UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

# HERRAMIENTA PARA ANÁLISIS DE PROGRAMAS DESARROLLADOS EN SCRATCH

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

MARÍA JOSÉ BERGER TRUJILLO

PROFESORA GUÍA:
JOCELYN SIMMONDS WAGEMANN

MIEMBROS DE LA COMISIÓN:
FEDERICO OLMEDO BERÓN
LUIS MATEU BRULE

SANTIAGO DE CHILE
2021

# Resumen

El Departamento de Ciencias de la Computación de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile (DCC) dicta periódicamente cursos dirigidos a niños enfocados en el desarrollo de programas en el lenguaje de programación en bloques Scratch, evaluando sus resultados con fines investigativos sobre su impacto en el aprendizaje sobre pensamiento computacional.

Actualmente, para evaluar los programas desarrollados en los cursos se utilizan dos herramientas: una rúbrica de evaluación de diseño propio y la aplicación web Dr. Scratch, la cual genera un reporte de análisis automatizados. Si bien el uso conjunto de estas herramientas ha resultado efectivo en la evaluación de los programas, existen limitaciones para las cuales aún no se encuentra solución. La principal, presente en ambos métodos, tiene que ver con el tiempo. En Dr.Scratch, los proyectos se analizan individualmente. En la rúbrica, se deben invertir horas de trabajo humano que no siempre están disponibles. Esto ha traído como consecuencia que actualmente exista un retraso de 2 años en el análisis de los programas y por ello, hay datos valiosos para la investigación que no están siendo considerados.

En este trabajo de título se propone crear un sistema que permita analizar un programa en Scratch, generando un reporte que será retroalimentado con la rúbrica mencionada. Se estudió el proceso de evaluación actual identificando las oportunidades de mejora y se diseñó una plataforma que unifica todos los ejes de evaluación actuales y administra de mejor manera los datos de cada curso.

Dado que Scratch no posee especificación formal, la estrategia utilizada fue implementar un parser guarde un programa en Scratch en la base de datos como un Diagrama de Control de Flujo con el objetivo de utilizar prácticas tradicionales de navegación de grafos y consultas de bases de datos para ejecutar distintos análisis.

Se implementó una aplicación web que expone una API y una interfaz donde se cargan programas individual o masivamente para su análisis. En ella, se integró la rúbrica de evaluación como complemento, reduciendo los tiempos de tabulación. Por último, se incorporó la visualización de los Grafos de Control de Flujo generados.

Este trabajo representa un primer prototipo a lo que se espera sea la principal herramienta de evaluación que utilicen los docentes en su investigación. Al elevar tecnicidad de los análisis y unificar todas las herramientas, se espera simplificar el proceso de evaluación obteniendo mejores datos en menos tiempo.

***A mi mamá, Paulina***
*Que me inculcó la importancia de estudiar y estuvo para recogerme cada vez que sentí que el mundo se venía abajo.*

***A mi abuela, Leonor***
*Que, aunque aún no entiende qué es lo que estudié, siempre será mi fan número 1.*

***Al resto de mi familia***
*Que supo que podía lograrlo antes de que yo misma me diera cuenta.*

***A mis amigos***
*Que sin ellos no hubiese sido posible.*

***Y a mi misma***
*Porque sí se pudo.*

# Agradecimientos

Gracias a mi mamá, por aconsejarme y guiarme a lo largo de estos 24 años de vida. Por ser mi mejor amiga, por contenerme cada vez que lo necesité, obligarme a valerme por mi misma y defenderme de cualquier cosa. Por enseñarme lo valioso que es ser responsable y ayudarme día a día a ser mejor persona.

Gracias a mi abuela por siempre tenerme la fe que yo no me tenía, por enseñarme cosas que no se pueden aprender de un libro y por rezarle a San Pancracio cada vez que necesité un milagro. Que fue bastante seguido.

Gracias a mi profesora guía, Jocelyn Simmonds, por aceptarme como memorista y confiar en que podía hacer un buen trabajo incluso cuando yo pensaba que no. Por escuchar una vez a la semana todas mis crisis existenciales con respecto al trabajo de título, mi futuro laboral y la vida en general. Por aportarme no solo sabiduría sino que también humanidad.

Finalmente, gracias a todos mis amigos y amigas, de antes y durante y después de mi paso por la universidad. Sin ustedes, con su contención emocional, sesiones maratónicas de estudio y risas que no paraban hasta hacerme llorar, jamás lo hubiese logrado.

# Tabla de Contenido

# Índice de Ilustraciones

# Capítulo 1

# Introducción

En la última década, la computación ha adquirido un papel fundamental en la vida del ciudadano promedio, volviéndose cada día más parte de la "cotidianidad" y dejando de ser un área exclusiva para expertos en el tema. Debido al crecimiento exponencial de las necesidades tecnológicas de la sociedad, se vuelve indispensable introducir a las personas, desde temprana edad, herramientas que les permitan tomar acción activa en los procesos de desarrollo tecnológico.

## 1.1. Antecedentes

Con el avance irrefrenable de la Ciencia y la Tecnología, la sociedad ha tenido que adaptarse, tanto en infraestructura como en educación, para no quedarse atrás. A pesar de las diversas iniciativas que se han tomado en Chile y el mundo por adaptarse a las nuevas necesidades tecnológicas, se estima que desde el año 2015 en adelante, existe y existirá en Chile un déficit del 39 % en profesionales especializados en Tecnologías de Información (TI) [5].

La inminencia de la llamada "era digital" ha motivado a diversas escuelas y universidades de todo el planeta a incorporar en sus actividades curriculares cursos de programación y desarrollo digital. Sin embargo, en la mayoría de estos programas no se enseñan métodos de Ingeniería de Software, es decir, no incluyen las metodologías necesarias para producir software de calidad y llevar buenas prácticas de programación en general [10].

Estudios señalan que métodos fundamentales de la Ingeniería de Software pueden ser enseñados, en forma de curso curricular o programa introductorio, a niños y niñas desde temprana edad [10]. Esto favorecería sus posibilidades de aprender técnicas avanzadas de programación en el futuro. Una de las herramientas más populares utilizadas dentro de los estudios mencionados es la de Programación en Bloques [10] pues son más visuales e interactivas en comparación a la programación tradicional.

La programación en bloques recibe su nombre debido a que cada instrucción de un programa es representada por este elemento, siendo la unión de dos o más bloques un script. Uno de los principales exponentes de este tipo de lenguajes es Scratch [6]: un lenguaje de programación dirigido a niños y niñas entre 8 y 16 años el cual permite la creación de his-

torias interactivas, juegos y animaciones, de forma que sus usuarios puedan desarrollar su capacidad de expresión, pensamiento lógico y creatividad.

En este contexto, el Departamento de Ciencias de la Computación de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, desde ahora DCC, dicta periódicamente cursos de 5 días de duración enfocados en el desarrollo de programas en Scratch, evaluando sus resultados con fines investigativos [9, 8, 11]. Los investigadores a cargo de estos cursos proponen que estas instancias educativas funcionan como un complemento a las actuales fuentes de aprendizaje sobre pensamiento computacional y desarrollo de proyectos. Con esta iniciativa, los estudiantes pueden desarrollar buenas prácticas de Ingeniería de Software desde antes de iniciar su educación formal en Ciencias de la Computación [9].

A grandes rasgos, el curso dictado por el DCC consiste en enseñar el uso correcto de Scratch y sus posibles aplicaciones, enfocándose en la aplicación de buenas prácticas de desarrollo. Más específicamente, las clases se dividen en 2 módulos: En el primero se realizan clases expositivas apoyándose en programas provistos por el equipo docente. En el segundo módulo, los alumnos divididos en grupos, deben desarrollar su propio programa en Scratch utilizando los nuevos conocimientos adquiridos y el material de ejemplo disponible.

Una vez finalizado el curso, los programas desarrollados por los alumnos deben ser evaluados por los docentes, enfocándose tanto en el proyecto en sí mismo como en el nivel de aprendizaje alcanzado por los alumnos. Los resultados obtenidos en estas evaluaciones son utilizados como datos para la investigación de la efectividad del curso sobre los estudiantes y si las metodologías aplicadas propician en aprendizaje y acercamiento al área de la programación.

## 1.2. Motivación

Como se mencionó anteriormente, los programas desarrollados por los alumnos deben ser evaluados por los y las docentes del curso de forma tal que permita extraer datos útiles para la investigación. Actualmente, se utilizan dos principales herramientas de análisis: la primera consiste en una rúbrica de evaluación diseñada por los investigadores [9], la cual implica la evaluación manual e individual de los proyectos. La segunda herramienta es la aplicación web Dr. Scratch [3], la cual identifica automáticamente la presencia de *code smells* y otras malas prácticas de desarrollo. Además, evalúa y califica la complejidad del programa.

Si bien el uso conjunto de ambas herramientas ha resultado efectivo en la evaluación de los programas desarrollados por los alumnos del curso, existen evidentes limitaciones para las cuales aún no se encuentra una solución. Una de ellas es que Dr. Scratch utiliza ciertas prácticas evaluativas que fuerzan una línea especifica al momento de desarrollar, pues asigna altos puntajes a determinadas acciones del código sin tomar en consideración los objetivos del programa. Además, es poco transparente a la hora de asignar los puntajes al código, pues no se puede saber por qué fueron añadidos o descontado puntos. Por otro lado, con la rúbrica descrita en *Assessing Software Development Skills among K-6 Learners in a Project-Based Workshop with Scratch*[9] se obtiene un resultado más acertado a la complejidad y correctitud del programa, sin embargo, es un proceso manual que requiere más tiempo y puede llevar a posibles errores.

Otra problemática identificada en ambos métodos de evaluación consiste en el volumen masivo de datos (programas) que se obtienen en cada edición del curso. Para evaluar en Dr. Scratch, es necesario subir los proyectos uno a uno para su análisis. Para complementar con la aplicación de la rúbrica, se deben invertir horas de trabajo humano que no siempre están disponibles. Esto ha traído como consecuencia que actualmente exista un retraso de 2 años en el análisis de los programas y por ello, hay datos valiosos para la investigación que no están siendo considerados.

Ante esto, surge la necesidad de una herramienta capaz de generar un análisis semi-automatizado de un programa desarrollado en Scratch, utilizando conceptos tanto de la rúbrica descrita como de Dr. Scratch de forma de potenciar los aspectos positivos de ambas formas de evaluación y compensar las falencias detectadas.

## 1.3. Objetivos

A continuación se presentan el objetivo general y los objetivos específicos de este trabajo de título.

### Objetivo general

El objetivo general de este trabajo de memoria es mejorar y expeditar el análisis de los programas desarrollados en los cursos de Scratch del DCC. Esto mediante el diseño e implementación una herramienta que permita evaluar un programa desarrollado en Scratch bajo diversos análisis, generando un reporte con la evaluación que puede ser retroalimentado con información adicional por los docentes del curso.

### Objetivos específicos

Para alcanzar el objetivo general expuesto, se propone cumplir con los siguientes objetivos específicos.

1. Seleccionar las tecnologías adecuadas para convertir un programa en Scratch a un Grafo de Control de Flujo (CFG).
2. Definir y aplicar sobre cada CFG métricas y análisis que determinen su complejidad en diferentes dimensiones.
3. En base a los análisis de cada CFG generar un reporte de resultados y una descripción de los elementos presentes y sus interacciones.
4. Diseñar e implementar un prototipo que permita llevar a cabo el flujo descrito, exhibiendo los datos obtenidos en forma de un perfil del proyecto que incluya la rúbrica de evaluación diseñada por el equipo docente.
5. Validar el prototipo comparándolo con los resultados obtenidos por la revisión manual aplicada en los cursos de Scratch del DCC. Además, solicitar a expertos (como los docentes de los cursos mencionados) y correctores de las ediciones previas de curso que prueben el prototipo y den su opinión.

Para cumplir con los objetivos mencionados, se propone el diseño e implementación de una

aplicación web que permita a los docentes unificar su sistema de evaluación actual y reduzca los tiempos dedicados a tabulación de resultados. El sistema debe considerar la creación de un parser de datos para el lenguaje Scratch que permita procesar los programas con la finalidad de ejecutar análisis sobre ellos y la exposición de dichos análisis con un nivel de tenicidad superior al proporcionado con las herramientas actuales. Además, debe mantener la consistencia con las investigaciones realizadas sobre versiones previas de los cursos incluyendo la rúbrica de evaluación que ha sido utilizada en ellos.

La metodología de trabajo fue un desarrollo de forma incremental separado en 3 etapas secuenciales iniciales: parser y análisis, backend, frontend y luego continuar añadiendo funcionalidad de forma paralela a las 3 etapas hasta lograr un producto mínimo viable.

## 1.4. Estructura de la memoria

El resto del documento se estructura como sigue: en el Capítulo 2 se presentan conceptos necesarios para entender el lenguaje de programación Scratch y la metodología de los cursos del DCC. Se explicita la forma de evaluación actual de los proyectos realizados en dichos cursos y se analizan otras alternativas comerciales al problema presentado junto a algunas tecnologías que podrían complementarlo de buena manera.

Luego en el Capítulo 3 se revisan los requerimientos del proyecto y se plantea y analiza la solución escogida, mientras que en el Capítulo 4 se explica como fue realizada la implementación de esta. En el Capítulo 5 se revisa la validación de la solución implementada, y finalmente, en el Capítulo 6 se exponen las conclusiones del trabajo realizado y se presentan lineamientos para futuras mejoras en el proyecto.

# Capítulo 2

# Marco teórico

En esta sección se explicarán de forma breve algunos conceptos y herramientas utilizadas en este trabajo de título que no son de conocimiento general. En la sección 2.1 se detallará el funcionamiento de Scratch y los datos asociados a sus elementos. En la sección 2.2 se detallarán las herramientas utilizadas actualmente por los docentes de los cursos de Scratch del DCC para analizar los proyectos. Finalmente en la sección 2.3 se mencionarán algunas herramientas disponibles para el análisis de proyectos sean o no incorporadas a este trabajo de título.

## 2.1. Scratch

Como se mencionó previamente, Scratch es una herramienta de programación basada en bloques dirigida a niños y jóvenes. Es de uso gratuito y se encuentra disponible para en `https://scratch.mit.edu/`, donde puede utilizarse de manera Online o descargar su versión de escritorio. Actualmente, se encuentran en la tercera versión de su lenguaje de programación (exportable a formato .sb3) y existe retrocompatibilidad con las versiones anteriores.

### 2.1.1. Plataforma

La interfaz de la plataforma, disponible en la Figura 2.1, se divide en 3 secciones principales: A la izquierda de la figura se encuentran los bloques disponibles para representar las acciones del programa. En la zona central, el espacio de trabajo para realizar los distintos ensamblajes de bloques para el personaje seleccionado. Finalmente, en el sector derecho de la interfaz, en la parte superior se encuentra el recuadro de ejecución, donde se puede observar qué ocurre al correr el programa. Y en el recuadro inferior, los personajes o elementos presentes en el programa, con sus características.

En este lenguaje, cada bloque representa una acción diferente y a cada personaje del programa se le puede asignar uno o más conjunto de bloques. Los bloques se dividen en nueve categorías según la funcionalidad general que representan y que se pueden observar en la Figura 2.2. Una descripción completa de las categorías y sus bloques asociados se puede observar en el Apéndice A.1.

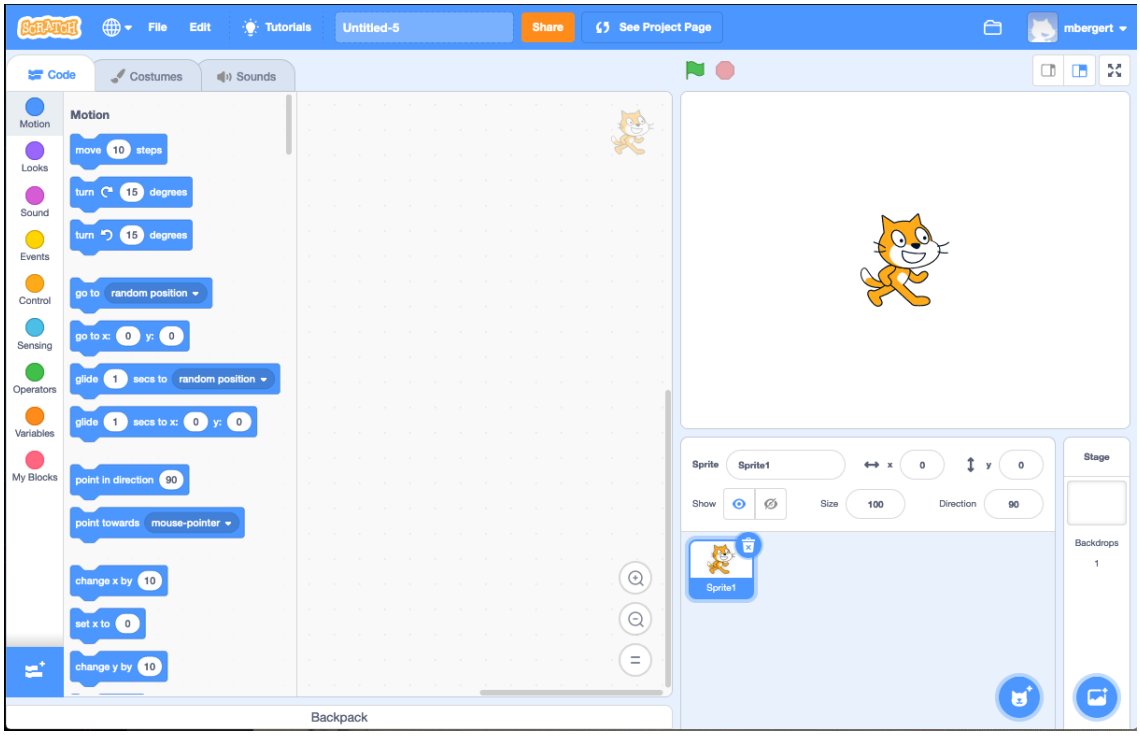



Figura 2.1: Interfaz básica de Scratch

La estructuración de un programa es, en general, libre. Sin embargo, algunas reglas se deben seguir sin importar el tipo de programa que se desee desarrollar. La más importante de ellas es que para cualquier secuencia de bloques, el primer bloque debe ser de tipo "Evento", de otra forma esa secuencia nunca será ejecutada y quedará como código inalcanzable dentro del programa. Cuando se ejecuta el bloque de tipo acción, el resto de la secuencia de bloques se ejecutará linealmente (de arriba hacia abajo) una vez. Para ejecutar secuencias de bloques varias veces, estas deben encontrarse dentro de una estructura cíclica (Tipo "Control") pero siempre debajo del bloque de Evento.

