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, más 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 mamá, 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 aguantarme en los momentos difíciles, por ayudarme a desconectar y por nunca dejar que tirase la toalla. Me demostraste 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 comprenderme 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, aprender y aprobar una asignatura 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 pa-

ciencia, 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, estoy seguro, 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, estudiar y analizar la duplicidad de código en proyectos de Scratch. Durante el proceso, se analizan los distintos llamados *bloques* que conforman cada proyecto y se obtiene como resultado una lista con aquellos que mayor duplicidad presenten. Para lograr este objetivo se realiza un estudio estadístico tanto a nivel **intra-sprite** como a nivel **intra-project** haciendo uso de algoritmos de clustering.

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 esta última habilidad la que motiva la ejecución de este trabajo.

La herramienta se desarrolla usando Python y sus diversas librerías como principal tecnología. Se extraerá la información necesaria del fichero con extensión JSON de un proyecto de Scratch para, posteriormente, obtener los datos deseados para el análisis.

Summary

This work aims to develop a tool capable of extracting, studying, and analyzing code duplicity in Scratch projects. The different blocks that make up each project are analyzed to obtain a list of the most duplicated ones. To achieve this objective, a statistical study is carried out at both **intra-sprite** and **intra-project** level, using clustering algorithms.

The project idea takes origin from other tools that evaluate aspects such as computational thinking, which encompasses skills like logical reasoning, synchronization, parallelism, information representation, and abstraction. It is this last skill that motivates the execution of this work.

The tool is developed using Python and its various libraries as the main technology. The necessary information will be extracted from the JSON extension file of a Scratch project to subsequently obtain the desired data for the analysis.

Índice general

Índice de figuras

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 **intra-proyect**, siendo los baremos de penalización distintos según cada caso. Esto como consecuencia de los límites de la propia aplicación que restringe la re-utilización de código de forma eficiente entre objetos.

Antes que nada, es muy importante entender a *grosso modo* el 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. Mas adelante se enumerarán los tipos de bloques.

- Un objeto o *sprite* son los objetos de mi proyecto. A nivel visual, un sprite puede ser el fondo de tu escenario o una imagen.

- Una función o *script* es una colección de bloques. Su orden es muy importantes, ya que determinan cómo los sprites interactúan entre sí, con el escenario y con el usuario, en caso de ser necesario. A nivel visual es una pila de varias piezas de puzle.

- Un bloque personalizado o *custom block* son las funciones o *scripts* definidos por el usuario.

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

En este trabajo se busca ponderar la relación existente entre el desarrollo de la abstracción con la información volcada en los bloques duplicados en el código de Scratch. El análisis de dicho código se realiza por el uso de algoritmos de agrupación mediante el intercambio de mensajes entre puntos de datos. [1]

1.1. Contexto Personal

Como alumno del Grado de Ingeniería en Sistemas Audiovisuales y Multimedia (ISAM) he adquirido distintas competencias a lo largo de los años de carrera. Las asignaturas de programación son las que mayor interés suscitaron en mí, así que después de unos años en los que, por motivos profesionales, me alejé de este campo, decidí volver a intentar adquirir conocimientos desarrollando software libre enfocado en la creación de aplicaciones con carácter pedagógico.

1.2. Motivación

Según se ha podido comprobar en la literatura científica [2] a mayor existencia de duplicidad de código menos se usa la abstracción, lo que ocasiona una mala síntesis en la descomposición de problemas. Los proyectos de Scratch hacen uso extensivo del clonado (i.e. copia y pega) lo cuál puede suponer una limitación a la hora de desarrollar distintas habilidades del pensamiento computacional.

Entendemos la abstracción como un proceso científico basado en dos fundamentos:

- Definir un modelo que permita pensar y descomponer un problema fundamental en problemas mas sencillos.
- Facilitar la proyección de técnicas que solucionen dichos problemas.

Los mecanismos de abstracción, según René Zuñiga, [3] 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 pensar 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. [4]

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 han implementado en el trabajo y se detallan conceptos para entender su estructura.
4. **Diseño e implementación:** En este capítulo se visualiza el funcionamiento del programa, desde la arquitectura que se sigue hasta el funcionamiento de las funciones y el hilo de ejecución completo.
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 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 comos 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 por proyecto *(intra-project)* en el lenguaje de programación Scratch.

El planteamiento inicial se enfoca en obtener datos para proyectos individuales pero, en realidad, el objetivo principal es aplicarlo a múltiples proyectos, por lo que es muy importante la redacción de código escalable. En grandes razgos, el código ha necesario seguir 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 Scratch.

- Diferenciar los bloques definidos por el programa de los definidos por el usuario. [2]

- Hacer uso de varios algoritmos de agrupamiento para determinar cuál es el mas eficiente en comparación con otros.

- Obtener y representar el nivel de duplicidad de bloques presentes en el código.

- Incluir test unitarios en las prácticas para controlar la calidad del código.

- Verificar y experimentar, en gran escala, con muchos códigos de Scratch.

- Definir patrones en 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 presenta una idea de proyecto relacionada con Scratch, sin embargo, no es hasta el mes de Abril que se da inicio con el trabajo investigativo 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 proyecto en Abril del 2022.

A continuación, se puede visualizar en la figura 2.1 el cronograma de elaboración en un diagrama de Gantt.

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

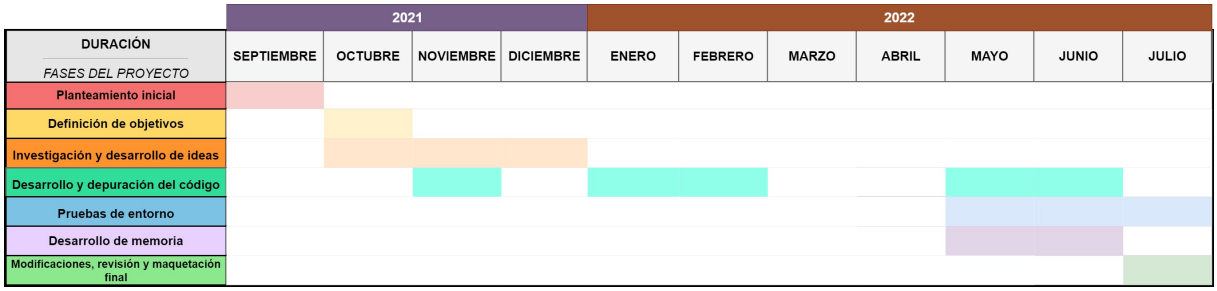

Figura 2.1: Cronograma de realización del proyecto

Para facilitar la organización, estructuración y consecución de objetivos se hace uso de la herramienta Trello[3], allí se dividen las tareas en tres bloques: *Investigativo* (Amarillo), *Desarrollo de código* (Verde) y *Desarrollo de memoria* (Rojo)

