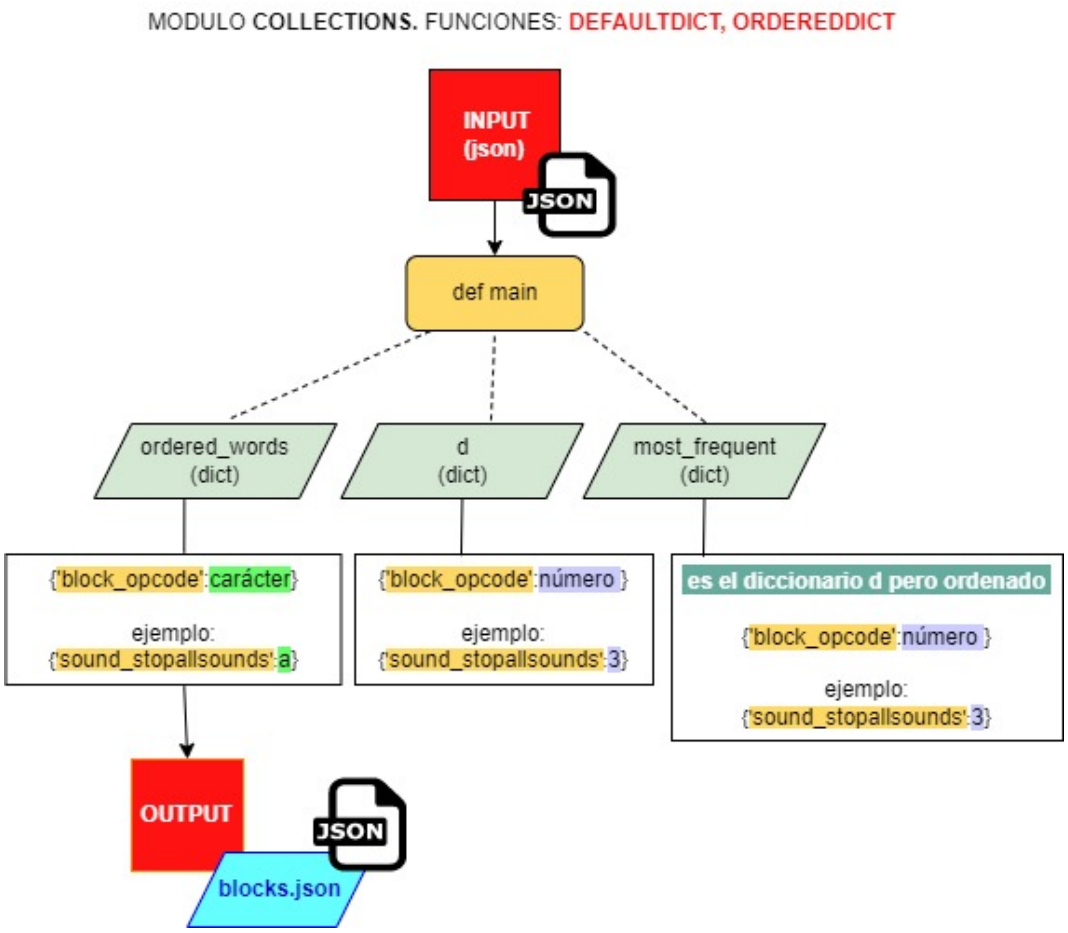

Figura 4.14: Diagrama de funcionamiento de most_frequent_blocks.py

4.4. Procesado de datos

4.4.1. statistics.py

Este script analiza los ficheros JSON generados por el código de **duplicateScripts.py** con ayuda de los módulos *counter*, *defaultdict* y *json*. Luego, la función **json2dna** carga el contenido del fichero **blocks.json** en el diccionario blocks_dict y, a continuación, define la variable *characters* que contiene todos los caracteres posibles. La función itera sobre cada elemento de la lista de duplicados y si el elemento no está en el diccionario del *blocks.json*, se le asigna un nuevo caracter a dicho bloque. Luego, se añade al diccionario de scripts, cada secuencia de bloques con los caracteres correspondientes asignados en blocks_dict.

Luego se añade el elemento al final de la lista block_list. Finalmente, la lista block_list se

convierte en una cadena y se añade a la lista scripts.