Los programas pueden ser tan simples o complejos como lo desee el usuario. Su complejidad dependerá del número de bloques utilizados y la naturaleza de los mismos. Los bloques se pueden ensamblar de dos maneras: secuencial y anidada. En los bloques ensamblados de forma secuencial se instala un bloque bajo el anterior para ejecutarse en ese mismo orden. En la forma anidada se ensambla un bloque dentro de otro. Generalmente, los bloques anidables son de tipo "Control" y ejecutan instrucciones condicionales o "loops". Los bloques dentro del bloque anidable se ejecutarán solo si se cumple la condición que este posee, en caso de "loops" se ejecutarán repetitivamente hasta que se rompa la condición de quiebre del bloque, en caso de existir.

En la Figura 2.3 se pueden apreciar dos secuencias de bloques, una completamente secuencial (izquierda) y otra de tipo anidado (derecha). Al inicio de los bloques secuenciales se observa el bloque de tipo "Control" que indica que se ejecutará si se hace click en la bandera. Al ejecutar esta acción se ejecutarán todos los bloques. Por otro lado, en la derecha se observa que el bloque inicial corresponde a presionar la tecla espacio. Por ende, cada vez que se presione se evaluará la condición del bloque azul. En caso de ser verdadera se ejecutará

| Categoría | Notas | Categoría | Notas |
|---|---|---|---|
| Movimiento | Mueve objetos y cambia ángulos. | Eventos | Contiene manejadores de eventos situado al principio de cada grupo de instrucciones. |
| Apariencia | Controla el aspecto visual del objeto, añade bocadillos de habla o pensamiento, cambia el fondo, ampliar o reducir. | Control | Sentencian condicionales "Si-sino", "Por siempre", "repetir" y "detener programa". |
| Sonido | Reproduce ficheros de audio y secuencias programables. | Sensores | Los objetos pueden interactuar con el ambiente que ha creado el usuario. |
| Lápiz | Control del ancho, color e intensidad del lápiz. | Operadores | Operadores matemáticos, generador aleatorio de números, sentencias "y" y "o" que comparan posiciones de los objetos. |
| Datos | Creación de variables y listas. Hay variables de la nube, pero aún no hay listas de nube. Se podrían implementar en la tercera versión de Scratch. | Más Bloques | Control de bloques y dispositivos externos. |

Figura 2.2: Tabla resumen con las categorías de Scratch

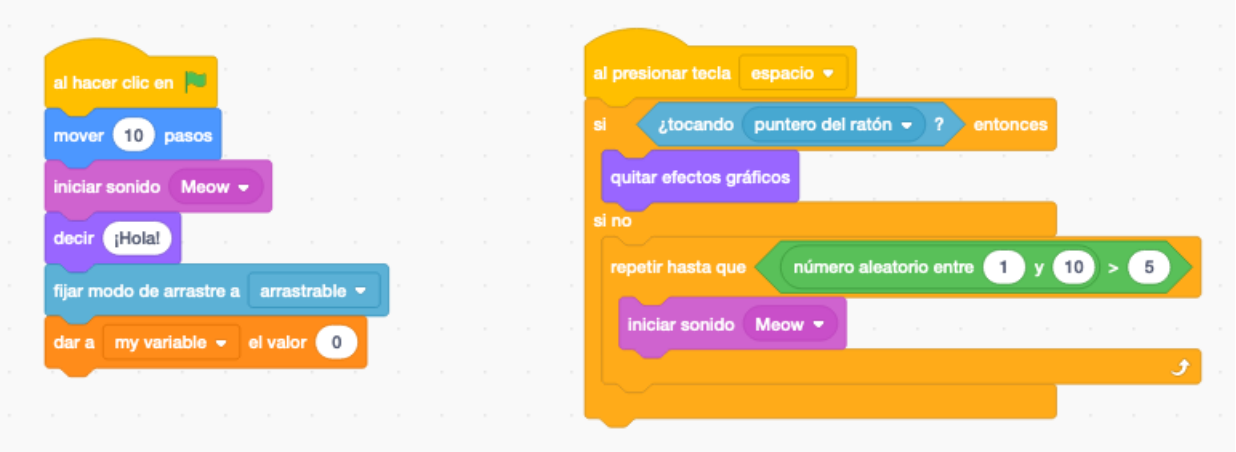



Figura 2.3: Dos ejemplos de secuencias de bloques en Scratch

el bloque morado, de lo contrario se ejecutará un bloque de tipo cíclico que solo terminará cuando se cumpla la condición del bloque verde. Todos los bloques están asociados al personaje de la Figura 2.1, pero se pueden agregar más personajes y escenarios que tendrán sus propios espacios de trabajo.

### 2.1.2. Descargable

En la subsección anterior se describió la interfaz gráfica donde los alumnos de los cursos de Scratch crean sus proyectos. Para entregarlos al equipo docente, deben utilizar la herramienta de exportación provista por el servicio, la cual genera un archivo en formato .sb3 (o .sb2 en sus versiones previas).

El archivo exportado, que puede convertirse manualmente a .zip para ver su contenido, contiene todos las imagenes y archivos de audio utilizados en el proyecto. Además, incluye un archivo llamado “proyect.json” que contiene la codificación en formato Json de todos los bloques y ajustes del proyecto. Para el proyecto utilizado como ejemplo en la sección anterior se generan los archivos de la Figura 2.4 donde se visualizan archivos de audio, SVG y Json.

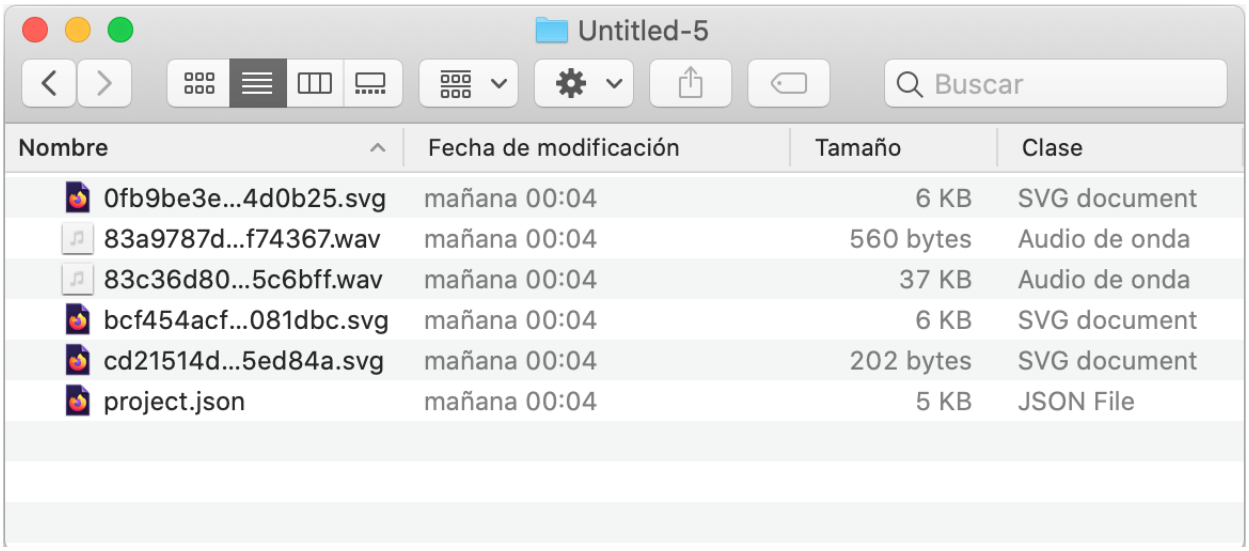



Figura 2.4: Contenido de archivo .sb3 para proyecto de ejemplo

En el Código 2.1 se muestra el primer nivel de "project.json", el cual contiene 4 llaves. "Targets" contiene una lista con información de todos los personajes, escenarios y bloques del proyecto. El resto de las llaves son configuraciones irrelevantes para este trabajo de título.

La lista "targets" contiene un diccionario para cada elemento del programa y ese diccionario contiene toda la información asociada a ese elemento. Por ejemplo, en la llave "1" de la figura, corresponde al personaje principal del proyecto creado en la sección anterior. Esto se puede identificar por la llave "name" que indica su nombre y la llave "isStage" que indica que no se trata de un escenario. La llave "blocks" contiene un nuevo diccionario de todos los bloques asociados al persona "Sprite1". El resto de las llaves no son utilizadas en presente trabajo de título.

```
{
"targets": [
    "0":{...},
    "1": {
      "isStage": false,
      "name": "Sprite1",
      "variables": {},
      "lists": {},
      "broadcasts": {},
      "blocks": {...},
      "comments": {},
      "currentCostume": 0,
      "costumes": [...],
      "sounds": [...],
      "volume": 100,
      "layerOrder": 1,
      "visible": true,
      "x": 20,
      "y": 0,
      "size": 100,
      "direction": 90,
```

```
      "draggable": true,
      "rotationStyle": "all around"
    },
    ]
"monitors: [...],
"extensions": [...],
"meta": {...}
}
```

Código 2.1: Extracto del archivo “project.json”

En el Código 2.2 se muestra el diccionario generado para el bloque de tipo “Evento” para cuando se acciona la bandera (“When flag clicked”) presente en la secuencia de bloques izquierda de la Figura 2.3. En él, se observa que cada bloque es identificado con un hash único (llave) y que contiene información sobre el tipo de bloque y su función específica (llave “opcode”), el bloque siguiente (“next”) y el bloque anterior (“parent”).

```
"blocks": {
        "N!%}4tQ@,:jXea=:GH!7": {
          "opcode": "event_whenflagclicked",
          "next": ")7G_L](k[kUxacvnU6NS",
          "parent": null,
          "inputs": {},
          "fields": {},
          "shadow": false,
          "topLevel": true,
          "x": 333,
          "y": 169
        },
        ...
        }
```

Código 2.2: Extracto del archivo “project.json” para bloque “When flag clicked”

Por otro lado, en el Código 2.3 se muestra el diccionario generado para el bloque condicional (if-else) de la secuencia de bloques del lado derecho de la Figura 2.3. En este se observa que no sigue la misma estructura lineal padre-hijo del código anterior, siendo la llave “next” del diccionario nula. Esto ocurre pues en realidad no hay más bloques fuera del condicional, de existir un bloque extra adjunto al final de la secuencia, sería su hash el que aparecería como valor de la llave “next”. Para este tipo de bloques, los bloques anidados en él se encuentran en la llave “input” que tiene como valor un diccionario con los bloques asociados a la condición (“CONDITION”), los bloques a ejecutarse si se cumple (“SUBSTACK”) y los bloques a ejecutarse si no se cumple (“SUBSTACK2”). El código completo de “project.json” se puede encontrar en el Apéndice B.1.

```
 ";W(xESf15Ei|!msEa^?p": {
          "opcode": "control_if_else",
          "next": null,
          "parent": ".L[~cJ4?8M:+A:+:uDJ0",
          "inputs": {
            "CONDITION": [
```

```
            2,
            "Ia`T],6KZ!jzdh.h~.O;"
          ],
          "SUBSTACK": [
            2,
            "~jjTN-s9:z*:%.+Fk*A^"
          ],
          "SUBSTACK2": [
            2,
            "fL+Wr.h]}$9!1Mh(~(mr"
          ]
        }
```

Código 2.3: Extracto del archivo "project.json" para bloque "if else"

## 2.2. Sistema de evaluación actual

Como se mencionó en la introducción, en los cursos de Scratch del DCC se utiliza un sistema mixto para evaluar y analizar los proyectos. Este se compone de una herramienta comercial de uso gratuito llamada "Dr. Scratch" y la evaluación manual utilizando una rúbrica diseñada por los investigadores a cargo del curso. A lo largo de esta sección se explicará el funcionamiento de ambas herramientas exponiendo sus ventajas y desventajas.

### 2.2.1. Dr. Scratch

En el área de análisis automatizado de programas en Scratch, Dr. Scratch se presenta como la principal herramienta comercial de uso gratuito [3]. Esta aplicación web permite a docentes y estudiantes automatizar el análisis de sus proyectos para comprobar si han sido programados correctamente, identificar ciertos errores generales, recibir retroalimentación para mejorar el código y, mediante puntuaciones y medallas, servir de estímulo para incentivar la mejora de habilidades de los programadores. En la Figura 2.5 se observa la página de inicio de la plataforma, disponible en `http://www.drscratch.org`, donde se permite cargar un proyecto tanto por su enlace a la página de Scratch o subiendo el archivo .sb3.

Una vez cargado el proyecto, Dr. Scratch lo analiza utilizando 4 niveles de competencia (Ninguno, Principiante, Desarollador y Competente) para los siguientes ítems: paralelismo, pensamiento lógico, control de flujo, interactividad con el usuario, representación de la información, abstracción, y sincronización. En base a las categorías anteriormente descritas, se asigna puntaje a cada una de ellas (máximo 3 puntos por categoría correspondientes a los niveles mencionados anteriormente) sumando un máximo de 21 un puntos. Además, reporta si existen malas prácticas de programación tales como código inalcanzable, mal uso de los objetos, entre otras.

En la Figura 2.6 se muestran los resultados obtenidos al cargar el proyecto de ejemplo con el que se trabajó en la sección anterior. En la interfaz se observan 3 secciones. En la primera (arriba a la izquierda) se muestra el puntaje obtenido por el programa (en este caso, 10/21). En la sección de la derecha se muestra el puntaje desglosado por categoría. La descripción de lo evaluado en cada una es accesible clickeando el nombre de la categoría. En la esquina

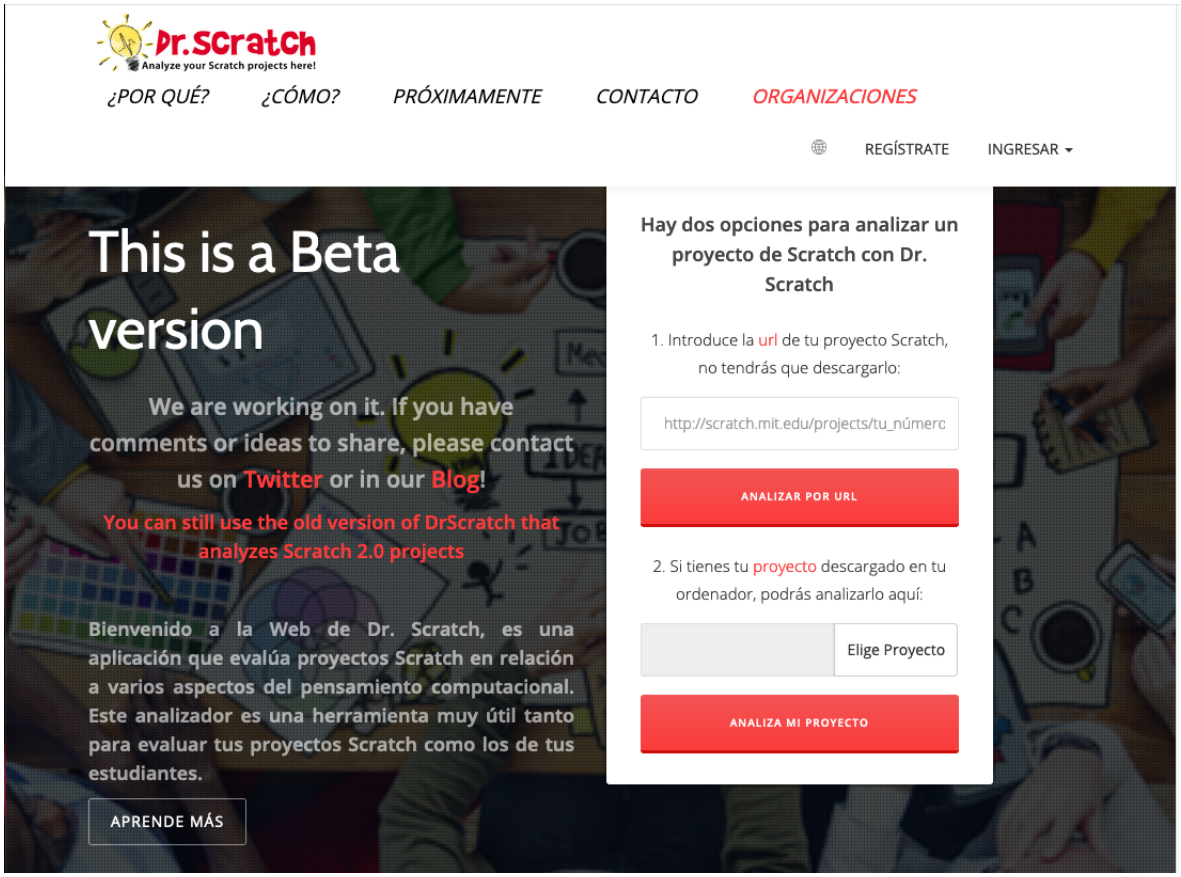



Figura 2.5: Página de inicio de Dr. Scratch

inferior izquierda se listan los malos hábitos de programación (*code smells*) presentes. Dada la simplicidad del proyecto analizado, el no haber obtenido buenos resultados es esperable.