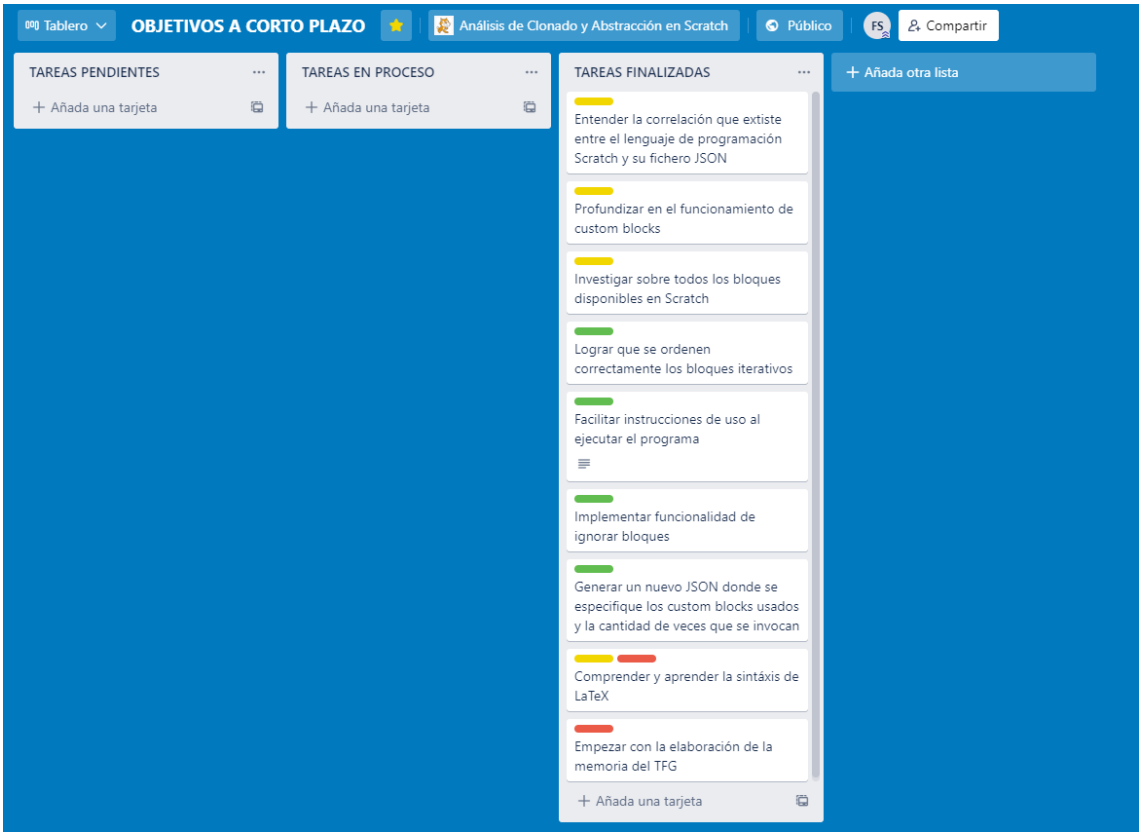

Figura 2.2: Tablero de Trello con objetivos a corto plazo

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

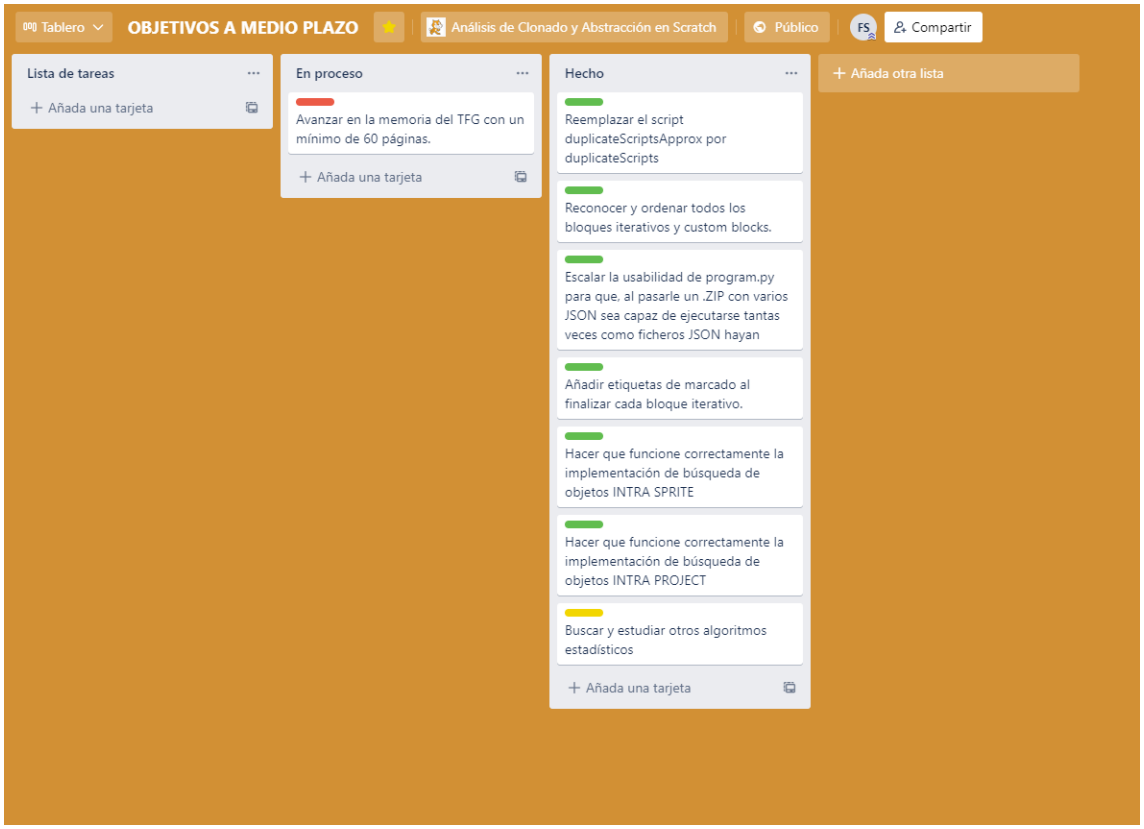

Figura 2.3: Tablero de Trello con objetivos a medio plazo

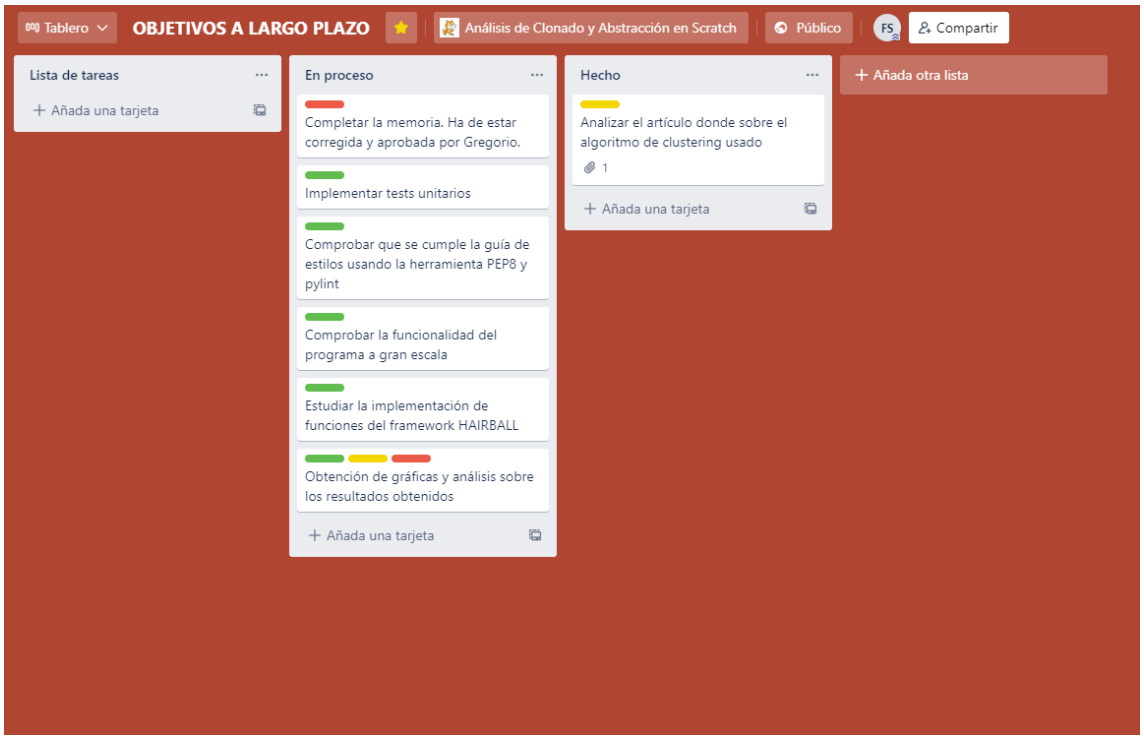

Figura 2.4: Tablero de Trello con objetivos a largo plazo

Capítulo 3

Estado del arte

En este capítulo se introducirán 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 interactuar entre estos elementos. [5].

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 usan Scratch son los que tienen una edad media entre los 13 y los 16 años.

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*. Existen varios categoría de bloques, siendo de especial interes 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.