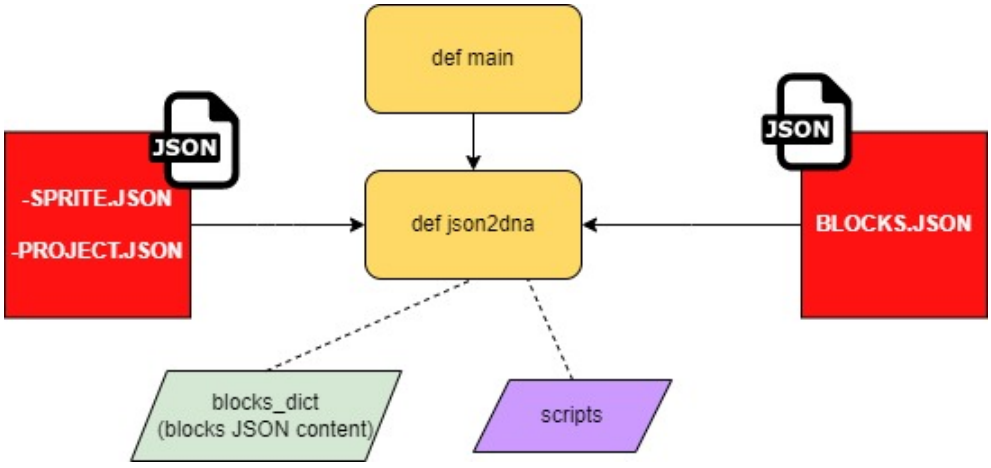

Figura 4.15: Diagrama de funcionamiento de statistics.py

4.5. Análisis de datos

4.5.1. cluster.py

El algoritmo que se emplea es el algoritmo de propagación por afinidad es un algoritmo de aprendizaje automático que se utiliza para encontrar patrones ocultos en datos no estructurados. Se basa en la idea de que los datos que se parecen a unos pocos puntos de datos son más probable que sean similares a otros puntos de datos. El algoritmo se ejecuta en iteraciones, en cada iteración, se seleccionan algunos puntos de datos y se les asignan etiquetas. Luego, el algoritmo busca otros puntos de datos que se parezcan a los puntos de datos etiquetados y les asigna las mismas etiquetas. El algoritmo continúa iterando hasta que todos los puntos de datos hayan sido etiquetados.

El código de este script importa la función de levenshtein de la biblioteca distance y la función Counter de la biblioteca collections. Luego, define una función llamada json2dna que toma como argumento un archivo JSON y devuelve los scripts como caracteres. En la función main, el código convierte los scripts en una matriz y luego usa la función de levenshtein para calcular la similitud entre los scripts. Finalmente, el código imprime los resultados del clustering.

4.6. Visualización de resultados

A continuación, un ejemplo genérico sobre como se visualizan los resultados por línea de comandos:

```
*** STARTING ANALYSIS  ***

/// IGNORING BLOCKS ACTIVATED ///
(se muestra solo si se pasa el argumento condicional -i)

-- STARTING DUPLICATESCRIPTS.PY SCRIPT --

Looking for duplicate blocks in (nombre de archivo)

Minimum number of blocks:  X

XX total blocks found
XX total blocks ignored
XX total scripts found
XX total sprites found
XX intra-sprite duplicate scripts found
XX project-wide duplicate scripts found
XX custom blocks found in all project
XX custom blocks calls found in all project

-- END OF DUPLICATESCRIPTS.PY SCRIPT --


-- GETTING INTRA SPRITE STATISTICS --

10 most common:
XX times
        BLOCK NAME
        BLOCK NAME
```

```
        ...

Different blocks: XX



-- GETTING INTRA PROJECT STATISTICS --

10 most common:
XXX times
        BLOCK NAME
        BLOCK NAME
        ...


Different blocks: XX



-- STARTING CLUSTER.PY SCRIPT --

(CLUSTERING INFORMATION)

-- END OF CLUSTER.PY SCRIPT --



*** ENDING ANALYSIS  ***
```

Capítulo 5

Experimentos, validaciones y resultados

La experimentación para este proyecto se ha desarrollado a modo de ensayo y error. Primero, para probar la efectividad del código se crean muchos programas en Scratch, que de manera controlada contienen bloques de código duplicado con múltiples combinaciones a nivel *intra-sprite* y *project-wide*. Se ejecuta el programa varias veces y se evidencian las limitaciones al detectar estructuras complejas, imperfectos que poco a poco se va corrigiendo, hasta que los resultados coinciden con la información gráfica diseñada en Scratch.