En la Figura 2.7, se crea un proyecto más complejo, disponible en `https://scratch.mit.edu/projects/383615942/`, con 3 personajes y movimientos coordinados inspirado en *Flappy Bird*[1]. Como se observa en la Figura 2.8, al analizarlo en Dr. Scratch se obtiene un mayor puntaje, pero solo por 2 puntos. Al no existir mayor retroalimentación sobre cómo se asignan los puntos al programa en específico (solo existe una descripción general), no es fácil determinar qué considera la plataforma como "mejor" o "peor" programa, pues a pesar de la creación de un programa al menos 3 veces más grande y complejo, no se ve necesariamente reflejado en los resultados.

De los experimentos anteriores se puede concluir que, si bien Dr. Scratch es una buena herramienta para conocer los elementos que están presentes en un programa, carece de la transparencia necesaria para que el usuario pueda determinar si dichos elementos eran necesarios para un correcto funcionamiento o no. El asignar puntajes en un rango pequeño descontextualiza la real magnitud del proyecto, pues un programa con 2 personajes podría tener una calificación muy similar a un programa 10 veces más complejo. Esta misma descontextualización puede conducir a que un programa de baja calidad termine con una alta puntuación solo por tener instrucciones de alta complejidad pero sin mucha coherencia en

[1] `https://es.wikipedia.org/wiki/Flappy_Bird`

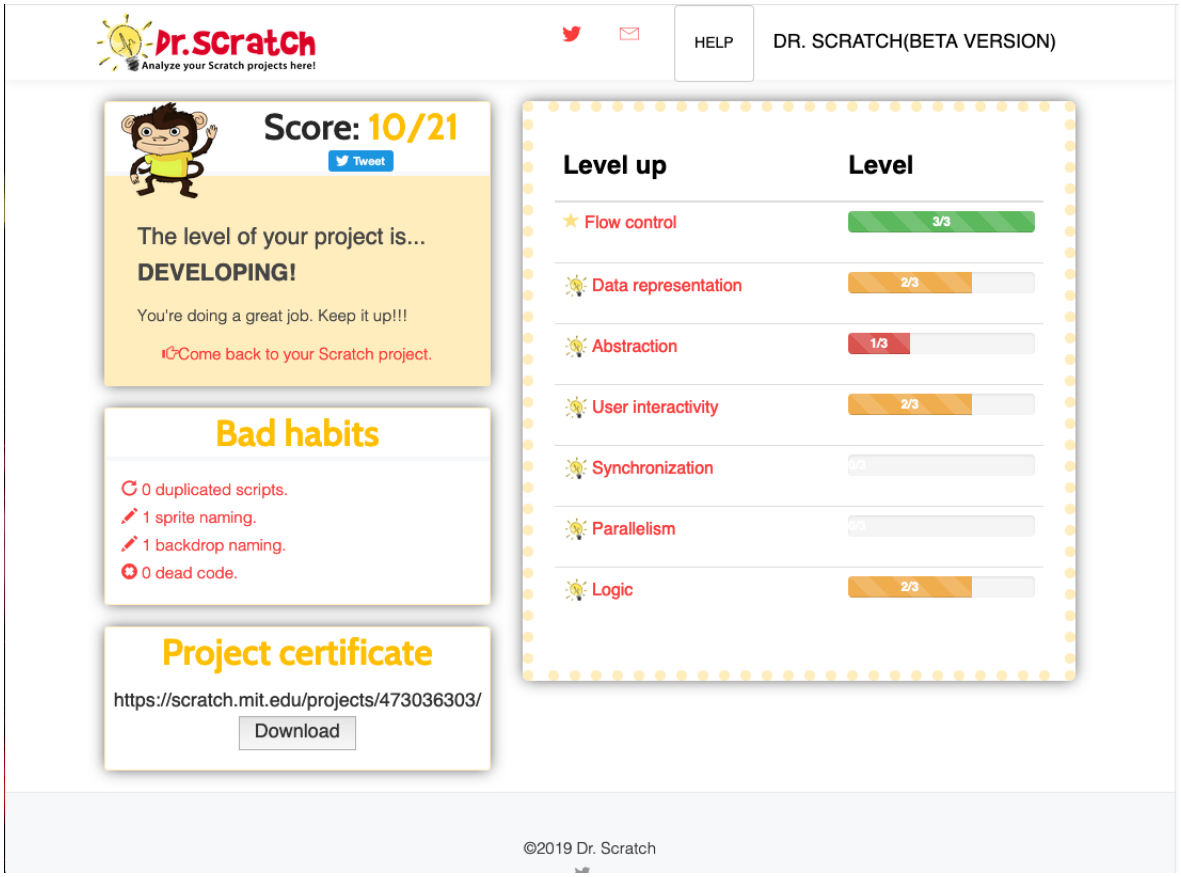



Figura 2.6: Análisis del programa mostrado en la Figura 2.3 en Dr. Scratch

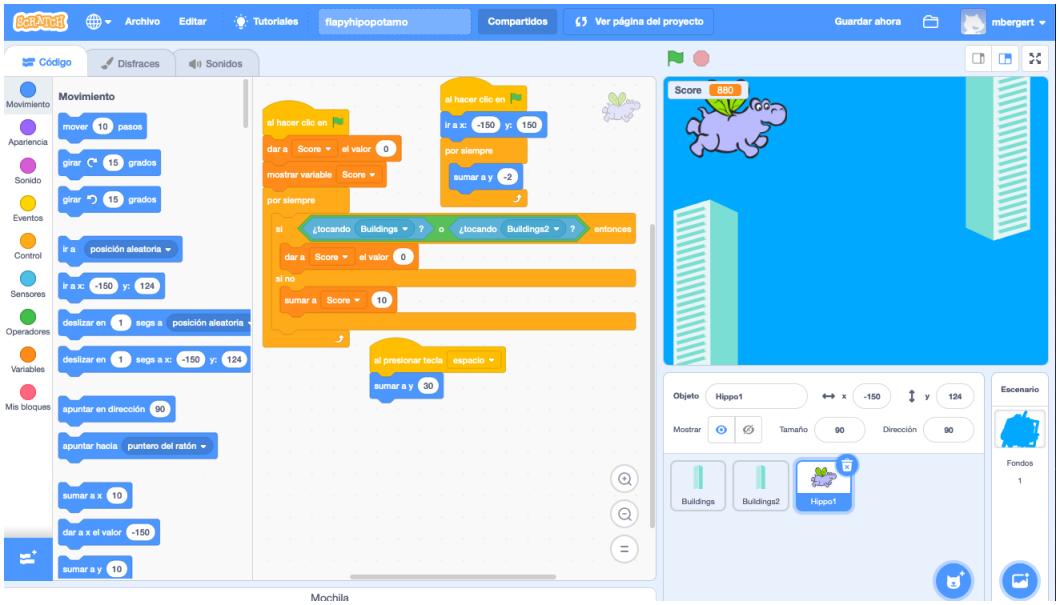



Figura 2.7: Juego más complejo en Scratch

relación al programa y viceversa.

### 2.2.2. Rúbrica de evaluación

Como complemento a Dr.Scratch, los docentes diseñaron una rúbrica de evaluación que permite analizar los proyectos de forma manual. Esta herramienta analiza bajo los mismos 4 niveles de competencia mencionados en la subsección anterior las categorías presentadas en la Figura 2.9.

Se observa que también se utiliza un sistema de 3 puntos por categoría como en Dr. Scratch, sin embargo, esto no genera las mismas limitaciones mencionadas. Una persona, a diferencia de una máquina, puede tener una visión global de un proyecto y definir su nivel de competencia con respecto a su totalidad y no el número de bloques de un tipo determinado posicionados de una forma específica.

Aunque certera, este tipo de evaluación también posee limitaciones que reducen su efectividad. En primer lugar requiere personas reales revisando cada proyecto individualmente, estas horas de trabajo son un recurso limitado que no siempre está disponible. Por otro lado, aunque se tuviesen horas de trabajo ilimitadas destinadas al análisis de los proyectos, cada revisión individual (que debe ser posteriomente tabulada) conlleva bastante tiempo en comparación a los segundos que tarda una herramienta automatizada, por lo que a mayor volumen de alumnos, se vuelve poco manejable su implementación. Por este motivo actualmente existe un retraso de aproximadamente 2 años de correcciones de proyectos, perdiendo información valiosa que podría ser utilizada para la investigación. Finalmente, los seres humanos son subjetivos y tienen distintas concepciones de lo que se considera aceptable, por lo que pueden existir discordancias entre evaluadores.

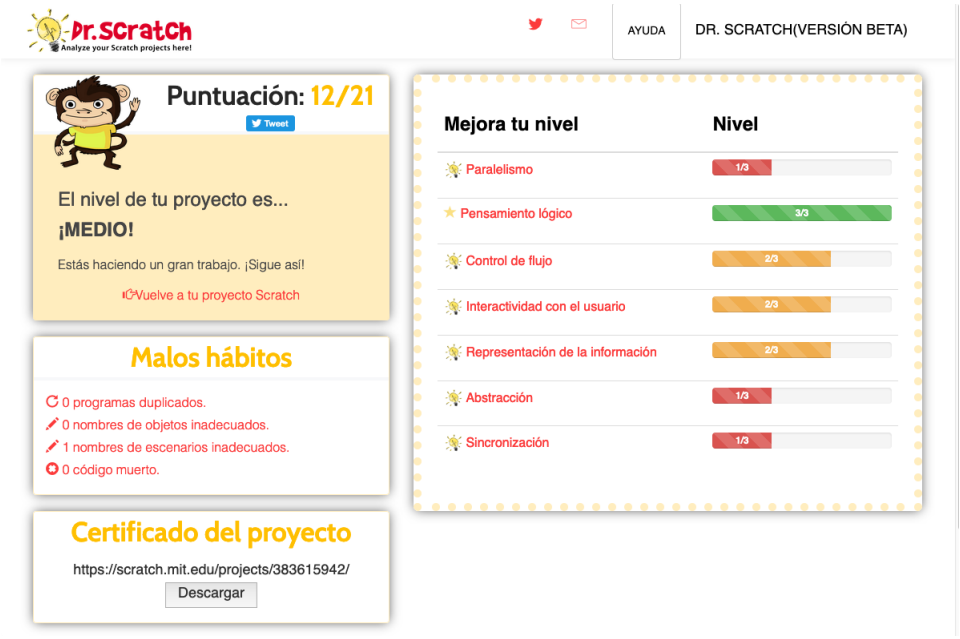



Figura 2.8: Análisis de juego más complejo en Dr. Scratch

**Table 1: Rubric for evaluating student projects**

| | None (0) | Beginner (1) | Developing (2) | Competent (3) |
|---|---|---|---|---|
| (A) % of sprites that only control themselves | [0%, 10%] | (10%, 40%] | (40%, 70%] | (70%, 100%] |
| (B) % of blocks that have a single purpose | [0%, 10%] | (10%, 40%] | (40%, 70%] | (70%, 100%] |
| (C) documentation | 0 comments | 1-2 comments | 3-6 comments | > 6 comments |
| (D) % of reachable blocks | [0%, 10%] | (10%, 40%] | (40%, 70%] | (70%, 100%] |
| (E) % of items with appropriate names | [0%, 10%] | (10%, 40%] | (40%, 70%] | (70%, 100%] |
| (F) % of superficial changes w.r.t sample projects | (70%, 100%] | (40%, 70%] | (10%, 40%] | [0%, 10%] |
| (G) project novelty | the expected behavior is not clear | quite similar to a sample project | extends a sample project in a novel way | quite different from the sample projects, or integrates 2 or more sample projects |

Figura 2.9: Rúbrica de evaluación utilizada en los cursos de Scratch del DCC [9]

## 2.3. Métricas y análisis relevantes

Fuera del mundo específico de Scratch, existen diversas métricas y tipos de análisis de programas que podrían utilizarse en una potencial herramienta evaluativa. Estas se aplican principalmente sobre programas representados en Grafos de Control de Flujo[2] (CFG por sus siglas en inglés). En la Figura 2.10 se muestran algunas representaciones de CFGs para instrucciones computacionales que crean nuevos caminos de ejecución.

Una de las métricas existentes es la Complejidad Ciclomática [2], la cual mide de forma cuantitativa el número de caminos linealmente independientes presentes en un CFG, es decir, busca contar los caminos forma que cada uno tenga al menos una arista que no está presente en ningún otro.

[2]Representación, en forma de grafo dirigido, de todos los caminos que pueden ser atravesados a través de un programa durante su ejecución.

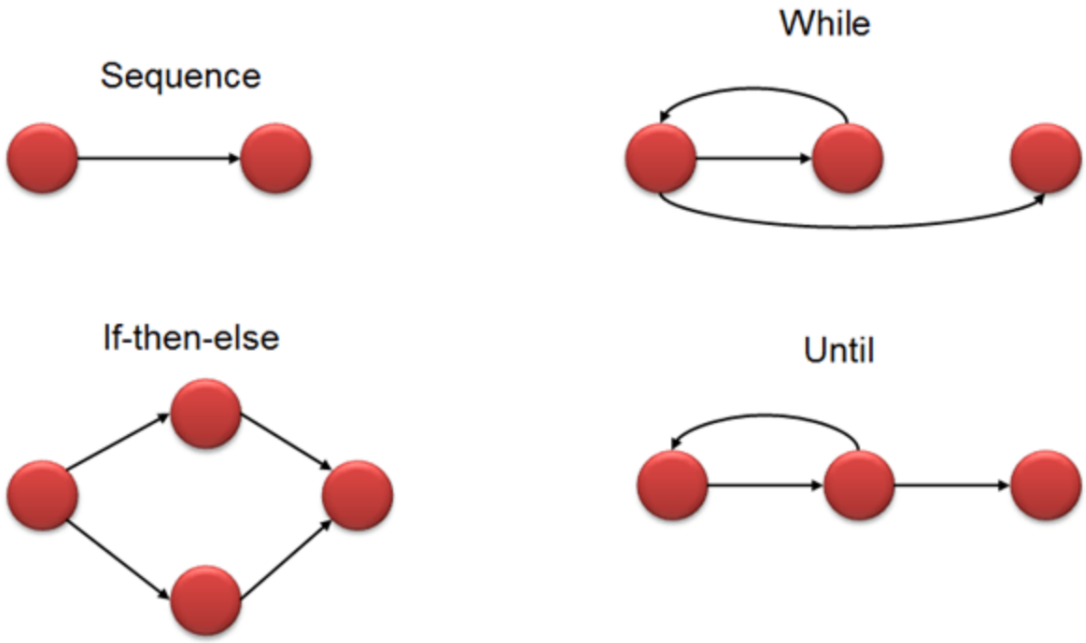



Figura 2.10: Ejemplos de Grafos de Control de Flujo para distintas condiciones que podrían encontrarse en un programa

Esto es útil pues permite, fácilmente, cuantificar la cantidad de condicionales de un programa sin que esto se limite a únicamente contar la cantidad de *ifs* presentes en él. Pues, la Complejidad Ciclomática, al contabilizar los posibles caminos, asigna la misma complejidad a un programa con varias instrucciones condicionales independientes y a un programa con una instrucción condicional compleja. La métrica resuelve el problema presentado anteriormente, en el que un programa con una instrucción condicional compleja tiene un puntaje mucho mayor a un programa con dos instrucciones condicionales simples, a pesar de tener la misma cantidad de caminos posibles.

En lo que a análisis respecta, existen los llamados **"Análisis de flujo de datos"** (*Data Flow Analysis*) [1]. En ellos, se deriva información sobre el comportamiento dinámico de un programa basándose en el código estático. Son sensibles al flujo de las funciones (o en el caso de Scratch, los *scripts*) y se subdividen en diversas aplicaciones, algunas de ellas son:

1. **Análisis de vida de las variables** (*Liveness Analysis*): Determina que una variable está "viva" en un determinado punto del programa si su variable es usada al menos una vez en el futuro. La motivación detrás de este análisis es determinar la existencia de código muerto (variables alocadas pero nunca utilizadas.
2. **Expresión disponible**(*Available Expression*): Una expresión $(x + y)$ está disponible en un nodo $n$ si cada camino desde el nodo inicial a n evalúa $x + y$ y las variables $x$ e $y$ no se redefinen después. La motivación de este análisis consiste en que si una expresión está disponible en el punto donde se evalúa, no necesita ser recalculada.
3. **Alcance de definición***(Reaching definition)*: Una definición de de variable $v$ alcanza un nodo $n$ si existe un camino desde d hasta $n$ tal que $v$ no se redefine.

La utilidad *Data Flow Analysis* en el proyecto, es que si se extiende a las lógicas de Scratch, permitirá identificar errores en los *scripts* que no se ven a simple vista. En el caso del juego de la Figura 2.7, por ejemplo, ayudaría a identificar casos como que no se incluyera el bloque de actualización de puntaje o que este se recalculara más veces de las necesarias.

Estos fueron los análisis definidos en primera instancia para el proyecto y de los cuales se esperaba fueran la piedra angular del trabajo de título. Sin embargo, debido a los problemas descritos en la subsección anterior y la necesidad de implementar un parser, quedaron fuera de alcance dada su complejidad. Si bien se implementaron métodos encargados de generar Grafos de Control de Flujo para cada secuencia de bloques, estos cuentan con limitaciones importantes por lo que no se considera como una tarea lograda exitosamente. Se pretende que incorporarlos sean uno de los primeros pasos a seguir a la hora de implementar las propuestas de trabajo futuro para la plataforma.

## 2.4. Otras herramientas comerciales o complementarias

Durante el trabajo investigativo previo a esta memoria de título, se investigaron diversas herramientas que pudieran complementar el proceso de conversión de datos y su análisis. En un inicio se tenía la esperanza de incorporarlas todas al proyecto, agregando valor y robustez al producto. Sin embargo, por distintos motivos que serán detallados para cada herramienta específica, no pudieron ser incorporadas al producto final aunque no se descarta reevaluar estas decisiones en el futuro.