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. [1]

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 información estructurada que es utilizada principalmente para transferir datos entre un servidor y un cliente. [2]

Tomando como referencia la sintáxis, 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, son un tipo de datos JSON válido. Puede tener la forma de un array, un string, un boolean, un objeto, un número o de tipo nulo.

Es importante mencionar el fichero JSON ya que es el principal elemento usado para la extracción y análisis de datos de mi proyecto en Scratch.

3.3. Python

Es un lenguaje de programación interpretado cuya filosofía hace énfasis en 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

[1] https://drscratchblog.wordpress.com/2015/04/20/habilidades-de-programacion-abstraccion/
[2] https://www.json.org/json-en.html

programación funcional. Python fue creado por Guido van Rossum a finales de los ochenta y su nombre se lo debe a los humoristas británicos *"Monty Python"*. [3]

Existen tres versiones principales de Python pero las más extendidas son las versiones 2 y 3, siendo la versión 3.10 las última actualización disponible. En este proyecto se hace un uso íntegro de la versión 3.9 de Python. Es importante indicar 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 durante su compilación y ejecución.

Este proyecto se apoya en una serie de bibliotecas y módulos de Python que hacen posible realizar el trabajo ejecutado. A continuación se detalla su funcionamiento y utilidad:

3.3.1. Sys

Módulo que permite interactuar con los 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.

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 ya han sido analizados, esto a fin de evitar crear excesivos documentos por cada ejecución.

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

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 un archivo ZIP.

3.3.5. JSON

Módulo que ofrece la posibilidad de manipular ficheros de tipo JSON. Es usado para parsear[4] los elementos del fichero JSON de cada proyecto Scratch.

3.3.6. Logging

Módulo que define funciones y clases para implementar un sistema de registro de eventos de log para aplicaciones y bibliotecas. Es usado para generar un fichero program_logs.txt donde se mostrará información relevante de la ejecución del programa. Es en este fichero donde se escribirán los errores 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 comparando texto e incluye funciones que brindan información usando varios formatos de diferencia comunes.

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

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

heurístico [5] 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.

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, set, y tuplas.[6]

counter

Es una subclase del tipo de datos *dict* para contar objetos separables. Los elementos se almacenan como llaves de diccionario 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 de factory 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. [7] Cuenta con algoritmos de clasificación, regresión, clustering y reducción de dimensionalidad. Además, presenta la compatibilidad con otras librerías de Python como NumPy y matplotlib.

[5] https://conceptodefinicion.de/heuristica/

[6] https://docs.python.org/3/library/collections.html

[7] https://scikit-learn.org/stable/modules/classes.html

cluster

Es una subclase que reúne diferentes algoritmos de agrupación no supervisada *(unsupervised clustering)*. En este trabajo se hará uso del algoritmo **AffinityPropagation** ya que se evidencia que su uso es más óptimo en comparación con otros algoritmos. FALTA POR EXPLICAR Y DESARROLLAR. En la imagen 3.1 se puede ver una comparativa visual sobre los distintos clasificadores de múltiples algoritmos.

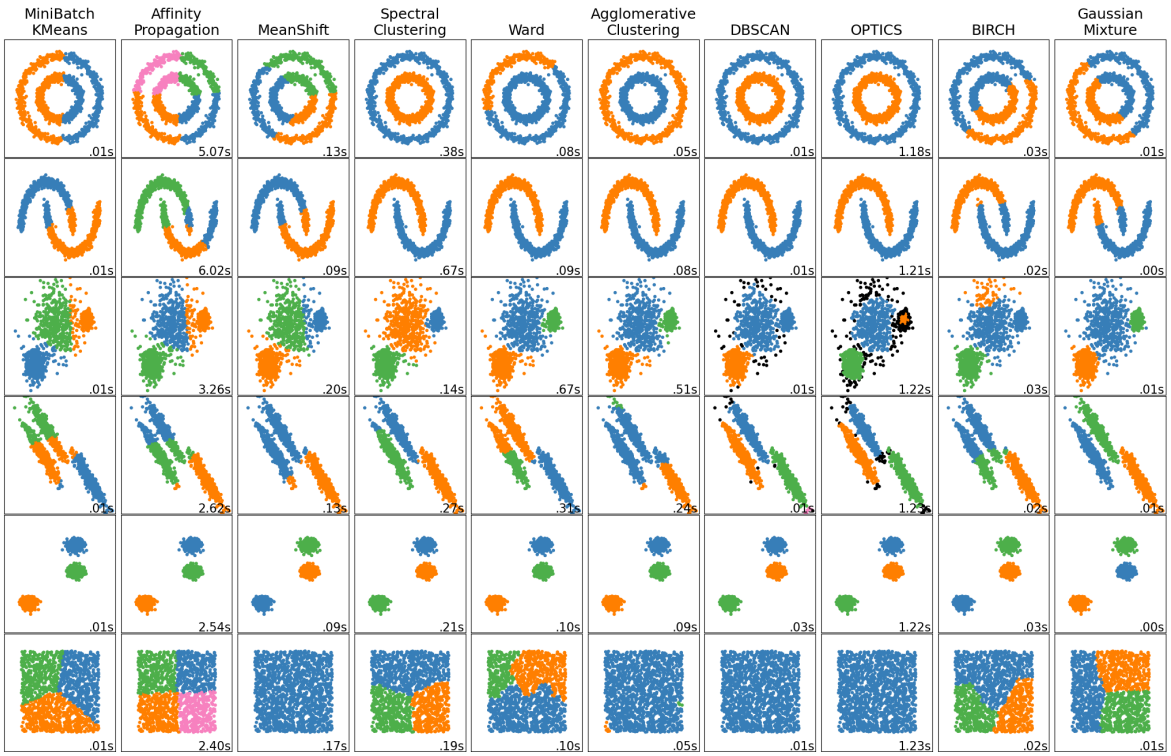

Figura 3.1: Comparativa de algoritmos de clustering

3.4. Hairball

Hairball en un plugin de Python que analiza proyectos de Scratch, ofreciendo a su salida información para el usuario relacionada con 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 representacion en formato de grafo de los datos de un programa en Python. 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, mal nombrado e inicialización de variables.[8]

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. Es el gestor con el que se han instalado todos los módulos y bibliotecas incluidos en este proyecto 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. [9]

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. 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 una de los proyect

- **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 ejecutándolos individualmente.

[8] https://pypi.org/project/hairball/

[9] https://pypi.org/project/pip/

- **Test runner**: es el componente encargado de ejecutar nuestros tests y proporciona un resultado. Puede disponer de una interfaz gráfica, texto o devolver un valor especial que indique el resultado de las pruebas.

FALTA POR DESARROLLAR.

Para iniciar la ejecución de un test con el módulo unittest se puede hacer ejecutando la siguiente orden mediante línea de comandos:

```
$ python -m unittest tests/test_something.py
```

3.7. Guía de estilo PEP8

PEP8 (Python Enhancement Proposal 81) 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. Es la guía de estilo elegida porque se usó durante la elaboración de prácticas de la asignatura Protocolos de Transmisión de Audio y Vídeos por Internet (PTAVI). Algunas de las reglas mas 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, control de versiones y llevar un control sobre el planteamiento y la consecución de objetivos:

- **Trello:** Es usada para gestionar y organizar, a modo de tableros, las fases y tareas por realizar tanto 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:** Es usada para alojar el código y mantener un control de versiones. Resulta muy útil para dar marcha atrás en caso de errores detectados mientras se avanza en la práctica, esto gracias al seguimiento de los cambios en el código

En el diagrama de flujo de la imagen 4.1 se muestra el entorno y las herramientas usadas.