También se ejecuta la herramienta sobre un proyecto con multitud de objetos y bloques, el fichero **test.sb3** y se comprueba, al abrir el código en la interfaz de Scratch, que los resultados son verídicos. En la imagen 5.1 se puede apreciar un extracto de código del archivo JSON contenido en el fichero *test.sb3* que contiene 96 *sprites*, 11.077 *blocks*, 1.245 *scripts* y 24 *custom blocks*.

Luego se decide poner a prueba la herramienta a gran escala. Para lograrlo se repite la ejecución iterativa en 200 ficheros contenidos en el archivo *projects.zip*. En la imagen 5.2 se puede apreciar el contenido de dicho archivo.

Al terminar la primera ejecución se comprueban que hay ciertos JSON que no son posibles de analizar, estos quedan registrados en el fichero de errores **program_logs.txt**. Se analizan uno a uno los ficheros problemáticos y se descubre que existen casos particulares en los que ciertas claves, consideradas imprescindibles para la estructuración y organización, no aparecen en algunos bloques. Finalmente se depuran los errores de detección con la ayuda de las pruebas unitarias tratando cada caso de forma general.

Se confirma la robustez del software al obtener un resultado exitoso en todos los ficheros

Figura 5.1: Extracto del código del archivo JSON contenido en el fichero test.sb3

sin importar las múltiples combinaciones que pueda contener.

Para analizar la funcionalidad y robustez de mi código se crean los ficheros *test_program.py*, *test_DuplicateScripts.py* y *test_most_frequent_block.py* en los cuales se comprueba la salida de funciones y clases de manera controlada. Por ejemplo, en el fichero *test_DuplicateScripts.py* se crea un test case sobre la función **get_custominfo**. Cuando en los datos de entrada están incompletos, bien sea porque falta un campo o porque está mal escrito, se observa que la salida no es la deseada. Esto se puede apreciar con la clave "proccode" que es requerida en esta función. También se confirma que si la clave no aparece en el diccionario Python muestra una excepción de error de tipo **KeyError**. En la imagen 5.3 se puede apreciar un extracto del código con las pruebas descritas.