### 2.4.1. Hairball

Hairball, según su definición oficial, es un sistema automatizado que puede ser utilizado tanto por un estudiante para señalar posibles errores o malas prácticas como por un calificador para ayudar a inspeccionar la implementación de los programas Scratch. Debido a que el análisis automático no podrá determinar el efecto sensorial y visual de los programas, Hairball se enfoca en la implementación, incluyendo prácticas de programación seguras/robustas [7].

La librería ofrece siete análisis en su repositorio (`https://github.com/ucsb-cs-education/hairball`), disponibles en la Figura 2.11. Algunos de ellos fueron contemplados en el proyecto, por lo que se esperaba que utilizarlos como validación de resultados, mientras que otros que quedaban fuera de alcance hubiesen servido como complemento.

En primera instancia, sí se tenía contemplado utilizar Haiball como librería dentro del proyecto con la finalidad de dar una perspectiva más completa sobre un proyecto. Sin embargo, el código de uso libre del repositorio, que fue actualizado por última vez hace 7 años, solo contaba con soporte hasta su segunda versión (.sb2). Cuando se tomó la decisión de desarrollar el parser para Scratch 3.0 debido a su retrocompatibilidad con versiones anteriores, el uso de de esta librería quedó fuera del alcance del proyecto.

Cuando Scratch migró a la versión 3.0, Hairball fue solo uno de los plugins disponibles en volverse incompatible. Debido a esto, en una investigación posterior, se encontraron múltiples APIs de uso público dedicadas a convertir proyectos en formato .sb3 a .sb2, por lo que no se descarta incorporar una de ellas a futuras iteraciones del proyecto y de esta forma poder

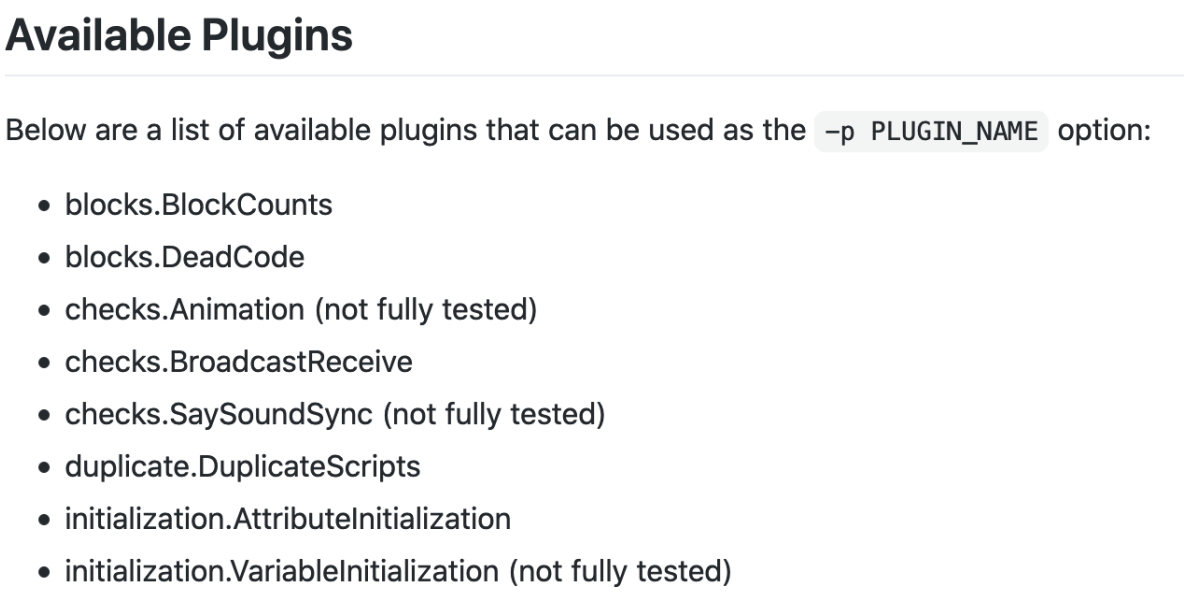



Figura 2.11: Análisis ofrecidos en el repositorio de Hairball

complementar los análisis actuales con los ofrecidos por Hairball.

### 2.4.2. Mulang

Mulang fue una de las herramientas contempladas como parser de datos. En su descripción oficial se define como una herramienta para analizar código fuente, que se basa en cinco componentes principales [4]:

1. Un árbol semántico abstracto, un lenguaje intermedio que permite expresar la estructura semántica, en lugar de sintáctica, de un programa de paradigmas múltiples.
2. Un conjunto de más de 90 inspecciones para consultar código, consultar código explícitamente (expectativas) o implícitamente (*Code Smells*).
3. Un lenguaje de definición de expectativas (Expectations Definition Language ó EDL), un lenguaje para definir expectativas personalizadas.
4. Una herramienta de línea de comandos para analizar tanto el código fuente en muchos lenguajes como el AST de Mulang. Esta herramienta se distribuye como un binario linux-amd64 y un paquete de JavaScript.
5. Interfaces de nivel superior en Ruby y Javascript que son más fáciles de usar y proporcionan algunas capacidades adicionales, como el análisis de expectativas y la humanización internacional automatizada.

En pocas palabras, esta herramienta toma el código en uno de los lenguajes soportados, y lo transforma en un AST (Árbol de sintaxis abstracta). Este árbol se puede consultar para obtener distintos análisis ya sean propios de Mulang (punto 2) o personalizadas (punto 3).

Al inicio del trabajo de título, se esperaba que toda la funcionalidad del parser fuese cubierta por esta herramienta y el enfoque del proyecto estuviese centrado en los distintos análisis y visualizaciones que se pudiesen obtener a partir del árbol semántico generado. De

esta forma, se podrían haber diseñado análisis mucho más complejos de los que finalmente se lograron.

Debido a la casi nula documentación y ejemplos disponibles para el uso de esta herramienta nunca fue posible generar un árbol semántico a partir de un programa en Scratch a pesar de seguir todas las instrucciones indicadas en la página oficial. Si bien se logró crear un AST para un programa de ejemplo en Python, los resultados eran difíciles de comprender y en general toda la semántica del lenguaje no era intuitiva. Esto incrementó de sobremanera el riesgo del proyecto, pues si se dedicaba tiempo al aprendizaje de la herramienta, al no existir seguridad de si los resultados entregados serían útiles o incluso correctos podrían perderse semanas valiosas de trabajo para el desarrollo del producto.

La pérdida de esta herramienta volvió necesaria la reestructuración del trabajo de título, incluyendo el diseño e implementación de un parser propio para los datos. Debido a esto, de momento no se contempla incluir Mulang en iteraciones futuras de la plataforma.

## 2.5. Resumen

En esta sección se discutieron las principales componentes de Scratch y su composición interna, haciendo hincapíe en las estructuras relevantes para el trabajo de título. Además, se detallaron las herramientas utilizadas en el proceso de evaluación actual utilizado en los cursos de Scratch dictados por el DCC. Finalmente, se describieron todas las herramientas investigadas previo al inicio de la memoria, el por qué no se incorporaron y si se espera que sean incluidas en iteraciones futuras del proyecto.

# Capítulo 3

# Análisis y diseño de la Solución

Como se dejó ver en los capítulos anteriores, el problema que se busca resolver es el análisis semi-automatizado de de programas desarrollados en el contexto específico de los cursos de Scratch ofrecidos por el Departamento de Ciencias de la Computación de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile.

En el proceso actual se identificaron dificultades como el tiempo requerido a los docentes para procesar y tabular manualmente los proyectos, la poca transparencia de las herramientas comerciales disponibles y la necesidad de unificar en un solo lugar toda la información disponible de cada proyecto.

Dentro de los principales desafíos a resolver se encuentran: La inexistencia de un parser que interprete el código tras los bloques de Scratch y lo transforme a algún lenguaje de programación tradicional, el análisis de los elementos del programa calculando de forma que entregue datos valiosos para la investigación y la incorporación de la rúbrica de evaluación para mantener consistencia entre los datos de cursos previos y actuales.

Para solucionar estos problemas, primero, en la Sección 3.1, se mencionarán cuáles son las principales historias de usuario que el sistema propuesto debe cumplir. Posteriormente en la Sección 3.2, se explica la solución planteada en el desarrollo de este trabajo, cómo aborda las historias y requisitos planteados, y cuáles son los puntos importantes considerados en el diseño expuesto.

## 3.1. Definición del problema

Para definir y delimitar de mejor manera el problema, se definen historias de usuario que narran lo que los usuarios esperan del sistema. Si bien existen más historias de usuario necesarias para la completitud del sistema, estas son transversales a cualquier sistema de esta naturaleza, por lo que solo se detallan aquellas que son distintivas del proyecto.

Los usuarios de estas historias son los docentes de los cursos de Scratch del DCC o posibles ayudantes puedan incorporar a los cursos con la función de llevar a cabo las evaluaciones. El problema inicial está pensado como un programa local y no interconectado entre distintos

revisores.

**HU-1 : Crear un Curso**

Un usuario podrá crear un curso con su semestre y año.

**HU-2 : Corregir un proyecto**

Un usuario podrá subir un proyecto a la plataforma en formato de .zip o .sb3 para ejecutar los análisis.

**HU-3: Corregir proyectos masivamente**

Un usuario podrá cargar masivamente varios proyectos a la vez en forma de carpeta comprimida (.zip) con todos los proyectos y ejecutar los análisis. Los análisis se ejecutaran individualmente para para cada proyecto.

**HU-4: Visualización de proyectos**

Un usuario podrá visualizar todos los proyectos que ha corregido previamente. Al abrir un proyecto se desplegarán los resultados de todos los análisis y la evaluación manual (rúbrica).

**HU-5: Evaluación manual**

Un usuario puede ver y editar la rúbrica de alguno de sus proyectos analizados. El usuario no puede editar los análisis automatizados.

**HU-6: Exportar rúbrica**

Un usuario puede exportar los datos de las rúbricas de todos los proyectos analizados de uno de sus cursos en formato .csv.

**HU-7: Exportar proyectos**

Un usuario puede exportar los proyectos que ha cargado previamente en la plataforma ya sea de forma individual o masiva. Estos serán descargados en su formato de origen.

## 3.2. Solución propuesta

Como se ha mencionado previamente en este capítulo, se identificaron desafíos o ejes principales que debe contemplar la solución propuesta al momento de resolver el problema. Para ello, se propone la división modular presentada en la Figura 3.1 y detallada a continuación.

1. **Procesamiento**: Implementación de un parser capaz de convertir los datos extraídos desde Scratch (Ver Código 2.1) a un lenguaje de programación tradicional para su análisis.
2. **Análisis y métricas**: Diseño de consultas o cálculos que permitan extraer la mayor información posible de los datos del programa.
3. **Resultados y reporte**: Generación de resultados visualizables que expongan de forma

transparente los análisis ejecutados que complementen la herramientas de corrección manual disponible.

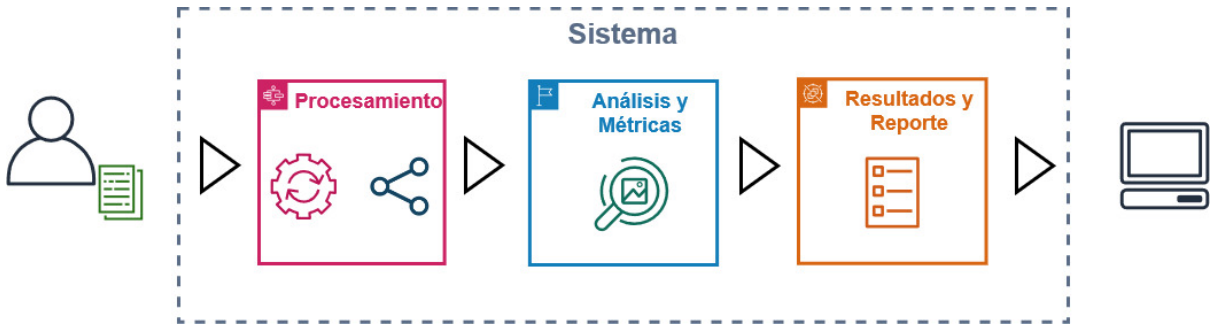



Figura 3.1: Diagrama de la solución propuesta

Para llevar al cabo el flujo efectivo de los componentes mencionados y respetando las interacciones descritas en las Historias de Usuario y la Figura 3.1, se propone el diseño e implementación de una aplicación web que permita a los docentes unificar su sistema de evaluación y reduzca los tiempos dedicados a tabulación de resultados.

En las siguientes subsecciones se explica la solución propuesta dividida en las distintas componentes que forman el sistema: el modelo de datos, las interacciones disponibles y las distintas interfaces web.

### 3.2.1. Modelo de datos

Dada la necesidad de guardar la información de los proyectos por un periodo indeterminado de tiempo y considerando que reprocesar cada vez todos los datos podría ser insostenible ante grandes volúmenes, se determina que la estrategia más adecuada es intentar asociar cada secuencia de bloques (Se entiende como secuencia cada proceso independiente dentro del programa, en la Figura 2.3 del Capítulo 2 se muestran 2 secuencias) a la estructura de un Grafo de Control de Flujo[1] como los de la Figura 3.2 donde cada bloque corresponde a un Nodo y las interacciones entre ellos serán definidas por las aristas.

Debido a lo anterior, se define un modelo de la Figura 3.3 que además de contener entidades básicas para el funcionamiento de una plataforma web, permite soportar la creación de estas estructuras de forma consistente de forma que, junto con identificar las secuencias de bloques, existan las relaciones necesarias para determinar a qué personaje o escenario pertenecen, dentro de qué proyecto se encuentran y asociadas a qué curso. Al mismo tiempo, que permita almacenar información sobre interacciones entre dos o más secuencias de bloques y que todo lo anterior esté conectado a la rúbrica de evaluación tal que se mantenga la cohesión de cada proyecto.

En la siguiente lista se describen brevemente cada una de las entidades de la Figura 3.3:

- **Course**: Instancia de un curso definida por su semestre y año. Es la estructura padre, directa o indirectamente, de todas las demás entidades del modelo. Permite organizar las evaluaciones.

[1] `https://en.wikipedia.org/wiki/Control-flow_graph`

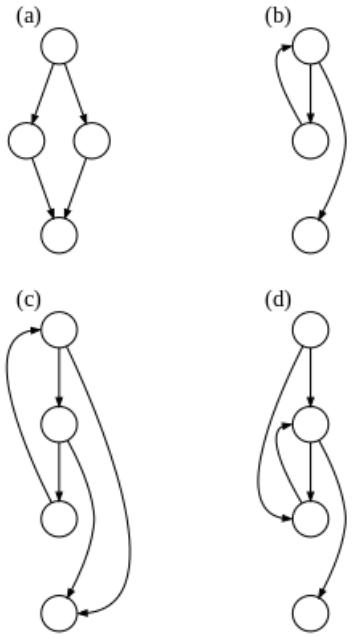



Figura 3.2: Algunos ejemplos de CFG: (a) un if-then-else. (b) un bucle while. (c) un bucle natural con dos salidas. (d) un CFG irreducible: un bucle con dos puntos de entrada. Fuente: `https://en.wikipedia.org/wiki/Control-flow_graph`

- **Project**: Contiene toda la información relacionada a un proyecto, incluida la ruta para el archivo .sb3 y un identificador interno para mantener cohesión con el sistema de evaluación actual. Esta instancia debe pertenecer a un curso y ser borrada en caso de eliminarse dicho curso.
- **Rubric**: Contiene atributos para cada categoría de la rúbrica, donde se espera almacenar el nivel de completitud que alcancen en cada uno de ellos. Además, instancias para escribir comentarios en cada atributo si el evaluador así lo desease. Cada rúbrica debe estar vinculada a un proyecto y eliminarse en caso de borrar dicho proyecto.
- **Actor**: Contiene la información de un actor del programa, su finalidad es poder definir que bloques pertenecen a él.
- **Block**: Contiene la información de cada bloque individual del sistema, con atributos para identificar a que actor y proyecto pertenecen. Además de su información distintiva (que categoría y acción especifica realizan), almacena el nombre del bloque anterior y bloque siguiente. Corresponde a los nodos de un CFG. Estan asociados a un proyecto.
- **Arch**: Representa las conexiones entre 2 bloques y su tipo, fue pensada para describir las relaciones entre todos los bloques involucrados en un ciclo o instancia condicional.
- **Stage**: Lleva registro de los escenarios de un proyecto con propósito de contabilizarlos.
- **Graph**: Tabla que agrupa los nodos y arcos de cada grafo de un proyecto, asociandolos a su actor.