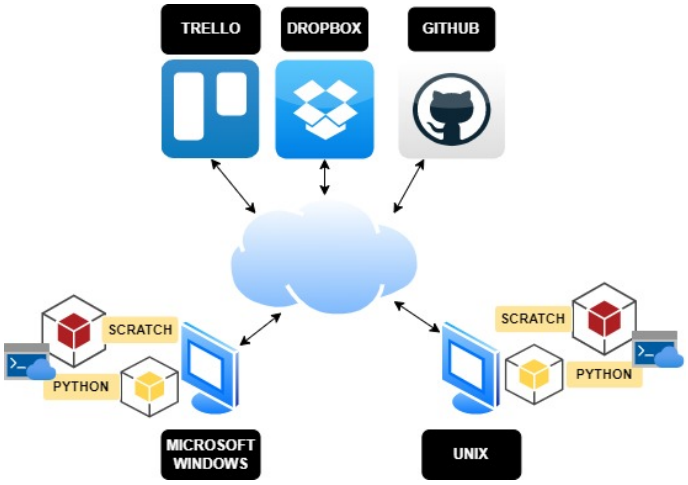

Figura 4.1: Diagrama de entorno y aplicaciones

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 los siguientes scripts, *duplicatescripts.py*, *most_frequent_blocks.py*, *statistics.py* y finalmente, *cluster.py*

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 indicados 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 se escribirán 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 archivo ZIP.

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

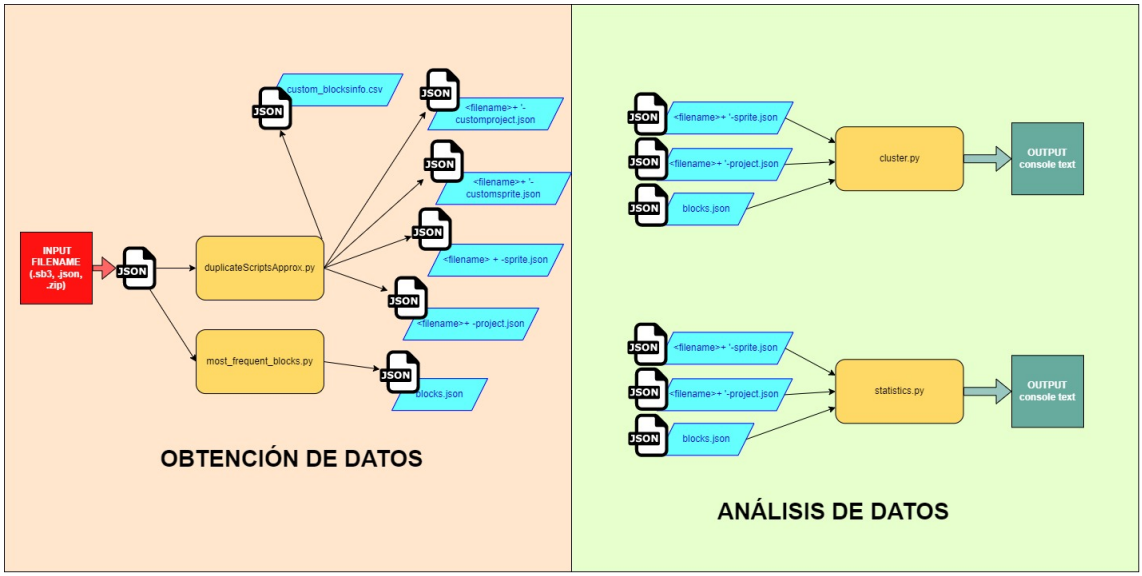

Figura 4.2: Fase de extracción de datos

4.2.1. program.py

Sirve como script principal. Su tarea principal es servir de hilo de ejecución de todo el programa, para ello extrae y parsea el contenido de ficheros JSON. En este programa se declara la forma como se escribirán los registros de control de ejecuciones en el archivo de logs, *program_logs.txt*, útil para depurar y encontrar excepciones durante la ejecución.

4.2.2. duplicateScripts.py

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

El contenido del JSON se recorrerá a modo de diccionario, empezando por la clave *targets* que contiene todos los objetos. Seguidamente se extrae la información relevante, según cada bloque, de la clave *blocks*. En el código se irá almacenando, en una variable de tipo diccionario, la clave/valor: *block_id:opcode*, lo cual será importante para encontrar y añadir, en su posición correcta, a los bloques de control y personalizados. En la imagen 4.3 se aprecia una vista de árbol general de las clave/valor del fichero JSON, y en la imagen 4.4 un diagrama de la estructura general de un bloque genérico.

Todas las categorías de bloques suelen tienen una estructura similar pero no todos sus cam-

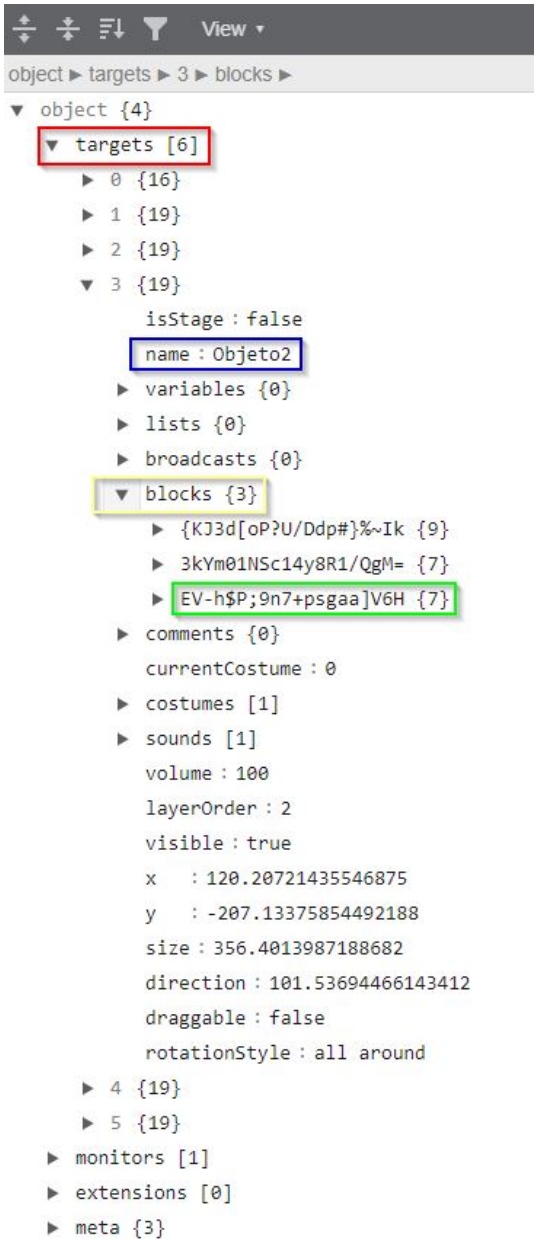

Figura 4.3: Vista de árbol del contenido de un fichero JSON

pos 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 en el 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:

1. Si el valor de **topLevel** es True significa que el elemento es el primer bloque de mi script.

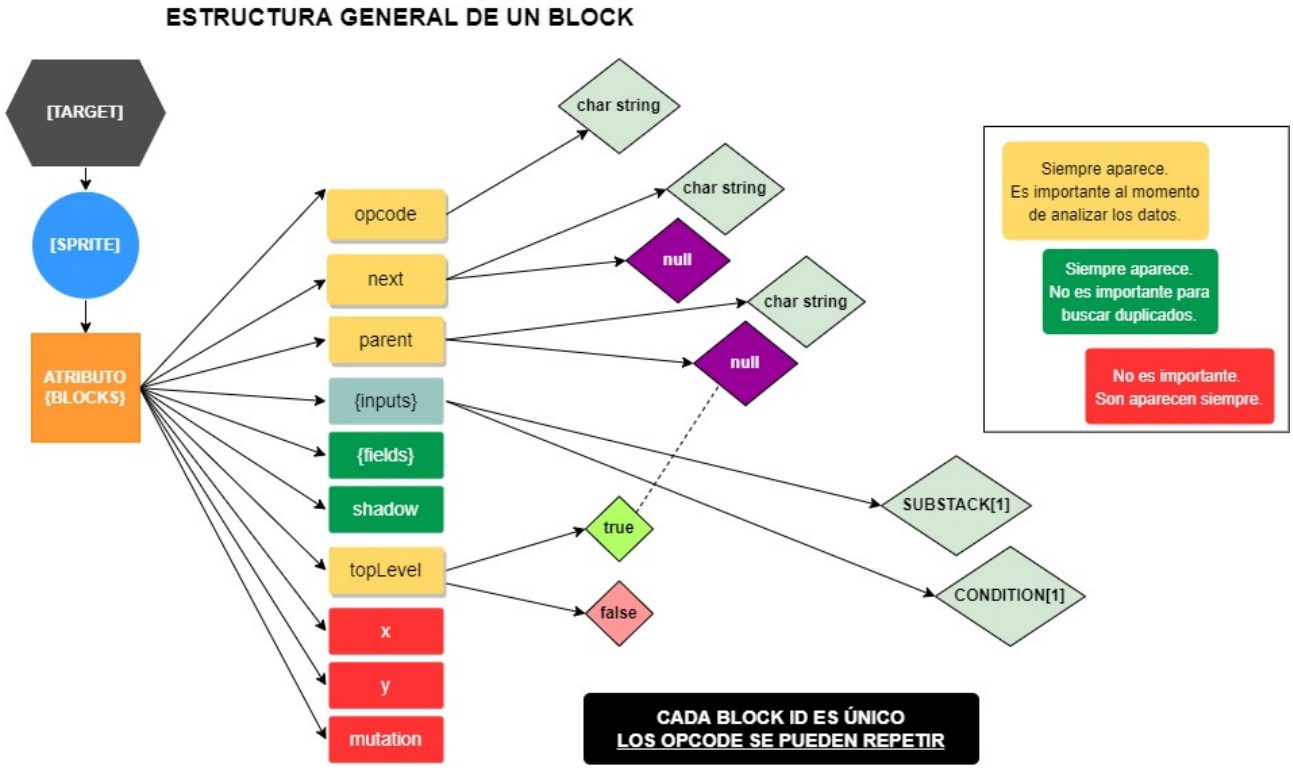

Figura 4.4: Estructura general de un bloque genérico

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 ver un ejemplo de esto.

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 las imagenes 4.6 y 4.7 se ejemplifican estos casos.

6. Para el caso especial del bloque control_if_else el valor **substack2** contiene otra lista con

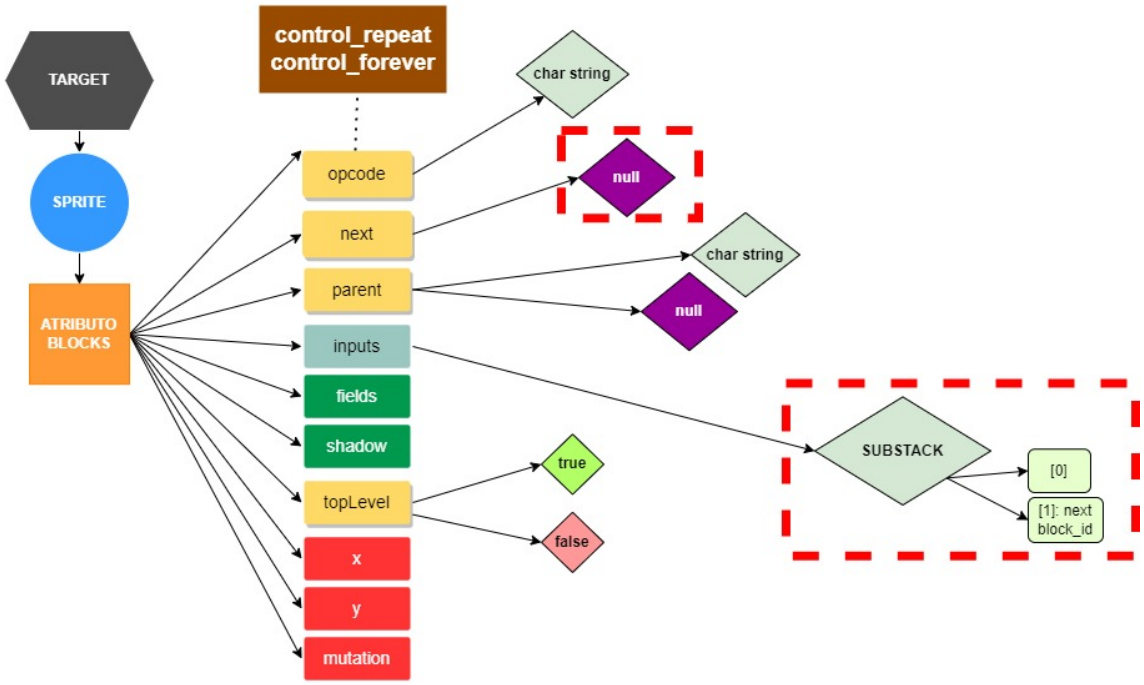

Figura 4.5: Estructura general de un bloque de tipo control_repeat y control_forever

la secuencia de bloques internas en el hilo condicional. En la imagen 4.11 se puede ver un ejemplo de la estructura del JSON y en la imagen 4.12 un script y sus elementos.

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

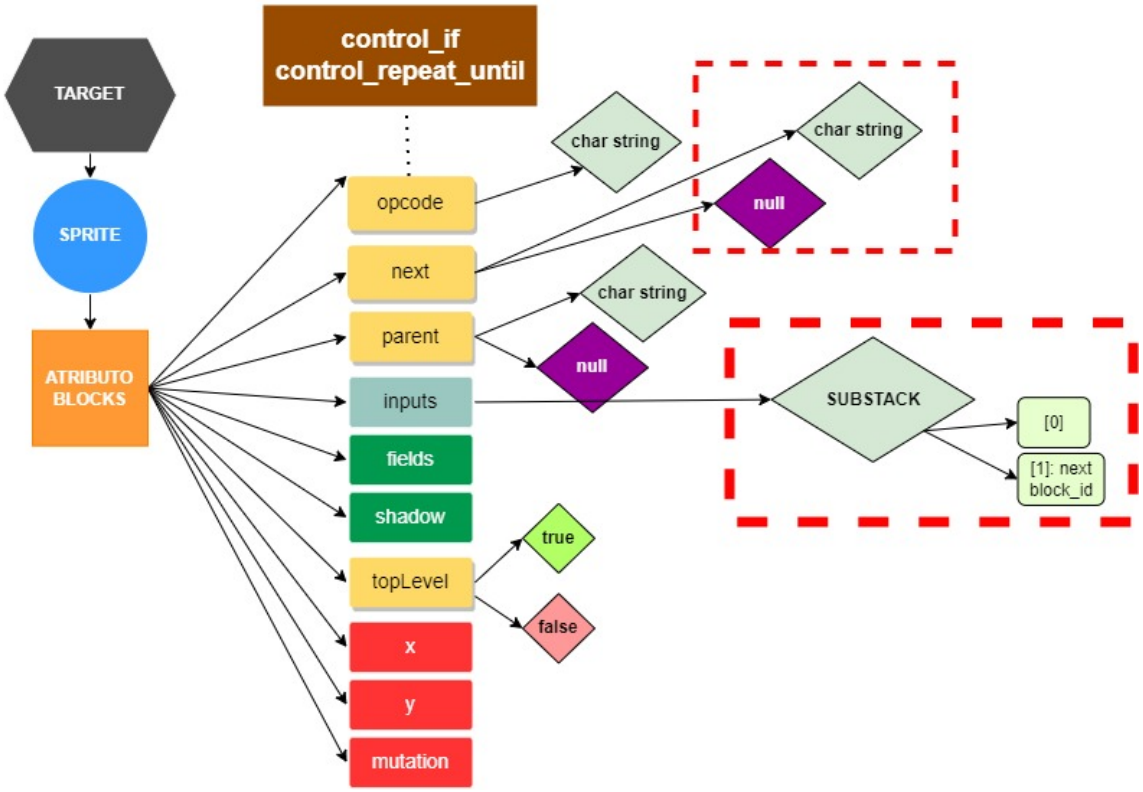

Figura 4.6: Estructura de un bloque condicional de tipo control_if o control_repeat_until