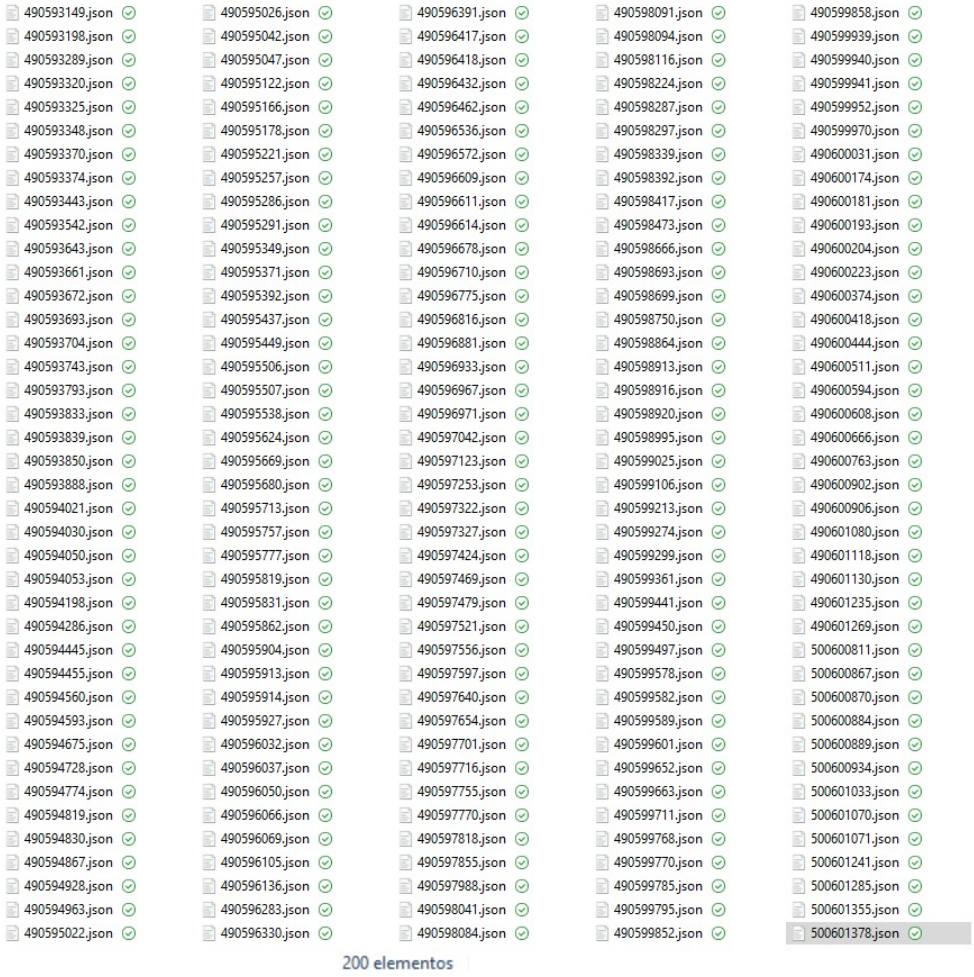

Figura 5.2: Contenido del fichero projects.zip

Para cuantificar el rendimiento es necesario realizar pruebas de velocidad y escalabilidad del software. Estas pruebas se hacen con el objetivo de obtener y comparar el tiempo de ejecución al ignorar, o no, ciertos tipos de bloques. Al momento de realizar las pruebas de rendimiento el resultado no siempre es exacto o fácil de interpretar, principalmente porque hay muchos factores que influyen en el valor que se obtiene. Por ejemplo, el rendimiento puede verse perjudicado por los procesos que el ordenador tenga en segundo plano, la velocidad de procesado del dispositivo, la capacidad de memoria en disco; entre otros.

A continuación, se realizan dos pruebas de rendimiento, la primera con el módulo **datetime** y la segunda con el módulo **Profile**.

```
def test_custominfo(self):
    self.assertEqual(get_custominfo({'parent': "0",
                                     'mutation': {'proccode': "A",
                                                  'argumentnames': "B"}}),
                     {"type": "procedures_prototype",
                      "custom_name": "A",
                      "argument_names": "B",
                      "n_calls": 0, "blocks": "0"})

    self.assertNotEqual(get_custominfo({'mutation': {'proccode': "A",
                                                     'argumentnames': "B"}
                                        }),
                        {"type": "procedures_prototype",
                         "custom_name": "A",
                         "argument_names": "B",
                         "n_calls": 0, "blocks": "0"})

    self.assertEqual(get_custominfo({'mutation': {'proccode': "A",
                                                  'argumentnames': "B"}
                                     }),
                     {"type": "procedures_prototype",
                      "custom_name": "A",
                      "argument_names": "B",
                      "n_calls": 0, "blocks": "empty"})

    self.assertEqual(get_custominfo({'mutation': {'proccode': "A",
                                                  'argumentnames': "B"}
                                     }),
                     {"type": "procedures_prototype",
                      "custom_name": "A",
                      "argument_names": "B",
                      "n_calls": 0, "blocks": "empty"})

    self.assertRaises(KeyError, get_custominfo,
                      {'any': {'proccode': "A", 'argumentnames': "B"}})

    self.assertRaises(KeyError, get_custominfo,
                      {'mutation': {'any': "A", 'argumentnames': "B"}})

    self.assertRaises(KeyError, get_custominfo,
                      {'mutation': {'proccode': "A", 'any': "B"}})
```

Figura 5.3: Código de un test case sobre una función de duplicateScripts.py

Pruebas de rendimiento con datetime

Este tipo de prueba de *"fuerza bruta"* consiste en restar el tiempo final de ejecución con el tiempo de inicio. En la tabla 5.1 se puede apreciar una comparativa con los resultados al ejecutar tres veces los ficheros *test.sb3* y *projects.zip*. En la columna izquierda de la tabla se observa el tiempo de ejecución al no ignorar bloques *(-i)* y en la columna derecha al hacerlo.

	Tiempo de ejecución	
Fichero	Sin ignorar bloques	Ignorando bloques
test.sb3	10,891 s	08,598 s
	12,404 s	10,096 s
	11,958 s	08,141 s
project.zip	12 m 56,030 s	09 m 53,710 s
	17 m 33,782 s	11 m 14,063 s
	15 m 21,103 s	10 m 49,338 s

Cuadro 5.1: Cuadro comparativo de los tiempos de ejecución con datetime

Pruebas de rendimiento con Profile

En esta prueba se hace uso del módulo **Profile** que proporciona un conjunto de estadísticas que describen la frecuencia y el tiempo en el que se ejecutan varias partes del código [18]. Para facilitar la lectura e interpretación se muestra el resultado de forma resumida en la tabla 5.2 con el número de llamadas no inducidas por recursividad y el tiempo que tarda en ejecutarse.