## 3.3. Análisis automatizados

En términos de análisis automatizados y tomando en cuenta la forma en la que se diseñó el modelo de datos, se espera dividirlos en 2 categorías presentadas en las subsecciones a

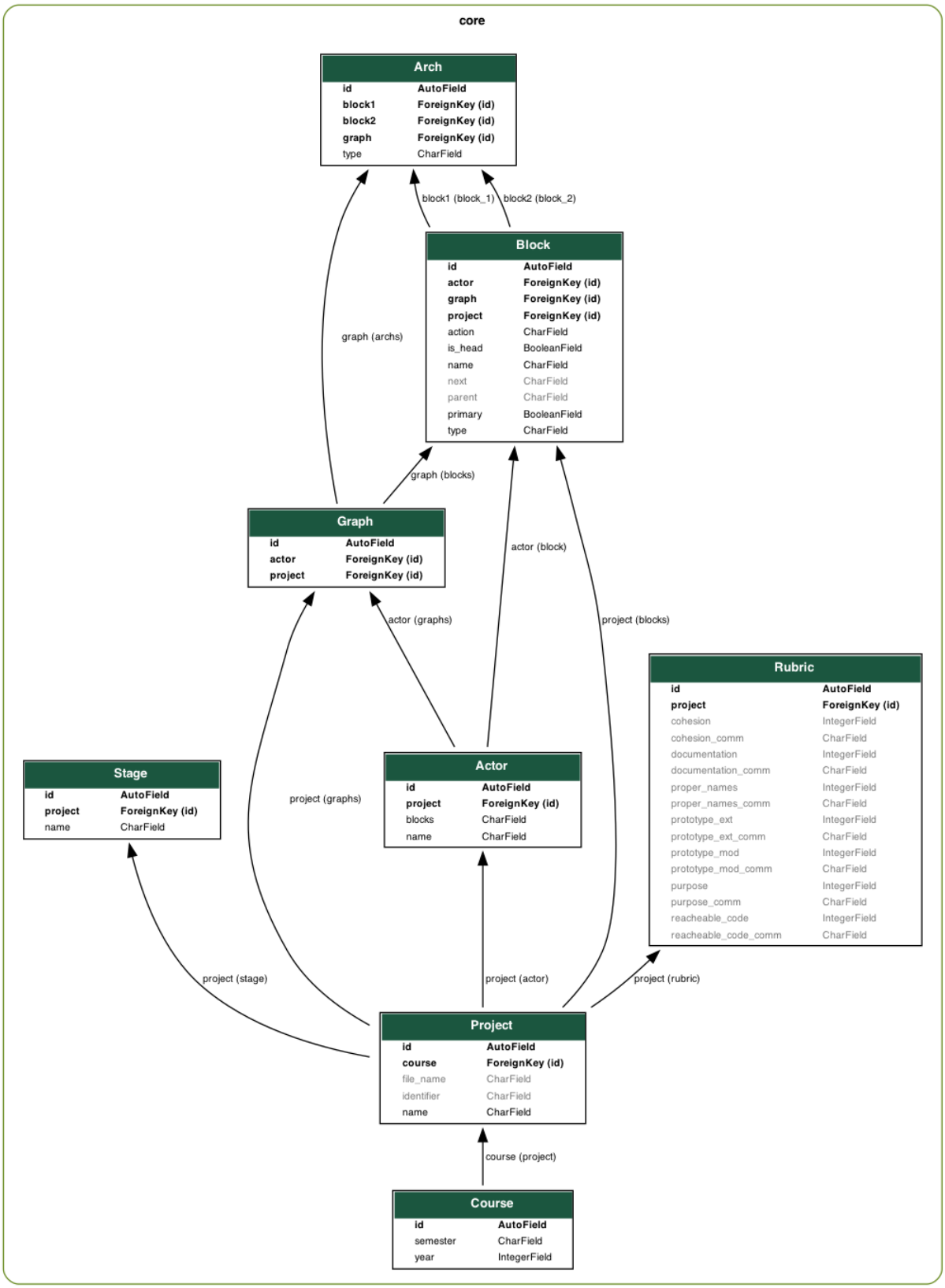



Figura 3.3: Diagrama del modelo de datos

continuación.

### 3.3.1. En base a consultas

Se busca contabilizar los diversos elementos presentes en un programa tal como actores, escenarios, eventos, interacciones, cantidad de bloques, etc., con la finalidad de representar la complejidad del proyecto en base a sus partes, sin especificar necesariamente el comportamiento especifico del mismo. Gracias a la forma en la que fue diseñado el modelo, se llevará a cabo utilizando consultas directas a la base de datos facilitadas por Django.

Como primer paso, se implementarán consultas que cuenten los bloques del programa, separándolos por actor y por tipo. A su vez, se contará el número de actores y escenarios, los grafos que estos generan y los nodos que componen cada grafo. Luego, a partir de estos datos, se implementarán funciones que recorran los bloques para identificar las interacciones entre los distintos actores del programa y también, aquellos bloques hayan quedado sin ejecutar (código muerto).

### 3.3.2. En base a grafos de control de flujo

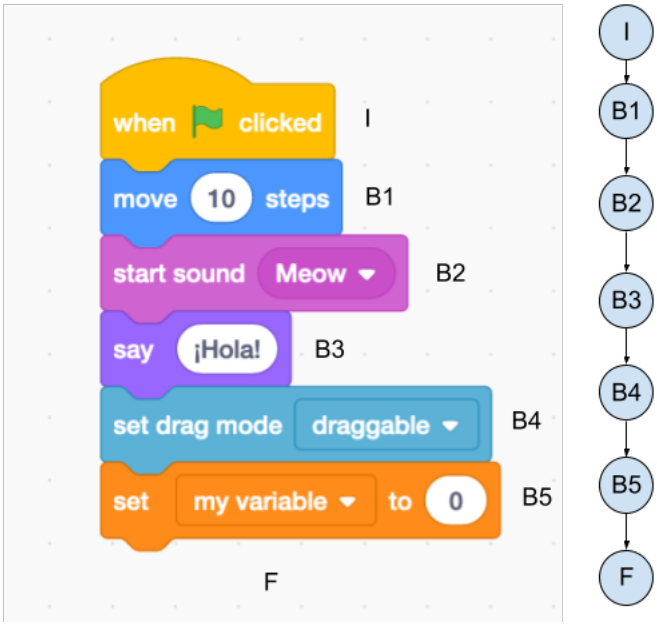
Se espera implementar una función que permita, a partir de los bloques y aristas que se guarden en la base de datos y apoyándose en el modelo Graph, generar grafos de control de flujo (CFGs) que se muestren en la ficha de cada proyecto y que faciliten la comprensión orgánica de la estructura del mismo. En la Figura 3.4 se muestra un ejemplo del tipo de transformación que se busca conseguir basándose en el programa de ejemplo mostrado en la Figura 2.3.

En la Figura 3.4a se muestra el CGF generado para una secuencia lineal y se observa como ambas poseen una estructura similar. Mientras que en el programa de Scratch se observa un bloque sobre otro, en el CGF se ve un nodo sobre otro, pero unidos por una arista. Los programas en Scratch no poseen un bloque formal de finalización de programa, por lo que en el CFG se representa con el nodo F.

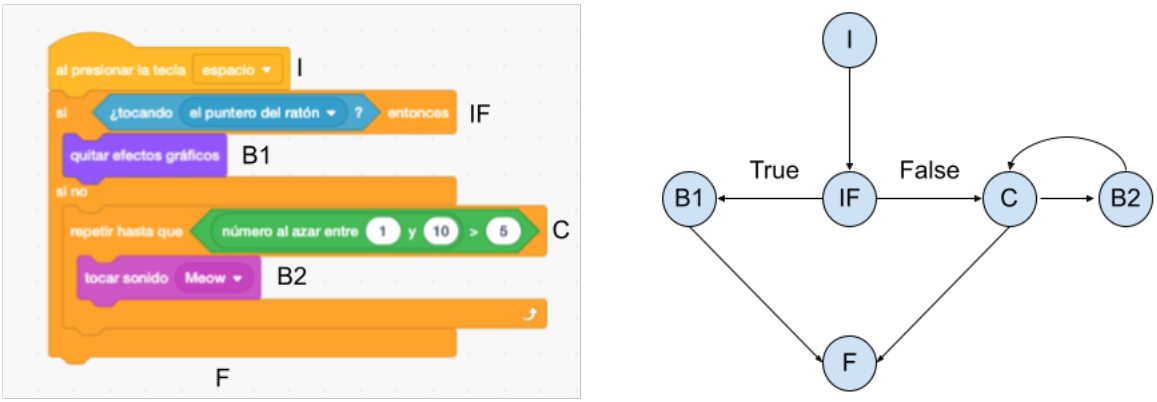
Por otro lado, en la Figura 3.4b, se muestra el CFG generado para una secuencia anidada. En él se observa que para cada anidación de bloques no condicionales, se crea una nueva bifurcación en grafo. Un ejemplo de esto es el bloque IF, donde se puede observar que en lugar de seguir un camino recto ahora el grafo posee dos caminos independientes entre sí. Equivalente es el caso para el bloque C.

Sobre estos grafos de espera poder aplicar fórmulas que permitan calcular características más técnicas del proyecto, por ejemplo, su complejidad ciclomática, el análisis de vida de sus variables, obtener un segundo cálculo sobre el código inalcanzable para contrastarlo con el obtenido a través de consultas a la base de datos, etc.

Como se mencionó en el Marco Teórico, los cálculos de variables y código inalcanzable quedaron fuera de alcance en el trabajo de título debido a la necesidad de implementar un parser. Sin embargo, se decidió modelar los CFGs de igual modo para, además mostrarlos y de calcular la complejidad ciclomática, dejar una base para incorporar el resto de las operaciones en el futuro.




(a) CFG correspondiente a secuencia lineal




(b) CFG correspondiente a secuencia anidada

Figura 3.4: Ejemplo CFGs generados a partir del programa de la Figura 2.3

Al obtener los CFGs se esperaba calcular su complejidad ciclomática utilizando la fórmula:

$$CYC = E - N + 2P$$

Donde E corresponde al número de aristas, N el numero de nodos y E el número de componentes disconexas del grafo. Por ejemplo, en la Figura 3.4b se tienen 6 nodos, 7 aristas y solo una componente. En este ejemplo se consideran ambas secuencias como independientes, por lo que cada una tiene su propia complejidad y solo un elemento conexo.

$$CYC = 7 - 6 + 2 * 1 = 3$$

Es decir, la complejidad ciclomática de ese bloque es 3, lo cual se comprueba al ver que presenta 3 caminos linealmente independientes que se pueden ejecutar: $(I - IF - B1 - F)$, $(I - IF - C - F)$ y $(I - F - C - B2 - C - F)$

Lamentablemente, en la sección de implementación se mostrará que debido a una inconsistencia en la generación de los CFGs no se pudo utilizar esta fórmula para calcular la complejidad ciclomática. Sin embargo, en la misma sección se propondrá una implementación alternativa con resultados consistentes para los casos de prueba.

## 3.4. API e interfaz web

Para incorporar al usuario al flujo de datos descrito, se diseño una Interfaz Web con la que este puede interactuar y visualizar. La interfaz se conecta a distintos endpoints, direcciones web que entregan o reciben contenido, que son expuestos por el sistema como una API, con los cuales se procesarán las acciones solicitadas por el usuario.

Los endpoints diseñados pueden dividirse en dos principales categorías: existen aquellos destinados a exponer información propia del sistema como lo son las distintas instancias existentes del modelo de datos. Los otros tienen como finalidad iniciar algún proceso dentro de la plataforma como lo son el análisis de proyectos, el cálculo de resultados o la exportación de datos.

### 3.4.1. Endpoints del sistema

En esta subsección se presentarán en detalle los endpoints más relevantes para un usuario. Existen otros endpoints de carácter informativo del estado del sistema y que no realizan cálculos adicionales, los cuales serán expuestos en su versión genérica y se pueden encontrar en completitud en el Apéndice C.

**Genéricos:**

1. **Creación**: Permite crear una nueva instancia en la base de datos para un modelo determinado.
2. **Visualización**: Permite visualizar las instancias existentes de un modelo en la base de datos.

3. **Eliminación**: Permite eliminar de la base de datos una instancia de un modelo.
4. **Edición**: Permite modificar uno o más atributos de una instancia de un modelo.

**Específicos:**

1. **Carga de proyecto(s) al sistema**: Permite al usuario cargar uno o más (en formato comprimido) proyectos a la plataforma y los convierte en instancias del modelo de datos.
2. **Análisis de proyecto(s)**: Analiza todas las instancias del modelo de datos asociadas a ese proyecto para calcular la mayor cantidad de información relevante posible y luego exponerlas.
3. **Exportación individual de un proyecto**: Permite al usuario descargar el archivo .sb3 asociado a un proyecto.
4. **Exportación masiva**: Permite al usuario descargar los archivos .sb3 de todos los proyectos de un curso.
5. **Exportación de rúbricas**: Permite al usuario descargar los datos de las rúbricas de todos los proyectos de un curso en formato .csv.
6. **Creación de CFGs**: Calcula los nodos y aristas de los CFGs presentes en un proyecto para luego exponerlos.

### 3.4.2. Interfaz de usuario

Se crearon mockups para aterrizar las funcionalidades de la plataforma de manera cohesiva y poder definirlas de mejor manera. Como se observa en sus títulos, inicialmente se había determinado la herramienta como "Corrector Scratch" o se hablaba de corrección en distintas instancias. Esto fue modificado durante el planteamiento definitivo de la solución pues la finalidad de la plataforma no es "corregir" o implicar que un proyecto es "incorrecto", sino analizar y extraer datos.

En la Figura 3.5 se muestra la vista *Inicio* donde los usuarios pueden ver los cursos que han creado y acceder a ellos, crear nuevos o borrar los existentes. Corresponde a la vista de acceso al sistema ya que los cursos son la figura jerárquica más alta del sistema y de la cual dependen el resto de los datos.

En la Figura 3.6 se muestra la vista *Curso*. A esta vista se accede desde la vista de *Inicio* y se divide en dos partes. En el lado izquierdo de muestra la carga de proyectos, donde se puede cargar uno o más proyectos para ser analizados por el sistema. En el lado derecho se muestra una tabla con todos los proyectos asociados a ese curso, con su nombre, su descripción y la posibilidad de exportarlos singular o masivamente.

En la Figura 3.7 se muestra la vista *Proyecto* la cual se accede al presionar un proyecto en la vista *Curso*. Esta vista, al igual que la anterior, también se divide en dos secciones esta vez superior e inferior. En la vista superior se observan los resultados de los análisis automatizados, por ahora abstractos, que se espera sean representados en forma de gráficos o tablas según sea necesario. En la vista inferior se muestra la rúbrica de corrección manual, donde el usuario puede ingresar los resultados para cada categoría y comentarios de ser

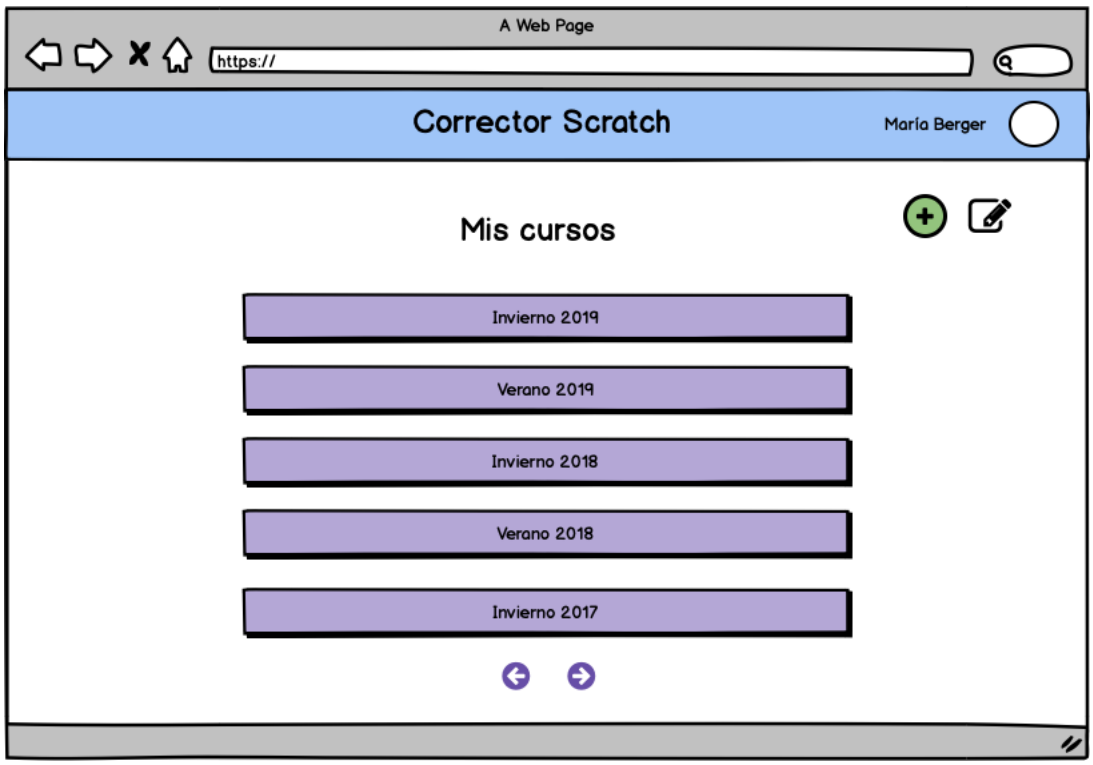



Figura 3.5: Vista de inicio

necesario. De esta forma se unifican ambos criterio de corrección (automatizado y manual) en una sola vista.

## 3.5. Resumen

Con las historias de usuario y el flujo de datos propuesto, se presentó un diseño del sistema a desarrollar que cumple con las características pedidas; en particular esta propuesta unifica en un solo lugar los pasos del proceso actual.

También se presentaron interfaces web del prototipo, que permitirán a los usuarios hacer uso del sistema propuesto. En los capítulos siguientes se discuten las decisiones tomadas en la implementación de esta propuesta, los resultados obtenidos, y la validación del sistema según los objetivos específicos de esta memoria.