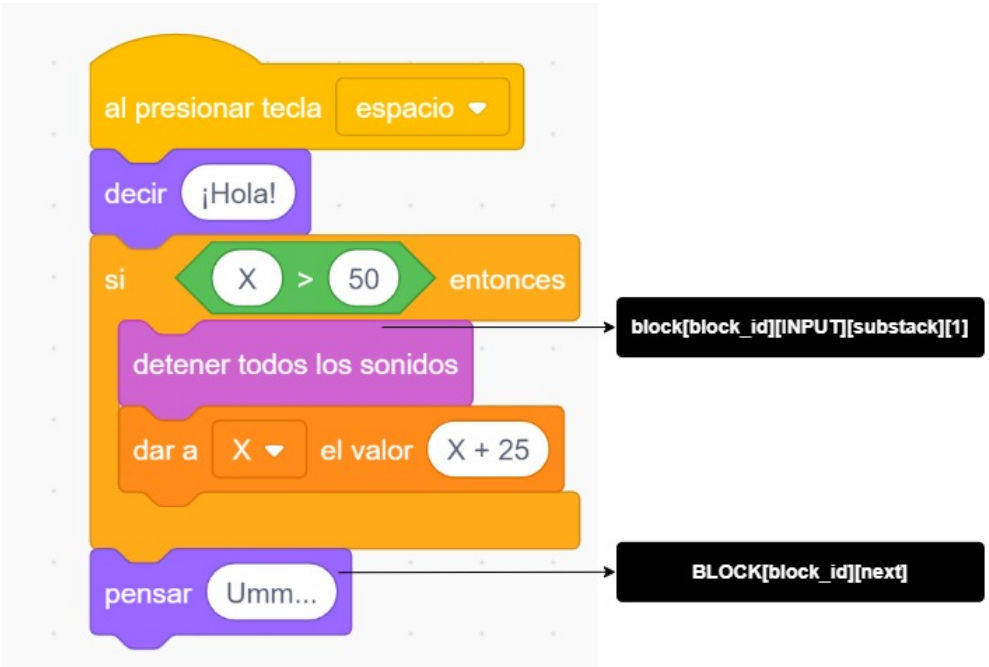

Figura 4.7: Script con bloques de control

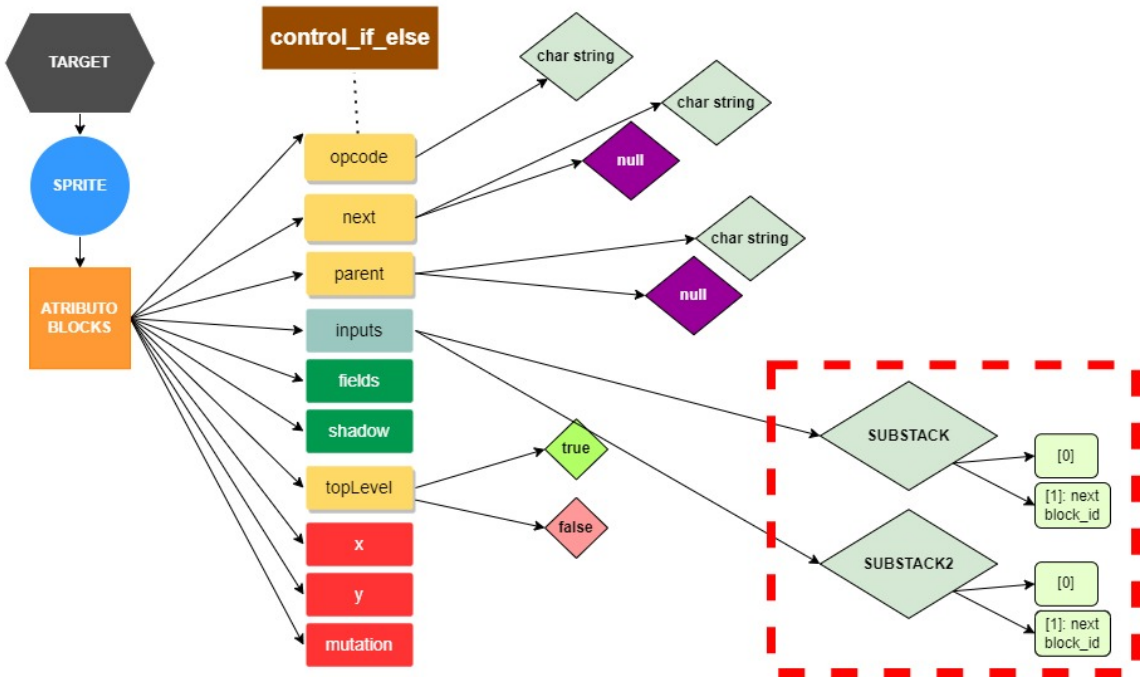

Figura 4.8: Estructura general de un bloque condicional de tipo control_if_else

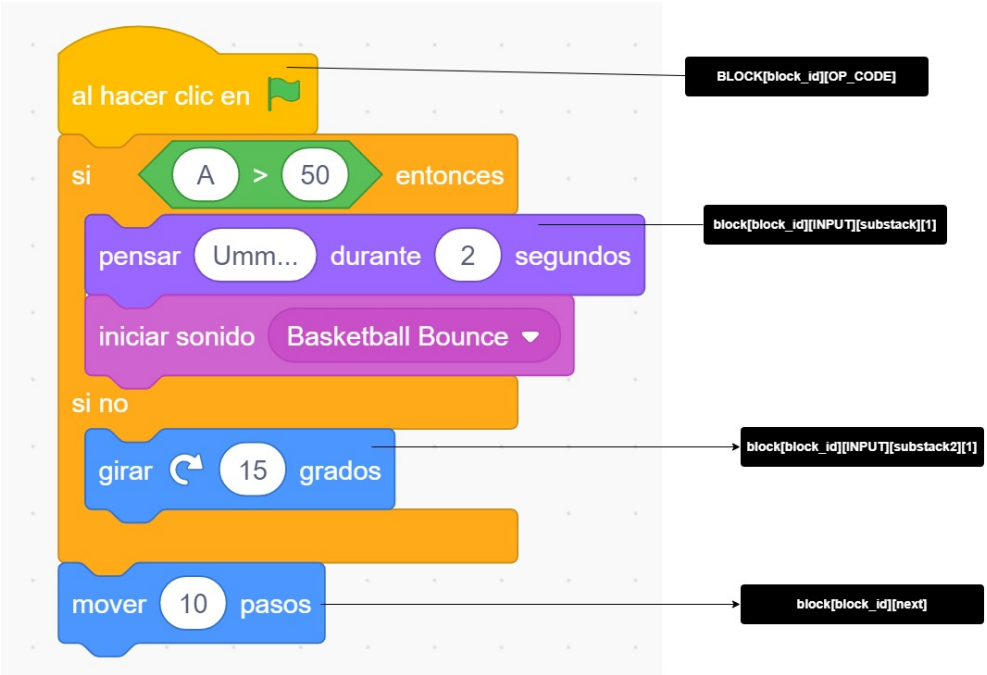

Figura 4.9: Script con bloque condicional de tipo control_if_else

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 que se hagan.