Fichero y argumento	Resultado
test.sb3 -i	**37.228.046** llamadas en 15,990 s
test.sb3	**57.851.803** llamadas en 23,745 s
project.zip -i	**2.983.280.176** llamadas en 1 h 1 m 33 s
project.zip	**4.734.914.750** llamadas en 1 h 56 m 12 s

Cuadro 5.2: Cuadro comparativo de los tiempos de ejecución con perfiladores de Python

Se concluye que el software presenta mayor rendimiento al ejecutarse con el comando opcional **-i**, por lo tanto, ignorando ciertos de bloques. Esto se confirma, según el resultado mostrado en ambas tablas, porque ejecuta un menor número de llamadas a funciones en el código. Se recomienda ejecutar el programa de esta forma ya que además de garantizar mayor velocidad de ejecución, optimiza la información mostrada respecto a la duplicidad.

Capítulo 6

Conclusiones

Cuando empecé a aprender a programar en Scratch, una de las cosas que me llamó la atención fue la facilidad con la que se podía duplicar código. Al principio, no entendía realmente la importancia de esto, pero a medida que seguía aprendiendo más sobre este lenguaje, empecé a darme cuenta del impacto que esto tiene en el desarrollo de la abstracción. La razón por la que el código duplicado es un problema en Scratch es porque es un lenguaje de programación visual. Esto significa que el código no se escribe en texto sino que se crea encajando bloques, lo que facilita su copia y pega, bien sea intencionada o por accidente.

La abstracción es el proceso de ocultar complejidad detrás de una interfaz más simple. En Python, esto se consigue normalmente creando funciones o clases. Al abstraer los detalles de una parte concreta del código, hacemos que sea más fácil de entender y reutilizar. En Scratch, es un poco más complicado ya que la propia interfaz nos limita la reutilización de los bloques personalizados. Por ejemplo, si creamos una función elemental en el objeto *"personaje"* solamente podremos hacer uso de ella dentro de dicho objeto obligando a crear nuevamente la función tantas veces como objetos se desee la implementen.

Por otro lado, el código duplicado es el que se repite varias veces en un programa. Esto puede ser un gran problema para la mantenibilidad porque si necesitamos cambiar el código, tendremos que cambiarlo en múltiples lugares u objetos en este caso. Esto puede conducir a errores si el código no se actualiza correctamente.

En la siguiente sección se debate sobre la consecución de los objetivos propuestos al inicio de este trabajo.

6.1. Consecución de objetivos

El objetivo principal era crear una herramienta que fuera capaz de detectar la duplicidad de código en Scratch. Se puede concluir y confirmar que se logra dicho objetivo ya que se desarrolla una aplicación interactiva, a través de línea de comandos, capaz de contabilizar bloques, sprites, scripts e identificar aquellos que se repitan, tanto a nivel *intra-sprite* como *project-wide*. También se reconocen los *custom blocks* y se contabilizan sus llamadas dentro del programa y, a la vez, se pueden omitir, según indicación del usuario, los bloques contenidos en el fichero *IgnoreBlocks.txt*.

Toda la información sobre la duplicidad de código es mostrada a través de línea de comandos, de una forma legible y simplificada para que, con el código visual de Scratch, el usuario pueda hacer las correcciones pertinentes en su código redundante.

Además de esto, se realizan pruebas con más de 200 archivos JSON contenidos en el fichero *projects.zip*, los cuales son analizados, sin errores, informando sobre los bloques que mayor duplicidad presentan cada uno de los proyectos.

No existe una conclusión clara sobre la definición de patrones para detectar duplicidad de código, por lo menos no a nivel de programación. Esto se debe a que las pruebas han de realizarse con muchos mas proyectos, sin embargo, se propone su desarrollo en las futuras mejoras de los próximos trabajos.

Por último se realizan pruebas unitarias del código. Esto garantiza que el funcionamiento y la ejecución de las funciones desarrolladas es el correcto según los valores que reciba.