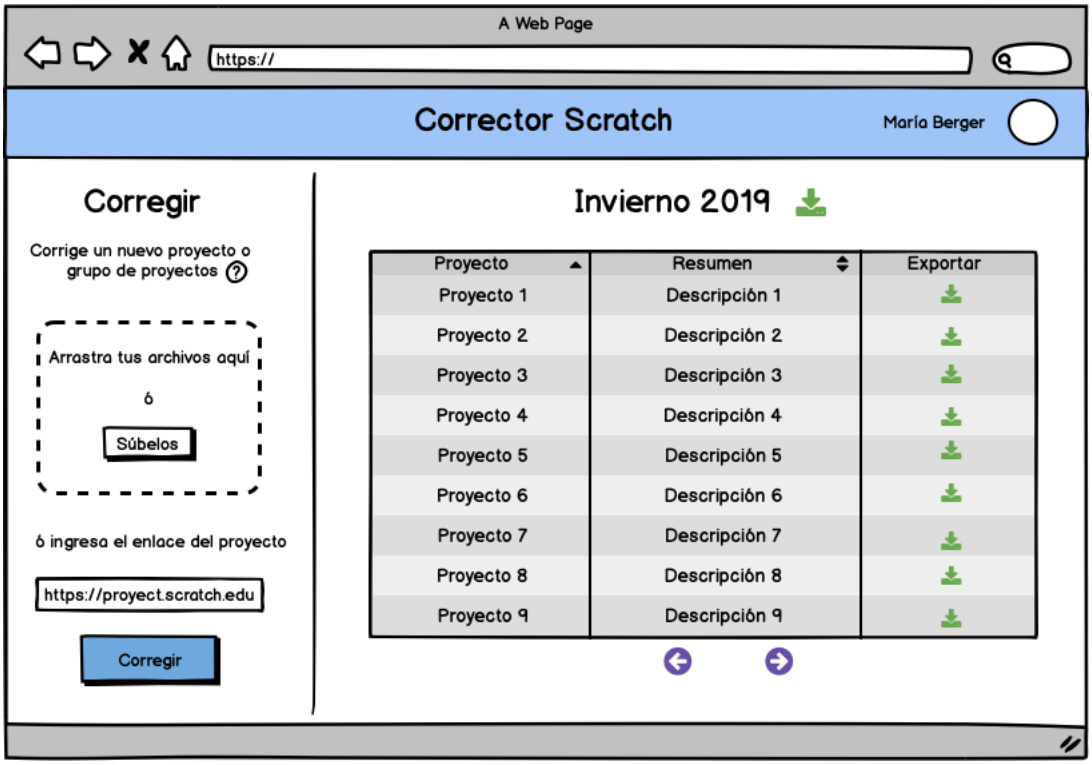



Figura 3.6: Vista de un curso

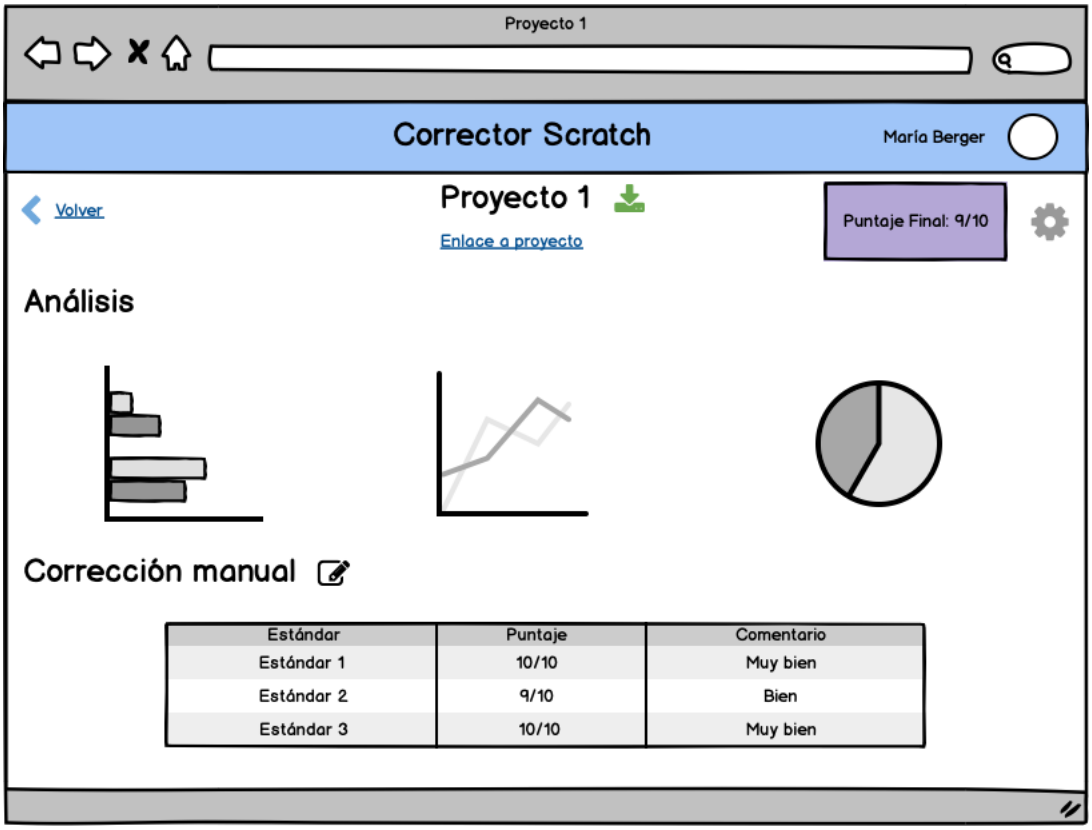



Figura 3.7: Vista de un proyecto

# Capítulo 4

# Implementación

Tomando en consideración las decisiones y el diseño propuesto en el Capítulo 3, se describe a continuación cómo se realizó la implementación de la plataforma descrita y las tecnologías usadas para el desarrollo del sistema en general. Primero, en la Sección 4.1, se revisan los aspectos generales del sistema, las tecnologías utilizadas, la metodología de trabajo que se siguió, y los distintos componentes del prototipo desarrollado. Luego en la Sección 4.2 se discute en detalle la implementación de los ejes principales presentados en el Capítulo 3, junto con las decisiones tomadas, los cambios a la propuesta inicial y los problemas que debieron ser solucionados.

## 4.1. Aspectos generales

El desarrollo de la plataforma se llevó a cabo en 3 etapas iniciales secuenciales hasta obtener un producto mínimo viable (desde ahora MVP) correspondiente a la unificación de ellas. Luego, se prosiguió de manera incremental incorporando funcionalidad al MVP de forma paralela entre sus partes hasta la finalización del proyecto.

La primera etapa consistió en la implementación del parser encargado de interpretar la información de los bloques de Scratch (originalmente en formato .json) y almacenarla como instancias de las entidades del modelo de datos visto en el capítulo anterior. La segunda etapa consistió en extender el backend que se había creado para almacenar los datos del parser para que sea una API. La tercera etapa consistió en implementar una versión inicial de la interfaz web propuesta (o frontend) con la cual se consultarán los endpoints del backend y se tendrá el manejo completo de la plataforma.

Una vez desarrolladas e interconectadas estas tres etapas se continuó agregando el resto de funcionalidad de forma paralela entre las 3. Cada 2 semanas se presentaba a la profesora guía (y clienta de la aplicación) los avances realizados para ir definiendo y priorizando las siguientes características a desarrollar.

La estructura del prototipo y los detalles de su implementación se explica en cada una de las subsecciones siguientes. El esqueleto del sistema fue implementado siguiendo un modelo de tres capas típico de una aplicación web, donde se tiene una capa de datos, de lógica y

una de presentación. El detalle sobre la implementación del parser, la extracción de datos y algunos endpoints complejos se detallará en la siguiente sección.

### 4.1.1. Datos

Entre las opciones comerciales de uso gratuito disponible, se eligió PostgreSQL como sistema administrador de bases de datos, porque es una tecnología de código abierto y tiene buena respuesta para grandes volúmenes de datos. Si bien para un uso local del sistema, como el que se implementa en este trabajo de título, es posible e incluso recomendado utilizar un administrador menos poderoso como lo es, por ejemplo, sqlite se eligió PostgresSQL en miras de extender este sistema en un futuro a una versión online que contemple múltiples usuarios (todo el equipo docente) con acceso a los mismos cursos.

### 4.1.2. Lógica

Para implementar todas las operaciones del sistema detalladas en este trabajo de título, se decidió usar Python y Django. Por un lado, se tenía más experiencia en esta tecnología que cualquier otra equivalente. Además, los objetos de Django poseen una dinámica compatible con el algoritmo implementado para el parser de Scratch, lo que facilita la integración de ambos pues así el parser es a ser parte del backend y no una API separada.

Finalmente, Django posee el framework Django REST, que permite construir APIs web de forma rápida y reusable a partir de objetos de Django. Esto facilitó la segunda etapa de desarrollo cuando se implementó la funcionalidad del backend sin haber creado la interfaz web, pues el framework proporciona la interfaz gráfica para probar la funcionalidad de los endpoints durante su desarrollo.

### 4.1.3. Presentación

Se implementó una interfaz web separada del sistema principal, que permita a los usuarios hacer uso del sistema de una forma amigable desde un navegador. La aplicación esta implementada con ReactJS, y consume la API del sistema principal para funcionar.

Esta aplicación se compone de las 3 vistas principales que se presentaron en los mockups del Capítulo 3: *Inicio, Curso* y *Proyecto*. Tal como sus nombres lo indican, en la vista de *Inicio*, un usuario puede visualizar todos sus cursos creados. Además, puede crear nuevos o eliminar existentes. En *Curso* puede ver todos los proyectos asociados a dicho curso y añadir nuevos proyectos para ser analizados. Finalmente, en *Proyecto* se pueden visualizar los resultados de los análisis y la rúbrica de evaluación manual.

## 4.2. Parser

En Capítulo 2 se describió el tipo de datos que provee un proyecto descargado desde la plataforma de Scratch (formato .sb3). Como primera acción, el parser debe procesar el proyecto cargado para extraer el archivo "project.json" y manejarlo como si fuese un diccionario de Python. Previamente se han mencionado la estructuración de los datos iniciales del proyecto y el modelo de datos creado para su almacenamiento, pero hasta ahora no se ha definido

una estrategia de conversión concreta y consistente con los diferentes tipos de proyectos que provee el lenguaje de bloques.

El principal desafío a la hora de definir un parser para Scratch, es que este lenguaje no posee una especificación formal y la única documentación oficial compete a la interfaz interactiva (tipos de bloques y su uso) y no a los datos tras ella. Debido a lo anterior fue necesario realizar una ingeniería reversa sobre el lenguaje comparando proyectos en la interfaz gráfica versus el json que generaban de forma incremental hasta lograr especificar las distintas secuencias y patrones involucrados. Si bien no es posible asegurar que este algoritmo funcionará para cada proyecto posible en Scratch, se probó con proyectos de ejemplo y de previas ediciones de los cursos del DCC con resultados consistentes.

El algoritmo generado se divide en 4 pasos principales: Preparación y manejo de archivos, Identificación de actores y escenarios, conversión para bloques secuenciales y conversión bloques para bloques anidados.

### 4.2.1. Preparación

No fue necesario definir una estrategia especial para el manejo de la extensión .sb3 pues como se estableció previamente, el archivo se comporta como un archivo comprimido estándar al modificar manualmente su extensión. Es por esto que bastó con procesar la carpeta con la librería nativa de Python "zipfile" la cual extrae archivos comprimidos asumiendo su extensión como .zip sin revisar extensión real.

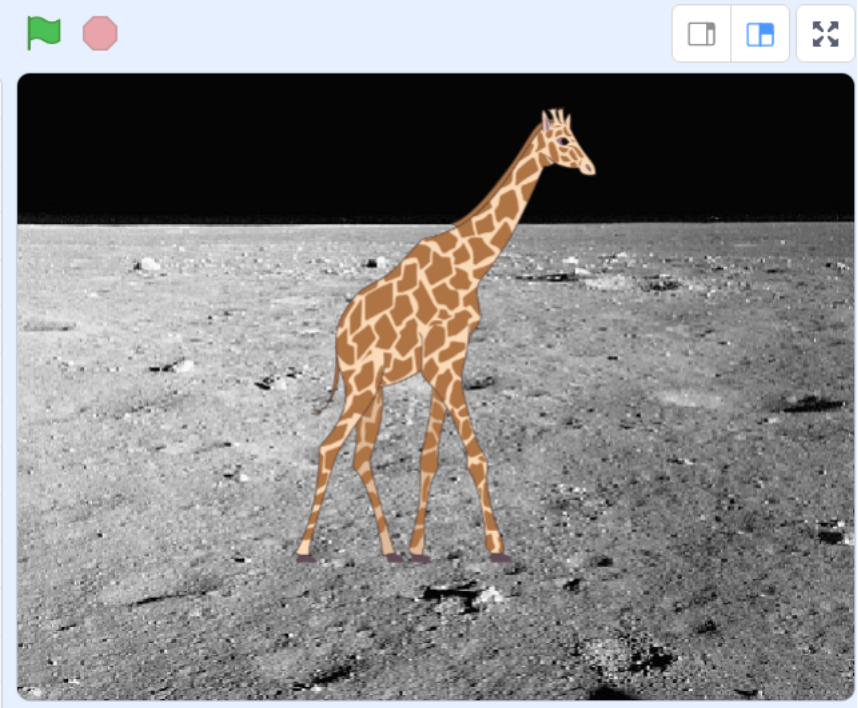
Para soportar la carga masiva de proyectos, se analiza inicialmente la extensión del archivo cargado. Si esta es .zip, se extraen todos los nombres su contenido a una lista. En caso de ser .sb3 se agrega ese único archivo a la lista. Con esta acción luego basta con iterar el parser sobre esa lista para soportar ambos tipos de carga de archivos.

Debido a la necesidad de mantener registro de los archivos cargados, en caso de requerirlos en el futuro, se utilizan librerías nativas de Python para crear una carpeta destinada a ese curso (en caso de no existir) y guardar todos los archivos dentro. De esta forma, se mantiene un registro ordenado de los proyectos, optimizando la descarga al tener menos archivos totales en cada ruta especifica.

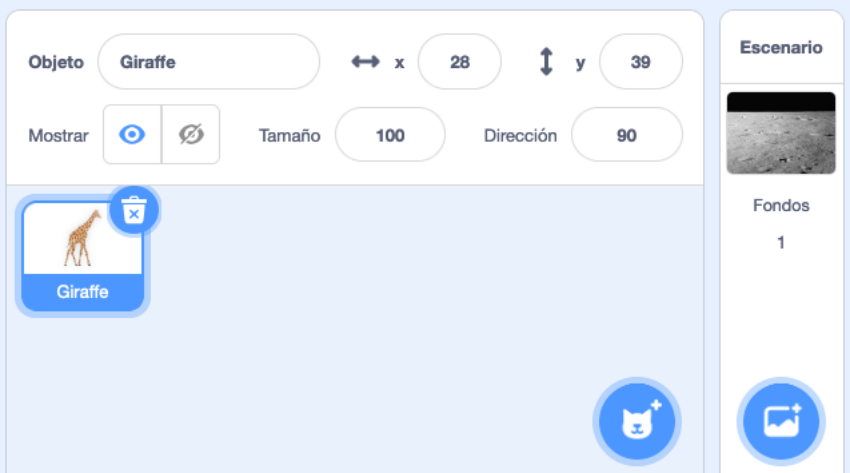
En este punto del parser, solo se crean instancias de Proyecto y Rubrica en el modelo de datos. Proyecto se inicializa con la información del archivo cargado y la ruta de su ubicación local. Rúbrica se inicia con todos sus campos vacíos y asociada al proyecto.

### 4.2.2. Actores y escenarios

Para esta subsección y las siguientes se creará un proyecto de forma incremental para poder acotar de mejor manera los ciclos del parser a las necesidades que van generando los distintos proyectos. Esto es meramente con fines demotrativos y debe tenerse en consideración la iteración final corresponde al json real generado. Además, se irán omitiendo progresivamente atributos no utilizados del json para aumentar su legibilidad y poder resaltar de mejor manera las diferencias entre uno y otro. Para comenzar, solo se mantendrá la lista de "Targets" de cada json.



(a) Visualización




(b) Configuración

Figura 4.1: Programa de ejemplo en Scratch

Para comenzar se crea programa en Scratch con exactamente un personaje ("Girafe") y un escenario cuyas visualización y configuración se encuentran disponibles en la Figura 4.1. Este proyecto no contiene bloques, solo los elementos previamente mencionados. En el json generado por este programa, disponible en el Código 4.1, se observan solo dos elementos: el primer diccionario de la lista corresponde al escenario y el segundo al actor.

```
"targets": [
  {
    "isStage": true,
    "name": "Stage",
    "blocks": {},
  },
  {
    "isStage": false,
    "name": "Giraffe",
    "blocks": {},
  }
  ]
```

Código 4.1: Extracto del archivo "project.json"

En el Código 4.1, se observa que en ambos elementos de la lista la llave "blocks" correspondiente a los bloques se encuentra vacía, sin embargo es posible identificar claramente los elementos del programa sin necesidad de conocer o existir bloques. Esto permite determinar que la única información intrínseca de los elementos de un programa es su nombre.

Debido a lo anterior, el primer paso del algoritmo es separar estos datos y crear instancias en el modelo de datos de tipo Actor y Stage según corresponda. En la práctica, como se observa en el Código 4.2, esto se lleva a cabo con un ciclo *for* que crea las instancias asociándolas al proyecto creado. Es importante notar que en el modelo de Actor se guarda el diccionarios de bloques completo (en este caso vacío), esto facilitará la creación de bloques en la siguiente subsección.

```
targets = data['targets']
for element in targets:
    if element['isStage']:
        stage = Stage(name=element['name'], project=project)
        stage.save()
    else:
        actor = Actor(name=element['name'],
                      blocks=element['blocks'],
                      project=project)
        actor.save()
```

Código 4.2: Primer ciclo del parser de datos: extracción de personajes y escenarios

### 4.2.3. Procesamiento de secuencias de bloques lineales

Al programa de la subsección anterior se le agregan los bloques de la Figura 4.2, correspondientes a una secuencia lineal de bloques. El primero corresponde a un bloque de tipo "Evento" y el segundo a un bloque de tipo movimiento.