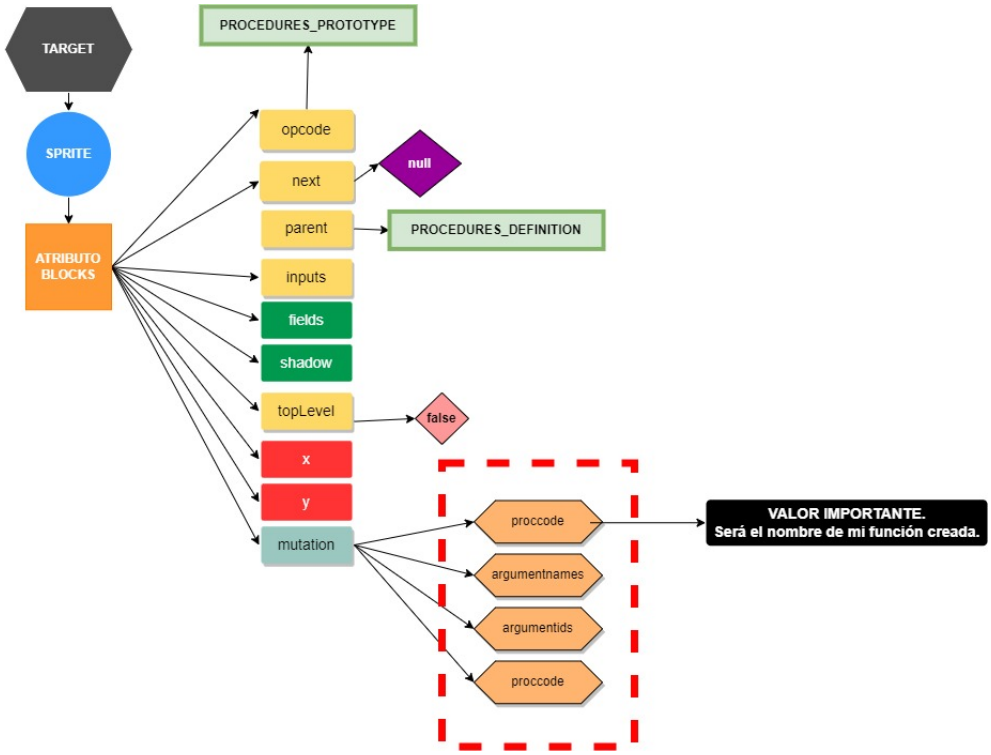

Figura 4.10: Estructura general de un bloque de tipo block_prototype

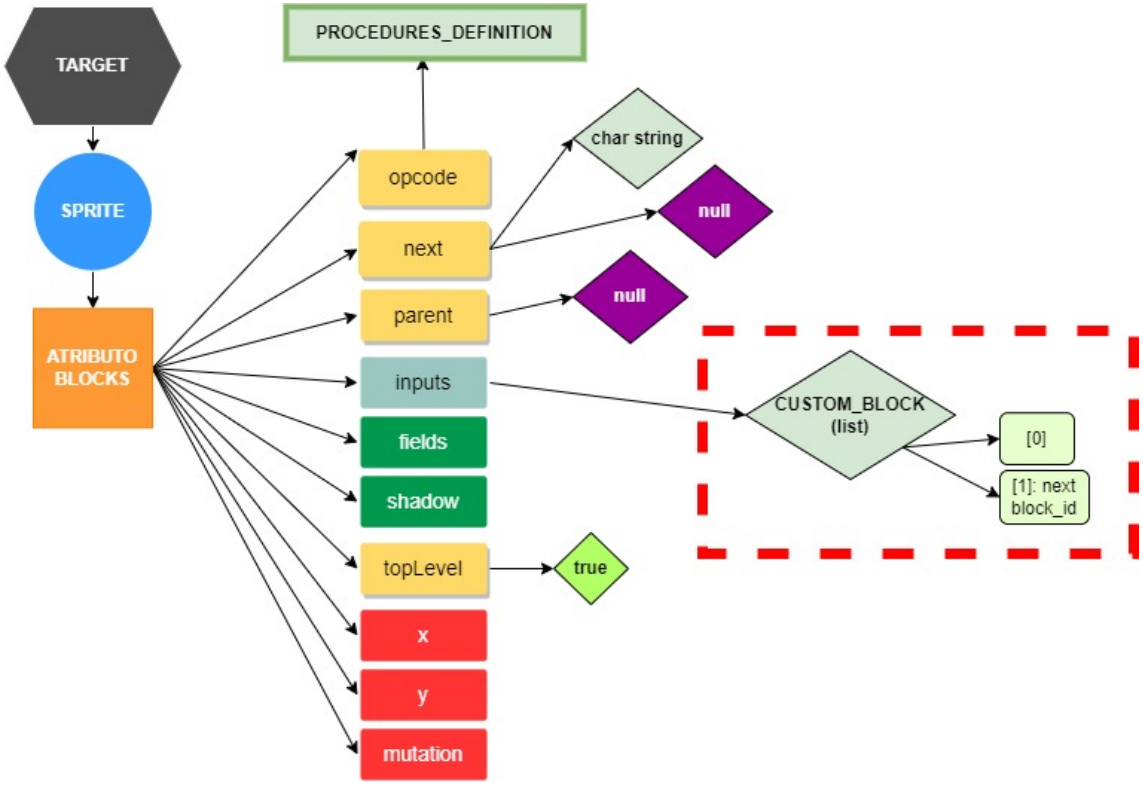

Figura 4.11: Estructura general de un bloque de tipo block_definition

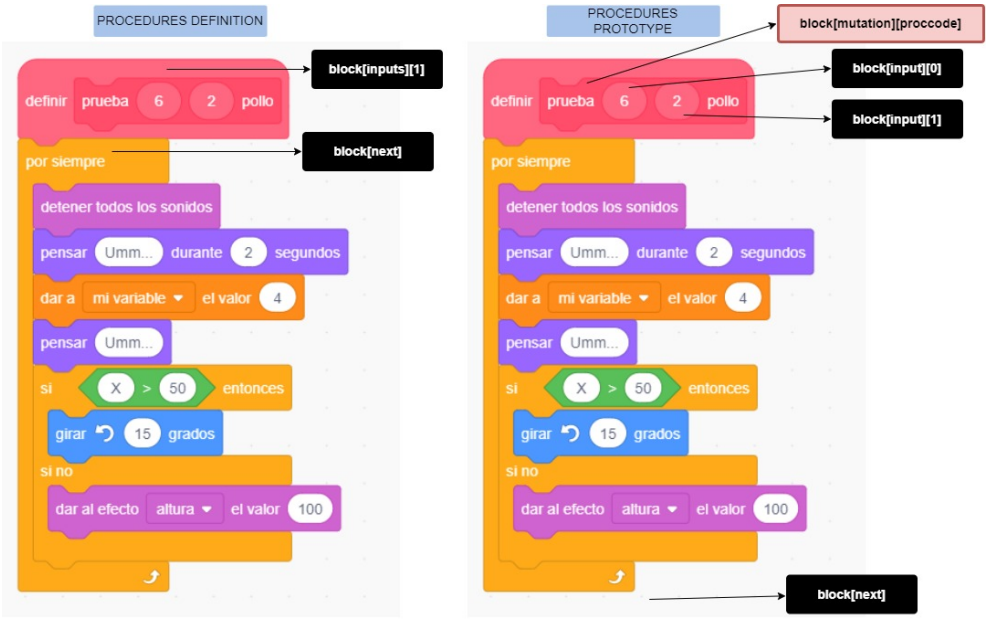

Figura 4.12: Diferencias entres los bloques custom_block

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, en el otro diccionario *self.blocks_dict* que almacena como clave el block_id de cada bloque y como valor su opcode. Además, este último es vital 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 (*wide-project*). 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 mas se repitan.

4.3. Obtención de datos

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

- ”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 las cadena de bloques personalizados mas repetidos en todo el programa.

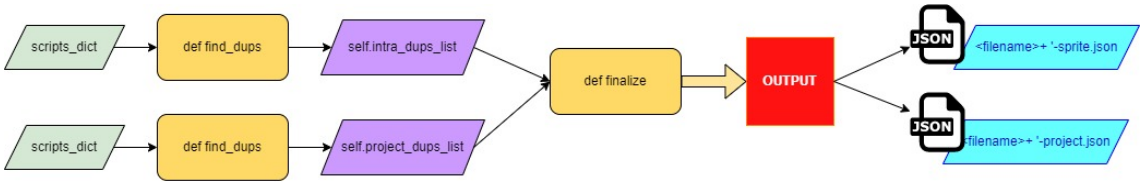

Figura 4.13: Diagrama de la última fase de ejecución duplicateScripts.py

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**. Se puede apreciar el diagrama del funcionamiento del script en la imagen 4.14