6.2. Conocimientos aplicados

El aprendizaje y el conocimiento son fundamentales para el éxito en cualquier ámbito. A menudo, se asume que el conocimiento es una habilidad innata que se adquiere a través de la experiencia y el aprendizaje. Sin embargo, durante el transcurso del grado, me percate que para lograr el éxito se requiere de una combinación de capacitación, desarrollo del conocimiento y mucha práctica.

Quiero enumerar, por asignatura, las habilidades aprendidas que han facilitado la realización de este trabajo:

- **Informática I:** En esta asignatura tuve mi primer contacto con la programación. Aprendí las nociones básicas y aceleré el desarrollo de mi pensamiento computacional.
- **Estadística para Sistemas Audiovisuales:** En esta asignatura aprendí como representar e interpretar datos. Ha sido de gran ayuda para analizar las gráficas de los experimentos realizados.
- **Informática II:** En esta asignatura aprendí verdaderamente como programar. Aprendí un lenguaje complejo como Ada que me ayudó a organizar la legibilidad y eficiencia de mi código.
- **Protocolos para la Transmisión de Audio y Vídeo en Internet:** En esta asignatura comprendí realmente el alcance y las posibilidades del desarrollo de código. Fue mi primer contacto con Python, lo cual confirmó mi preferencia por las asignaturas en las que se tuviese que programar.
- **Tratamiento Digital del Sonido y Tratamiento Digital de la Imagen:** En estas asignaturas descubrí como funcionan los algoritmos de clustering y sobre el Machine Learning. Fue realmente interesante descubrir estas técnicas para entender como determinar patrones y manejar grupos de datos.
- **Construcción de Servicios y Aplicaciones Audiovisuales en Internet:** En esta asignatura aprendí sobre el desarrollo de aplicaciones web en lenguajes como HTML5, CSS, JavaScript, Node y Electron. Aunque en este proyecto no he tenido la oportunidad de desarrollar una interfaz web gráfica, me permitió reforzar mi interés en las asignaturas enfocadas en la programación, sin importar el lenguaje a usar.

6.3. Conocimientos adquiridos

El desarrollo de este trabajo ha supuesto un gran reto, pues es el primer proyecto que he realizado de esta complejidad. Durante las distintas etapas de elaboración he adquirido una serie de aprendizajes y habilidades que, a continuación, comparto:

- La importancia del desarrollo del pensamiento computacional. Al amar tanto la programación me he podido percatar que el pensamiento computacional es fundamental para el

manejo de la información, la solución de problemas y la comprensión del comportamiento humano y debe desarrollarse preferentemente desde edades tempranas. Su desarrollo mejorará la competitividad e innovación, a la vez que forma nuevas aptitudes y valores en los niños [19].

- La existencia de las pruebas unitarias. De las lecciones que más me han marcado durante la realización de este proyecto, ya que su uso e implementación pasaran a ser fundamentales al momento de desarrollar software.

- El lenguaje de programación Scratch. Desconocía totalmente la existencia de este lenguaje y su aplicación gráfica. De haberlo conocido seguro que hubiera desarrollado mi pasión por la programación a una edad más temprana. Considero alta su importancia para dar ese primer contacto con el mundo digital y para pensar de forma lógica a la vez que se resuelven problemas de forma creativa.

- El uso de ficheros JSON y de archivos de log. He podido apreciar lo sumamente importante que es escribir y descifrar la información contenida en archivos JSON para entender la estructura de cualquier aplicación. También es importante la depuración de código a través de ficheros de control para así controlar y organizar los problemas que van surgiendo.

- Algoritmos de clustering en Python. He tenido la oportunidad de leer y aprender sobre diversos algoritmos, comparándolos entre sí y ampliando mi conocimiento sobre el alcances y utilidad que puede tener el aprendizaje máquina.

- La facilidad de LaTeX. Aprender su estructura y semántica ha sido de gran utilidad para la maquetación de este trabajo.

6.4. Trabajos futuros

El software tiene un amplio rango de mejoras tanto a nivel de código como de funcionalidades. Se sugieren las siguiente implementaciones para los próximos trabajos:

- Brindar información al usuario donde se le indique otras alternativas para reducir la duplicidad de bloques, sin alterar la lógica del funcionamiento.