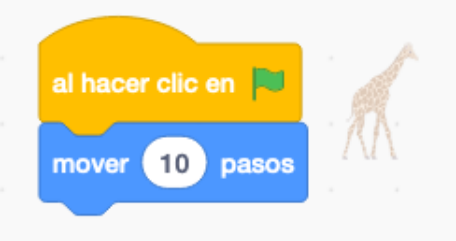



Figura 4.2: Secuencia lineal de bloques para ejemplo de la Figura 4.1

Al extraer el programa, se obtiene el archivo json del Código 4.3, al cual se le han omitido los datos referentes al escenario. En él se observa que ahora la llave "blocks" posee 2 instancias. De toda la información cada bloque, de momento solo se utilizan 3 llaves: "opcode" que corresponde al tipo de bloque (en el caso de la primera instancia es "event") y la acción especifica que representa ("whenflagclicked"), "next" que indica el nombre del bloque que viene inmediatamente después y finalmente, "parent" que indica el nombre que viene inmediatamente antes.

```
1    {
2      "isStage": false,
3      "name": "Giraffe",
4      "blocks": {
5        "Q4!2xV4$yL+Ppm(p)LP3": {
6          "opcode": "event_whenflagclicked",
7          "next": "|?OiOR({L-e5~},=d+F^",
8          "parent": null,
9          "inputs": {},
10         "fields": {},
11         "shadow": false,
12         "topLevel": true,
13         "x": 981,
14         "y": -737
15       },
16       "|?OiOR({L-e5~},=d+F^": {
17         "opcode": "motion_movesteps",
18         "next": null,
19         "parent": "Q4!2xV4$yL+Ppm(p)LP3",
20         "inputs": {
21           "STEPS": [1,[4,"10"]]
22         },
23         "fields": {},
24         "shadow": false,
25         "topLevel": false
26       }
27     }
28     }
29     ]
```

Código 4.3: Extracto de "project.json" para ejemplo de la Figura 4.2

En este ejemplo se observa como el primer diccionario, correspondiente al bloque de tipo


Figura 4.3: Secuencia lineal que genera bloque duplicado

"evento", tiene como valor "next" al bloque de tipo "motion" mientras que el bloque de tipo "motion" tiene como valor "parent" al bloque de tipo "evento". Esto indica una relación bilateral entre bloques padres e hijos, permitiendo que desde cualquier bloque de una secuencia se sepa todos los bloques que están antes y los que están después. Debido a lo anterior, se deduce que la estructuración de los bloques y sus relaciones divergen de la definición tradicional de un CFG, donde las aristas son en una sola dirección, permitiendo adoptar una definición más libre.

Como primera instancia, se definió el ciclo para crear los modelos asociados a los bloques como se muestra en la Figura 4.4, en ella se observa que para cada actor itera sobre la lista de bloques que se guardó en el modelo Actor en la subsección anterior, creando una instancia del modelo Block para cada uno de ellos. Como se mencionó previamente, de toda la información disponible para cada bloque solo se utilizarán las llaves "opcode", "next" y "parent" junto al nombre del bloque, con el detalle de que para la llave "opcode" se separa el tipo de bloque y la acción representada para guardarlos como atributos separados en la base de datos, facilitando su posterior acceso.

```
for actor in Actor.objects.filter(project=project):
    blocks = eval(actor.blocks)
    for block in blocks:
        full_type = blocks[block]['opcode'].split('_')
        type = full_type[0]
        action = blocks[block]['opcode'][len(type)+1:]
        block = Block(name=block,
                      type=type,
                      action=action,
                      next=blocks[block]['next'],
                      parent=blocks[block]['parent'],
                      actor=actor,
                      project=project)
        block.save()
```

Código 4.4: Segundo ciclo del parser de datos: procesamiento de secuencia lineal

El enfoque descrito funcionó efectivamente para la mayoría de los proyectos testeados. Sin embargo, se detectó que para algunos bloques específicos un bloque en la interfaz gráfica de Scratch crea más de un bloque en el json, dividiendo las acciones en dos o más bloques separados. Esta acción se detecto en bloques de tipo "control", "sound" y "motion" pero no fue posible establecer qué acciones gatillan este comportamiento y cuáles no.

Un ejemplo de lo anterior ocurre para una secuencia lineal como la de la Figura 4.3, en ella se ve un programa muy similar al usado en el ejemplo anterior, siendo el reemplazo del segundo bloque el único cambio. A pesar de ser bloques del mismo tipo ("control"), en el caso de la Figura 4.2 existían dos instancias de bloques, mientras que en este caso se genera el json del Código 4.3.

```
    {
      "isStage": false,
      "name": "Giraffe",
      "blocks": {
        "Q4!2xV4$yL+Ppm(p)LP3": {
          "opcode": "event_whenflagclicked",
          "next": "9uT7V7{a;~8=T7FzNT6%",
          "parent": null,
        },
        "9uT7V7{a;~8=T7FzNT6%": {
          "opcode": "motion_glideto",
          "next": null,
          "parent": "Q4!2xV4$yL+Ppm(p)LP3",
        },
        "0iMQx(RaBz!1_~2lrCq.": {
          "opcode": "motion_glideto_menu",
          "next": null,
          "parent": "9uT7V7{a;~8=T7FzNT6%",
        }
    }
]
```

Código 4.5: Extracto de "project.json" para ejemplo de la Figura 4.3

En el Código 4.5 queda en evidencia que a pesar de solo existir un bloque de tipo "motion" en el programa, en el diccionario existen 2. Si se mira en detalle, el bloque del ejemplo ejecuta la acción de deslizarse por cierta cantidad de tiempo a cierta posición. En los bloques del json se observa que la primera instancia corresponde solo al movimiento "glide_to", mientras que la segunda corresponde a la configuración del menú en el cual está seleccionado "posición aleatoria".

Si se mira en detalle, la primera instancia tiene como padre al bloque de tipo "evento", lo que se condice con la interfaz gráfica y con el atributo "next" del dicho bloque. Por el contrario, la segunda instancia de los bloques de tipo "motion" tiene como padre a la primera instancia, pero la primera instancia no tiene como "next" a la segunda. Esto rompe la biyeccionalidad que se había establecido.

Para solucionar lo anterior, se tomó la decisión de considerar solo el bloque que mantiene la relación biyectiva e ignorar el resto. Como no se quería perder la información de los bloques duplicados en caso de ser necesaria en el futuro, se incluyó un atributo "primary" en el modelo Block y se diseño un ciclo encargado de recorrer todas las secuencias de bloques marcando como primarias aquellas que cumplan con la biyectividad y como no primarias las que no. Con este cambio es posible contabilizar solo los bloques primarios, los que reflejan fielmente la cantidad de bloques de la interfaz.

La solución anterior se llevó a cabo ejecutando el ciclo del Código 4.6. Para cada actor, se realiza una consulta a la base de datos para determinar el bloque inicial de cada secuencia (parent=None) los cuales se definen como primarios. Luego, se recorren los bloques de cada secuencia por separado utilizando el atributo "next". Todo bloque que sea accesible a través de dicho atributo será marcado como primario ya que se estableció previamente que si bien todos los bloques duplicados tienen atributo "parent", no existen en el atributo "next" de ningún bloque.

```
for actor in Actor.objects.filter(project=project):
    heads = Block.objects.filter(actor=actor, parent=None)
    for head in heads:
        graph = Graph(actor=actor, project=project)
        graph.save()
        head.primary = True
        head.is_head = True
        head.graph = graph
        head.save()
        while (next):
            block = Block.objects.get(name=next,
                                      actor=actor,
                                      project=project)
            block.primary = True
            block.save()
            prev = block
            next = block.next
```

Código 4.6: Tercer ciclo del parser de datos: bloques primarios

### 4.2.4. Procesamiento de secuencias de bloques anidadas

La solución presentada en la subsección anterior, si bien fue efectiva para corregir el problema de los bloques duplicados, puso en evidencia otro caso excepcional que no había explícitamente abordado. En el Capítulo 2, en particular en el Código 4.7, se comentó que la estructura de aquellos bloques que son anidables difiere de la estructura general del formato del archivo "project.json", rompiendo la biyectividad establecida y clave para el enfoque con el que se intentan parsear los datos.

Al momento de la entrega de este trabajo de título, por método de inspección solo se han identificado dos categorías de bloques las que permiten anidación. En la Figura 4.4 se muestra una de ellas, correspondiente al tipo "control", la otra corresponde al tipo "operador" (bloques de color verde).

Si bien ambas categorías representan acciones diametralmente distintas, en términos del json generado tienen el mismo comportamiento. El Código 4.7 corresponde al descargable del proyecto de la Figura 4.4, donde se observa que la primera instancia del diccionario de bloques corresponde al bloque incial de tipo "evento" que tiene como hijo al bloque de tipo "control" quien a su vez lo tiene como padre, es decir, comportamiento biyectivo esperado. Sin embargo, al observar el bloque de tipo "control", no tiene un bloque siguiente.

Si se ejecuta el Código 4.6 de la subsección anterior, este solo marcaría como bloques


Figura 4.4: Secuencia anidada de bloques para ejemplo de la Figura 4.1

primarios las primeras dos instancias del diccionario de bloques, dejando el bloque de color morado como no primario, es decir, no presente en la interfaz gráfica. Considerando que el número de bloques que se pueden incluir dentro de un bloque que permita anidación es ilimitado, se puede perder una cantidad importante de información a la hora de contar.

En el Código 4.7 se observa que si bien el bloque de tipo "control" no tiene hijos, el bloque de tipo "looks" (anidado) si lo define como padre. Inspeccionando más a fondo, se observa que el bloque contenedor si hace una referencia al bloque anidado, pero en lugar de hacerlo como "next", es mencionado en la llave "inputs". Al probar distintos bloques de estas características, la distribución en la que se repartían los bloques dentro de "inputs" cambiaba constantemente, por lo que no fue posible definir un enfoque a utilizando dicha llave, que además conllevaría a modificar todos los ciclos anteriores.

```
{
      "isStage": false,
      "name": "Giraffe",
      "blocks": {
        "(;c~dqKqw7[0zZrT5h$N": {
          "opcode": "event_whenflagclicked",
          "next": "4^7!S5q}W=9sm2TJXg:.",
          "parent": null,
        },
        "4^7!S5q}W=9sm2TJXg:.": {
          "opcode": "control_forever",
          "next": null,
          "parent": "(;c~dqKqw7[0zZrT5h$N",
          "inputs": {
            "SUBSTACK": [
              2,
              "kW5P[hP$J?8J@9|vUuck"
            ]
          },
        },
        "kW5P[hP$J?8J@9|vUuck": {
          "opcode": "looks_say",
          "next": null,
          "parent": "4^7!S5q}W=9sm2TJXg:.",
      },
```

Código 4.7: Extracto de "project.json" para ejemplo de la Figura 4.4

Hasta el momento se poseen 3 piezas clave de información: La primera es que si bien el bloque contenedor no tiene como hijo a sus bloques anidados, estos si lo tienen como padre. La segunda es que los bloques anidados ya existen en el modelo de datos con toda su información pues fueron agregados en el segundo ciclo. La tercera y última corresponde a que si existiera un bloque posterior al bloque contenedor, este si saldría referenciado como hijo, por lo que puede diferenciarse fácilmente de los otros.

Con esta información se elabora el ciclo del Código 4.8. Para ejecutarlo, se agrega una instrucción al Ciclo 3, que especifica que para bloques de tipo "control" u "operator" se debe ejecutar una función recursiva que recupere todos los bloques anidados.

Por medio de la inspección, el bloque contenedor más grande identificado corresponde a la condición *if-else* de tipo "control" la cual posee hijos (no declarados) para condición, caso positivo y caso negativo. Es decir, si se consultan los bloques que tienen al bloque contenedor como padre sin contar el que esté abajo de él, el resultado de dicha consulta puede tener hasta un máximo de 3 elementos. De existir más bloques dentro del bloque contenedor, cada una de sus secciones se comporta como una secuencia lineal y, por ende, debe ser recorrida utilizando la estrategia del Ciclo 3.

Es debido a lo anterior es que la función *get_conditional_block* como primera acción consulta a la base de datos por los bloques que tengan como padre al bloque por el cual se llamó a la función que no cumplan con la biyectividad de las secuencias lineales. Se itera sobre ellos, marcándolos como primarios y utilizando la misma estrategia del Ciclo 3 para recorrer las secuencias de bloques lineales que cada bloque anidado pueda generar. Para cada nuevo bloque consultado a la base de datos se revisa su tipo y en caso de ser "control" u "operador" se llama recursivamente a la función para que revise las anidaciones correspondientes a ese bloque.

```
[...]
n_block.save()

if n_block.type == 'control' or n_block.type == 'operator':
    get_conditional_blocks(n_block, graph)

if n_block.next:
[...]

def get_conditional_blocks(block, graph):
    nested_blocks = Block.objects.filter(
                parent=block.name, project=project,
                actor=block.actor).exclude(name=block.next)

    # Explorations indicate in every query
    # 1.- Condition 2.- substack1 3.- substack2
    if len(nested_blocks) >= 2:
        condition_block = nested_blocks[0]
    else:
```

```
        condition_block = None

    while (len(nested_blocks) > 0):
        next_block = nested_blocks.first()
        next_block.primary = True
        next_block.graph = graph
        next_block.save()

        add_arch(block, next_block)

        if next_block.type == 'control' or /
                next_block.type == 'operator':
            get_conditional_blocks(next_block, graph=graph)

        prev = next_block
        next = next_block.next
        while (next):
            n_block = Block.objects.get(name=next, actor=actor,
                                        project=project)
            n_block.primary = True
            n_block.graph = graph
            n_block.save()

            add_arch(prev, n_block)

            # if block is type control it means there is more blocks

            if n_block.type == 'control' or /
                n_block.type == 'operator':
                get_conditional_blocks(n_block, graph=graph)

            prev = n_block
            next = n_block.next


        nested_blocks = nested_blocks.exclude(id=next_block.id)

       add_final_arch()
    return
```

Código 4.8: Cuarto ciclo del parser de datos: bloques anidados

Con este último ciclo, se incluyen todas las consideraciones necesarias para abordar los casos borde conocidos hasta el momento. Sin embargo, aún queda una problemática sin resolver: Si se quisieran visualizar los CFGs generados, no existe nada en la estructura utilizada que permita determinar qué bloques son de la secuencia lineal y cuales están anidados sin tener que repetir los ciclos 3 y 4.

Esto se solucionó incorporando la entidad Arch al modelo de datos, la cual representa las aristas de un CFG. Una instancia de esta entidad es creada cada vez que se salta de un bloque a otro y que en su atributo "type" describe el tipo de interacción, como por ejemplo

"direct" para secuencias lineales o "nested" para anidadas y "conditional" para secuencias de condiciones. Los cambios relacionados a esta incorporación se omitieron en favor de la legibilidad del código y fueron representados por la función *add_ arch()* y *add_final_ arch()*.

Además, como se mencionó en la etapa de diseño de la solución, se incorporó una nueva estructura al modelo de datos (Graph) encargada de relacionar los bloques y arcos de cada CFG asociándolos al proyecto y actor al que pertenecen, con la finalidad de facilitar el cálculo posterior de los CFG, por ello se observa que se añade como atributo a los bloques durante el recorrido.

Sintetizando, en la siguiente lista enumeran los pasos ejecutados por el parser al momento de procesar un proyecto. Es posible optimizar el tiempo de ejecución combinando ciclos, sin embargo, se prefirió mantenerlos separados pues al no existir una especificación formal del lenguaje, al momento de desarrollar, es más fácil detectar errores en el procesamiento de esta manera.

1. Se extrae y guarda localmente el archivo.
2. Se crean las instancias de Project y Rubric.
3. Se recorre la lista de "targets" creando instancias de los modelos Actor y Stage según corresponda. Se asocian al proyecto.
4. Para cada actor, se recorre el diccionario de bloques y por cada uno de ellos se crea una instancia del modelo Block.
5. Para definir el orden de los bloques se recorren todos los bloques asociados a un actor utilizando los datos *next* y *parent*, por cada conexión que se encuentre se crea una instancia del modelo Arch. Además, los bloques recorridos se marcan como "primarios" para excluir aquellos de información derivada que no son visibles en el programa.
6. Si el bloque es de tipo "control" u "operator", se utiliza una función recursiva para recorrer los bloques internos, ya que estos no siguen la estructura tradicional de *next* y *parent*. Estos bloques se marcan como primarios y se crean las instancias de modelo Arch diferenciando si corresponde a una condición o un bloque anidado.

## 4.3. Análisis automatizados

El propósito de convertir el proyecto a modelos de Django era precisamente facilitar el análisis posterior de ellos reduciéndolos a consultas a la base de datos. En esta primera versión del sistema, se definieron análisis netamente en base a conteos de bloques y generación de CFGs, se espera incluir análisis más complejos en versiones posteriores del sistema.

Actualmente, el sistema entrega la siguiente información:

- Cuentas generales como número total de bloques, actores, escenarios, CFGs generados, etc.
- Cuenta de bloques por tipo diferenciado por actor y en total, para bloques primarios y totales
- Cuenta de interacciones entre elementos del programa, ya sean objetos o personajes.

- Cuenta de bloques y CFGs inalcanzables, siendo definidos como aquellos sin un bloque de acción que desencadene su comportamiento.

Parte importante de la entrega de estos datos es que el frontend deba procesarlos lo menos posible, pero que a su vez sean lo suficientemente generales como para poder ser reutilizados según se requiera. Por ello, el formato de salida corresponde a un diccionario con dos entradas: "general" y "detail". La primera corresponde principalemente a cuentas totales: Total de actores, escenarios, CFGs generados, interacciones, total de bloques utilizados en el programa en general y separados por tipo de bloque y acción. La segunda entrada corresponde al detalle por actor de los bloques desglosado por tipo y acción. Teniendo cada actor su propio diccionario.

Por ejemplo, en el Código 4.12 se muestran los resultados de los análisis generados para la Figura 4.2. En él se puede ver la totalidad de calculos realizados y su distribución.

```
{
  "general": {
    "nactors": 1,
    "n_stages": 1,
    "n_graphs": 1,
    "nnodo": 2,
    "complexity": 1
    "nblock_Giraffe": 2,
    "hanging_graphs": 0,
    "hanging_nodes": 0,
    "interactions": 0,
    "block_event_primary": 1,
    "block_event_all": 1,
    "block_event_whenflagclicked_primary": 1,
    "block_event_whenflagclicked_all": 1,
    "block_motion_primary": 1,
    "block_motion_all": 1,
    "block_motion_movesteps_primary": 1,
    "block_motion_movesteps_all": 1
  },
  "detail": {
    "Giraffe": {
      "block_event_primary": 1,
      "block_event_all": 1,
      "block_event_whenflagclicked_primary": 1,
      "block_event_whenflagclicked_all": 1,
      "block_motion_primary": 1,
      "block_motion_all": 1,
      "block_motion_movesteps_primary": 1,
      "block_motion_movesteps_all": 1
    }
  }
}
```

Código 4.9: Análisis generados para la Figura 4.2

A continuación, se expondrán las consultas (Querys) utilizadas más representativas.