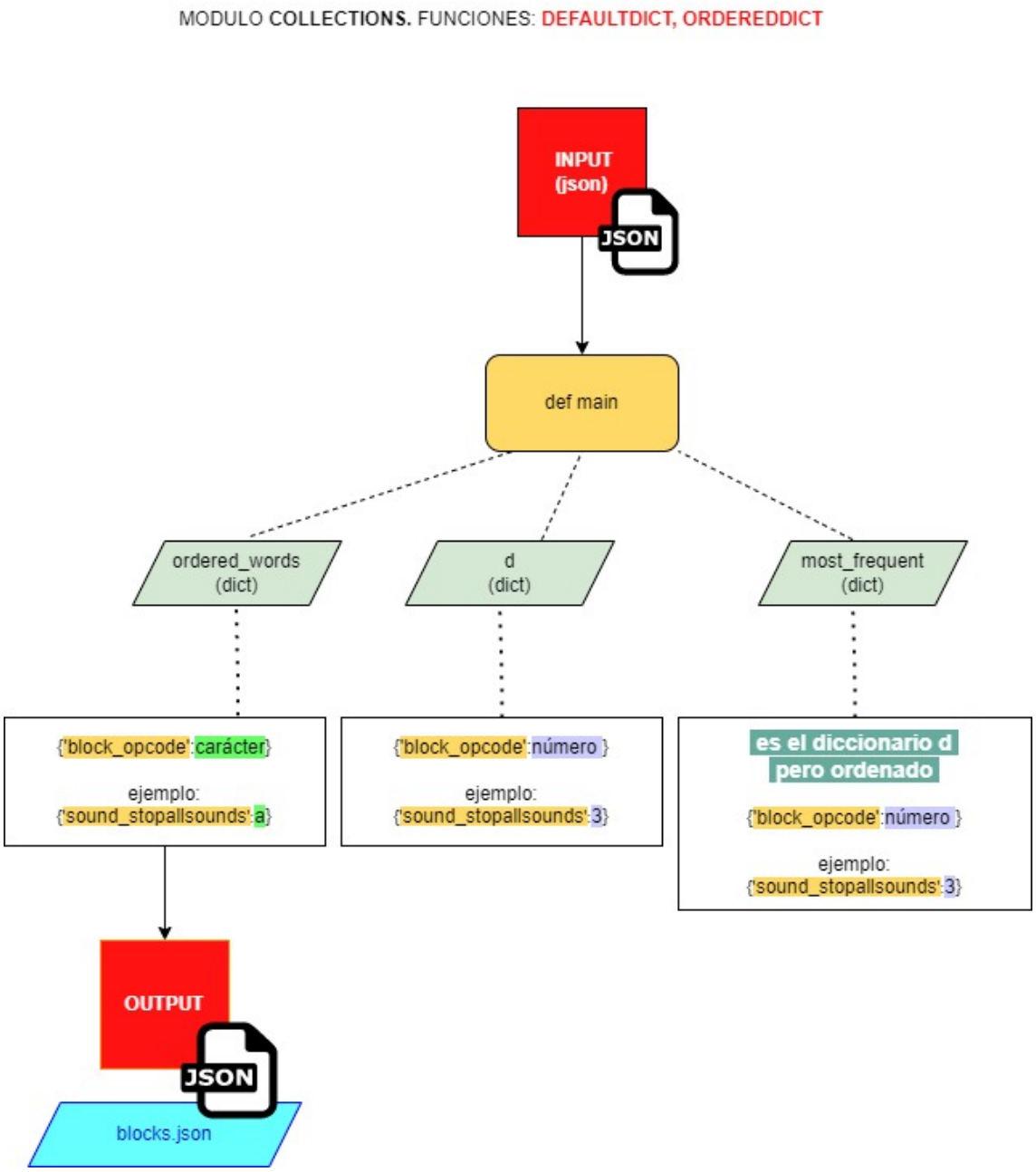

Figura 4.14: Diagrama de funcionamiento de most_frequent_blocks.py

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 conte-

nido 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, el código recorre una lista de duplicados y, para cada script en la lista, asigna una nueva letra a cada bloque que no tenga una asignada.

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.

La función json2dna devuelve la lista y... FALTA POR DESARROLLAR

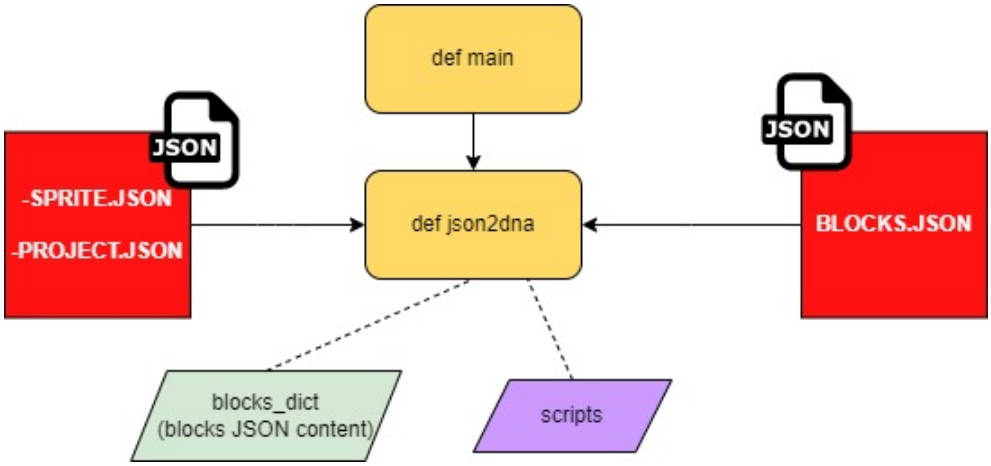

Figura 4.15: Diagrama de funcionamiento de statistics.py

4.5. Análisis de datos

4.5.1. cluster.py

Un algoritmo de clusterización es un método para agrupar objetos de tal manera que los objetos en el mismo grupo (cluster) sean similares entre sí, mientras que los objetos en diferentes grupos (clusters) sean lo suficientemente diferentes.

El algoritmo que se emplea es El algoritmo de propagación de 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.

FALTA POR DESARROLLAR.

4.6. Visualización de resultados

Se visualiza en el siguiente formato por línea de comandos:

```
*** STARTING ANALYSIS  ***

-- STARTING DUPLICATESCRIPTS.PY SCRIPT --

Looking for duplicate blocks in (FILENAME)

Minimum number of blocks:  X

XX total blocks found in all project
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 --
```

Capítulo 5

Experimentos, validaciones y resultados

Hablar de la pruebas de escalabilidad.

FALTA POR DESARROLLAR.

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 podría tener 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 de código. Esto hace que sea muy fácil duplicar código 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 custom_blocks. Por ejemplo, si creamos una función elemental en el objeto ”personaje”, solamente podremos utilizarla dentro de dicho objeto, obligando a crear nuevamente la función para tantos objetos como 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 también puede conducir a errores si el código no se actualiza correctamente.

También se ha podido comprobar en la literatura científica [2] que los proyectos de Scratch hacen uso extensivo del clonado lo cuál puede suponer una limitación a la hora de desarrollar habilidades del pensamiento computacional, especialmente, la abstracción.

FALTA POR DESARROLLAR.

6.1. Consecución de objetivos

FALTA POR DESARROLLAR.

6.2. Conocimientos aplicados

La capacitación y el desarrollo de los conocimientos 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, el conocimiento también puede adquirirse a través de la capacitación y el desarrollo de estas habilidades.

Durante el transcurso del grado, me percate que el éxito requiere de una combinación de capacitación, desarrollo de los conocimientos y experiencia. Esto es ciertamente aplicable a todas las habilidades que he desarrollado a lo largo de estos años. Quiero enumerar, por asignaturas, dichas habilidades aprendidas y que han facilitado realizar este trabajo.

1. Protocolos para la Transmisión de Audio y Vídeo en Internet

FALTA POR DESARROLLAR.

6.3. Conocimientos aprendidos

FALTA POR DESARROLLAR.

6.4. Trabajos futuros

FALTA POR DESARROLLAR.

Bibliografía

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

[2] 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.

[3] 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.

[4] 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/.

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

Apéndice A

Manual de usuario

USO

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

-i (OPCIONAL): Ignora los opcodes de bloques especificados en IgnoreBlocks.txt

USAGE

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

-i (OPTIONAL): Ignores blocks opcodes specified in IgnoreBlocks.txt

Apéndice B

Requisitos

Para poder hacer uso del programa es importante tener instalado los módulos y bibliotecas de Python necesarias para el funcionamiento del programa, por ello es necesario ejecutar el siguiente comando:

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