- Mostrar gráficas, a modo informe, donde se muestren los bloques más duplicados basado en un estudio de múltiples proyectos.
- Crear una interfaz gráfica para facilitar la interacción y uso. Se podría desarrollar una aplicación web interactiva alojada en GitLab.[1]
- Extender la funcionalidad de reconocimiento de bloques duplicados para reconocer la duplicidad de los *custom blocks*, tanto a nivel *intra-sprite* como *project-wide*.
- Integrar la herramienta en la aplicación de Dr. Scratch. Esto permitiría cuantificar a través de una puntuación la duplicidad de código presente.
- Ampliar las conclusiones haciendo uso de una base de datos con infinidad de proyectos de Scratch.
- Ampliar pruebas con otros algoritmos de clustering. Se podría realizar un trabajo investigativo donde se compare la efectividad entre algoritmos de aprendizaje supervizado y de aprendizaje no supervisado. Esto serviría para confirmar si efectivamente el algoritmo de propagación por afinidad es el más óptimo.

[1] https://about.gitlab.com/blog/2016/04/07/gitlab-pages-setup/

Bibliografía

[1] "Web sobre bloques personalizados."
https://en.scratch-wiki.info/wiki/My_Blocks.

[2] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007.

[3] I. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, pp. 368–377, 1998.

[4] R. F. Z. Muñoz, I. Grupo, J. A. H. Alegría, and C. A. C. Ordoñez, "Comprendiendo los procesos de abstracciónn computacional en los niños: un estudio de caso exploratorio," *I+ T+ C-Revista Investigaciín, Tecnología y Ciencia*, vol. 1, no. 8, pp. 57–66, 2014.

[5] A. Garrido, "Abstracción en programación," *Ciencia*, 2016.

[6] G. Robles, J. Moreno-León, E. Aivaloglou, and F. Hermans, "Software clones in scratch projects: On the presence of copy-and-paste in computational thinking learning," in *2017 IEEE 11th International Workshop on Software Clones (IWSC)*, pp. 1–7, IEEE, 2017.

[7] C. Arotuma and S. Soraya, "La programación como herramienta educativa-scratch," 2017.

[8] "Web sobre json."
https://www.json.org/json-en.html.

[9] "Web sobre python."
https://wiki.python.org/moin/BeginnersGuide/Programmers.

[10] "Web sobre el funcionamiento de sequence matcher."
https://towardsdatascience.com/sequencematcher-in-python-6b1e6f3915fc.

[11] "Web sobre los tipos de datos de collections."
https://docs.python.org/3/library/collections.html.

[12] "Web sobre sklearn."
https://scikit-learn.org/stable/modules/classes.html.

[13] M. F. Areed, "Python-based graphical user interface for automatic selection of data clustering algorithm," *International Journal of Future Generation Communication and Networking*, vol. 13, no. 3, pp. 451–461, 2020.

[14] B. Boe, C. Hill, M. Len, G. Dreschler, P. Conrad, and D. Franklin, "Hairball: Lint-inspired static analysis of scratch projects," in *Proceeding of the 44th ACM technical symposium on Computer science education*, pp. 215–220, 2013.

[15] "Web sobre pip."
https://pypi.org/project/pip/.

[16] "Web sobre unittest."
https://docs.python.org/3/library/unittest.html.

[17] "Web sobre pep8."
https://pep8.org/.

[18] "Web sobre los perfiladores de python."
https://docs.python.org/3/library/profile.html.

[19] J. M. Wing, "Computational thinking benefits society — social issues in computing)," tech. rep., International Institute of Infonomics. University of Maastricht, New York, May 2014.
http://socialissues.cs.toronto.edu/2014/01/computational-thinking/.

Apéndice A

Manual de uso

A.1. Descarga de ficheros

El código del trabajo desarrollado se puede encontrar alojado en el siguiente repositorio de GitHub: **https://github.com/felipsandoval/duplicatescripts-scratch**

A.2. Requisitos

Para poder hacer uso del programa es necesario tener instalado una serie de módulos y bibliotecas de Python. Para instalar todas estas dependencias es necesario ejecutar el siguiente comando:

```
$ pip install -r requirements.txt
```

A.3. Unittest

Para ejecutar el conjunto de test y verificar que las funciones, módulos y clases funcionan correctamente, es necesario ejecutar la siguiente orden por línea de comandos:

```
$ python3 -m unittest discover
```

A.4. Uso

```
$ python3  program.py  <fichero(.SB3 o .JSON o .ZIP)>  [-i]
```

-i *(OPCIONAL)*: Ignora los valores de bloques especificados en IgnoreBlocks.txt. También se ignoran las marcas de control: **END_LOOP**, **END_IF** y **END_ELSE_IF**.