### 4.3.1. Cuenta de bloques

En el Código 4.10 se muestra como, a partir de una Query para obtener todos los bloques de proyecto (*Block.objects.filter(project=project)*) se itera sobre los resultados para incluir cuentas por total de bloques y dividido según su tipo y acción.

```
for block in Block.objects.filter(project=project):
    if block.primary:
        statistics['general'][
            f'block_{block.type}_primary'] = statistics['general'][
                f'block_{block.type}_primary'] + 1
        statistics['general'][
            f'block_{block.type}_{block.action}_primary'] = statistics[
                'general'][
                    f'block_{block.type}_{block.action}_primary'] + 1
    statistics['general'][f'block_{block.type}_all'] = statistics[
        'general'][f'block_{block.type}_all'] + 1
    statistics['general'][
        f'block_{block.type}_{block.action}_all'] = statistics[
            'general'][f'block_{block.type}_{block.action}_all'] + 1
```

Código 4.10: Análisis generados para la Figura 4.2

Para las cuentas por actor se ejecuta un proceso análogo, pero utilizando una Query para los bloques de cada actor específico e iterando sobre ella ejecutando los análisis anteriores.

### 4.3.2. Interacciones

Mediante la exploración manual de la plataforma de Scratch, se determinó que las interacciones entre actores o con objetos se producen se tocan. Esta acción solo es contemplada por un bloque en especifico de tipo "sensing" únicamente para la acción "touchingobject". Directamente de lo anterior se deduce que para calcular las interacciones basta con realizar la siguiente Query:

```
Block.objects.filter(
                project=project, type='sensing',
                action='touchingobject').count()
```

Código 4.11: Query para contar interacciones

### 4.3.3. Código inalcanzable

Se ha mencionado previamente en este trabajo de título que los únicos bloques que no tienen padre son aquellos que dan inicio a una secuencia de bloques. También se ha comentado que las únicas secuencias de bloques que se pueden ejecutar son aquellas que empiezan con un bloque de tipo "evento". Por ello, para definir la cantidad de CFGs que nunca se ejecutan, basta contar aquellos bloques sin padre que no sean de tipo "evento", lo cual traducido a una Query significa:

```
Block.objects.filter(
                project=project, primary=True,
```

```
parent=None).exclude(type='event').count()
```

Código 4.12: Query para contar CFGs incalzables

El real desafío consiste en contar cuántos bloques nunca son ejecutados, pues las secuencias de bloques que no parten con un evento no poseen un tamaño determinado. Para solucionar esto se utiliza la misma estrategia que en los Ciclos 3 y 4 del parser, es decir, se recorren los bloques a través de sus atributos "parent" y "next" tomando como inicial los resultados de la Query anterior y teniendo en consideración la cuenta de bloques anidados.

En el Código 4.13 se observa que se definió una función destinada a contar los nodos inalcanzables ( *count_hanging_nodes()* ) la cual a su vez define una función recursiva para recorrer las subsecuencias lineales ( *count_conditional_blocks()* ). A simple vista se observa que su composición corresponde a exactamente a misma estructura implementada en los Códigos 4.6 y 4.8 con la diferencia que en lugar de realizar cambios en la base de datos, lleva un contador para cada bloque recorrido.

Este comportamiento repetido era de esperarse, pues se estableció esa como la mejor estrategia para generar y recorrer los CFG, se espera optimizar esta implementación en el futuro ya sea extrayendo comportamientos comunes o generando una estructura que almacene los CGFs completos para no tener que calcularlos cada vez que se requiera.

```
statistics['general']['hanging_nodes'] = count_hanging_nodes()

def count_conditional_blocks(block, count):
    nested_blocks = Block.objects.filter(
            parent=block.name, project=project,
            actor=block.actor).exclude(name=block.next)

    while (len(nested_blocks) > 0):
        n_block = nested_blocks.first()
        count = count + 1
        if n_block.type == 'control' or n_block.type == 'operator':
            count = count_conditional_blocks(n_block, count)
        if n_block.next:
            next_block = Block.objects.get(name=n_block.next, actor=n_block.actor,
                                           project=project)

            if next_block.type == 'control' or next_block.type == 'operator':
                count = count_conditional_blocks(next_block, count)

            count = count + 1
            prev = next_block
            next = next_block.next
            while (next):
                block = Block.objects.get(name=next,project=project)
                if block.type == 'control' or block.type == 'operator':
                    count = count_conditional_blocks(count)
                count = count + 1
                prev = block
                next = block.next
```

```
        nested_blocks = nested_blocks.exclude(id=n_block.id)
    return count

def count_hanging_nodes(count=0):
    heads = Block.objects.filter(parent=None, project = project).exclude(type='event')
    for head in heads:
        prev = head
        next = head.next
        count = count + 1
        while (next):
            block = Block.objects.get(name=next,
                                      project=project)

            if block.type == 'control' or block.type == 'operator':
                count = count_conditional_blocks(block, count)
            prev = block
            next = block.next
            count = count + 1
    return count
```

Código 4.13: Funciones para contar nodos inacalzables

### 4.3.4. Generación de CFGs

Para mostrar los CFGs en la ficha de un proyecto, se implementó un endpoint que, a partir de los grafos asociados a un proyecto (modelo Graph), crea una lista separada por actor de los nodos y aristas de cada grafo. El formato de salida de la información se definió de esa manera para que fuera más fácilmente tratada por la librería para graficar utilizada en React: *react-graph-vis*.

```
def plot_info(request):
    [...]
    for g in graphs:
        final_data = {"nodes": [], 'edges':[]}
        arch_list = []
        # Final node for the end of the program
        final_data['nodes'].append({'id':0, 'label':'End␣Program'})
        added_nodes = []
        for arch in g.archs.all():
            # Exclude blocks that represent conditions
            if arch.type != 'conditional':
                if arch.block1 not in added_nodes:
                    final_data['nodes'].append({'id': arch.block1.id,
                        'label': arch.block1.type +  '_' + arch.block1.action})
                    added_nodes.append(arch.block1)
                if arch.block2:
                    print(arch.block1.action, arch.block2.action, arch.type)
                    if arch.block2 not in added_nodes:
                        final_data['nodes'].append({'id': arch.block2.id,
                            'label': arch.block2.type +  '_' + arch.block2.action})
                        added_nodes.append(arch.block2)
                    edge= {'from': arch.block1.id, 'to': arch.block2.id}
```

```
                else:
                    print(arch.block1.action, arch.type)
                    edge = {'from': arch.block1.id, 'to': 0}

                final_data['edges'].append(edge)
        all_graphs[g.actor.name].append(final_data)
    return Response(all_graphs)
```

Código 4.14: Función para calcular los ejes y aristas de un CFG

En el Código 4.14 se observa como se itera por cada actor, grafo y arco, agregando al diccionario final todas las conexiones entre ellos. Por como se modeló la base de datos, no es posible diferenciar entre los nodos de acción de los de condición más que por la etiqueta de su arista, por lo que se decide excluir de la lista aquellos bloques que estén unidos de forma condicional a los bloques anidados. En el Código 4.15 se muestra la lista enviada al frontend con los nodos y aristar para uno de los grafos del personaje Sprite 1 de la Figura 2.3. En la Figura 4.5 se muestra como esa lista es mostrada en la ficha de ese proyecto.

```
1   {
2   "Sprite1": [
3   {
4       "nodes": [
5           {
6               "id": 0,
7               "label": "End Program"
8           },
9           {
10              "id": 583,
11              "label": "event_whenflagclicked",
12              "color": "#FFBF00"
13          },
14          {
15              "id": 584,
16              "label": "motion_movesteps",
17              "color": "#4C97FF"
18          },
19          {
20              "id": 585,
21              "label": "sound_play",
22              "color": "#CF63CF"
23          },
24          {
25              "id": 587,
26              "label": "looks_say",
27              "color": "#9966FF"
28          },
29          {
30              "id": 588,
31              "label": "sensing_setdragmode",
32              "color": "#5CB1D6"
33          },
34          {
```

```
                    "id": 589,
                    "label": "data_setvariableto",
                    "color": "#FF8C1A"
                }
            ],
            "edges": [
                {
                    "from": 583,
                    "to": 584
                },
                {
                    "from": 584,
                    "to": 585
                },
                {
                    "from": 585,
                    "to": 587
                },
                {
                    "from": 587,
                    "to": 588
                },
                {
                    "from": 588,
                    "to": 589
                },
                {
                    "from": 589,
                    "to": 0
                }
            ]
        },
        {...}
    ]
    }
```

Código 4.15: Ejemplo truncado de output para CFG del presonaje Building de la Figura 2.3

Al comparar los bloques de la figura Figura 2.3 con el resultado de la Figura 4.5 se observa que la representación es correcta y permite calcular fácilmente la complejidad ciclomática $E - N + 2 * P = 7 - 6 + 2 = 3$. Sin embargo, al generar los grafos de los programas más complejos, como el de la Figura 2.7, el CGF generado para la secuencia de bloques más grande del personaje Hippo mostró inconsistencias cuya causa no fue posible determinar. Como se observa en la Figura 4.6, se generan errores en la salida del ciclo "forever", donde tanto el nodo que representa el ciclo como el nodo interior apuntan al final del programa. Esta inconsistencia se presentó en la mayoría de los ciclos infinitos y no fue posible corregirla sin reestructurar gran parte del código de este trabajo de título, por lo que se propondrá como trabajo futuro.

En la imagen también se observa un nodo sensor apuntando hacia el final del programa, probablemente porque dado su nivel de anidación dentro de las condiciones, no quedo marcado

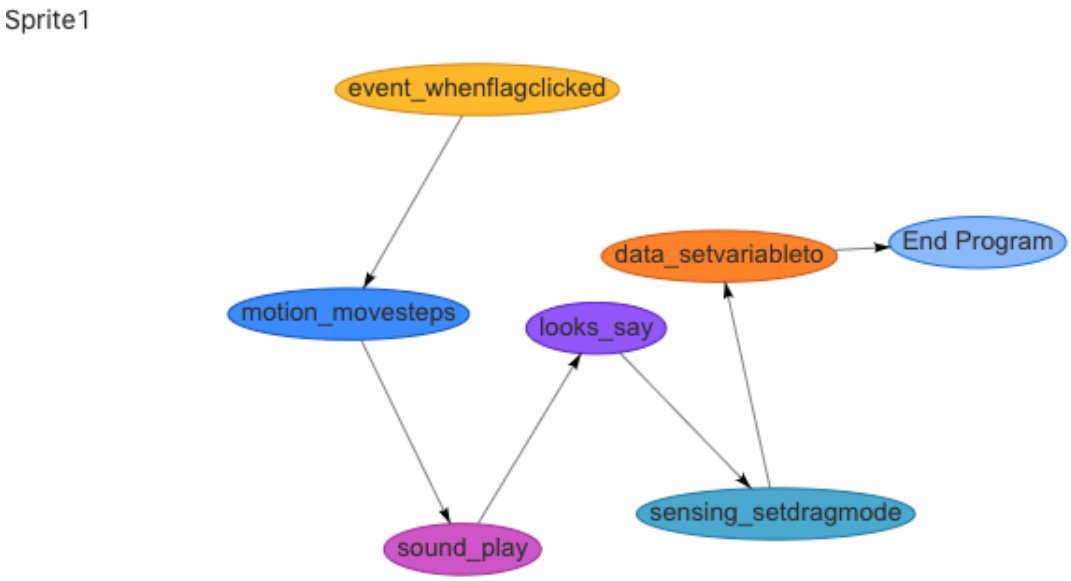



Figura 4.5: Representación en frontend de la lista de nodos y aristas del Código 4.15

como arista condicional.

Debido a lo anterior, los CFGs generados, aunque muy utiles para entender la estructura del proyecto, significan un riesgo muy grande a la hora de calcular la complejidad ciclomática ya que basta una arista extra para obtener resultados erróneos. En la próxima sección se propondrá un método alternativo para realizar este cálculo basado en la naturaleza de los bloques, con el fin de poder contar con esta información aunque no se haya corregido el algoritmo encargado de crear los arcos.

### 4.3.5. Complejidad ciclomática

Como se mencionó en capítulos previos, originalmente se buscaba calcular la complejidad ciclomática en base a Grafos de Control de Flujo. Sin embargo, debido a las limitaciones encontradas en su implementación, el riesgo de calcular la complejidad de forma incorrecta era muy alto, pues se comprobó que una falla en el diseño entregaba más aristas de las reales debido a la forma en la que se modelaba la presentación de los nodos anidados. A pesar de que se decidió mantener la presentación de los CFGs en la plataforma final, fue necesario encontrar otra forma de calcular la complejidad ciclomática que no dependiera de ellos.

Cuando se trabaja sobre CFGs, la complejidad ciclomática se calcula utilizando la fórmula $Complejidad = Aristas - Nodos + 2 * Componentes$, pero finalmente se reduce a una sola cosa: Cuantos caminos independientes de ejecución pueden producirse en un programa. Esto a su vez, puede reducirse a cuantas decisiones debe tomar el programa o, aplicado al caso de Scratch, cuantos nodos de decisión están presentes.

Debido a la naturaleza limitada de la programación en bloques, resulta sencillo saber de antemano que tipo de bloques abrirán un nuevo camino de ejecución, por lo que se intentará

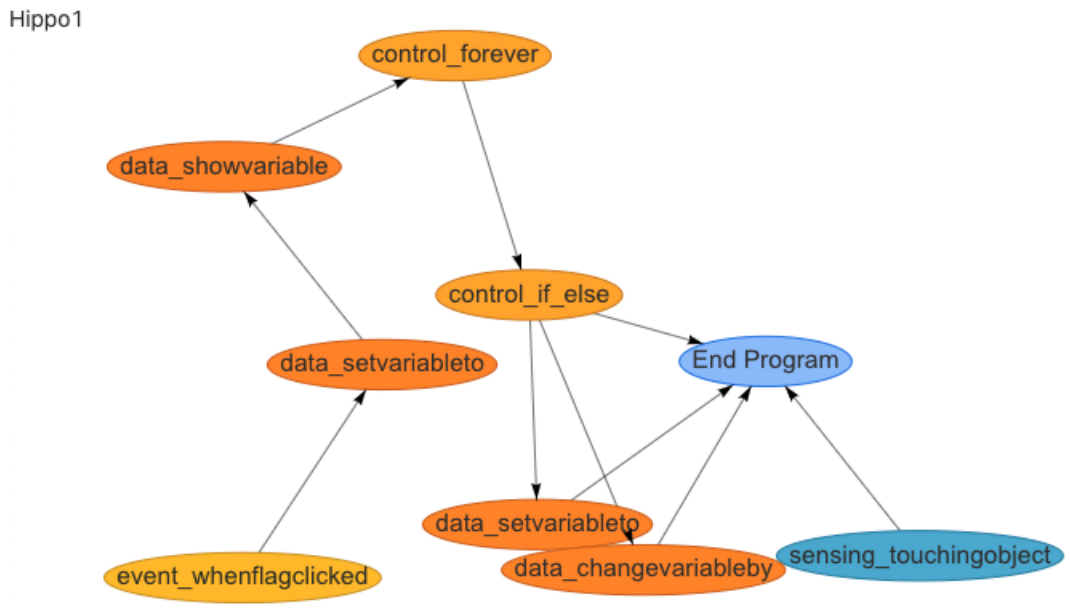



Figura 4.6: Ejemplo de CFG generado con errores

demostrar que, al menos como primera aproximación, es posible calcular la complejidad ciclomática de un programa en base a los CFG de únicamente los bloques que bifurcan la ejecución, en lugar de requerir calcular el CGF del proyecto completo.

Si un programa no posee ningún condicional, su complejidad ciclomática será 1. Con eso se deduce que cualquier bloque que no evalúe una condición, no aportará complejidad al programa. Inspeccionando los bloques disponibles en el Apéndice A, se observa que los únicos bloques con la capacidad de bifurcar la ejecución son aquellos de tipo control, en particular, los bloques "if", "if_else" y "repeat_until".

En las Figuras 4.7, 4.8 y 4.9 se pueden observar CFGs para los 3 tipos de bloques previamente mencionados. En el caso de la Figura 4.7, se puede ver que existen dos posibles caminos de ejecución: $I - IF - F$ y $I - IF - B1 - F$. Es decir, un bloque de tipo "if" agrega un nuevo camino de ejecución por sobre la cantidad original o, en otras palabras, aumenta la complejidad ciclomática en 1.

En el caso de los bloques de tipo "if-else" se tendería a pensar, dado que un bloque de tipo "if" aumenta la complejidad en 1, que se debería aumentar en 2. Sin embargo, al observar la Figura 4.8 se puede observar que nuevamente, solo existen dos posibles caminos independientes de ejecución. Esto se debe a que el camino principal queda comprendido dentro de la estructura "else". Debido a lo anterior, la presencia de este bloque también aumenta la complejidad ciclomática en 1.

Finalmente, para el bloque "repeat_until", en la Figura 4.9 nuevamente se encuentran 2 caminos de ejecución linealmente independientes: $I - C - F$ y $I - C - B1 - C - F$. Esta vez, la estructura agrega al procesamiento original, un posible ciclo donde se ejecuta el bloque $B1$ hasta que se cumpla la condición $C$, pero si esta se cumple desde la primera ejecución, el camino original se mantiene. Por ende, al igual que en los bloques anteriores, se aumenta la

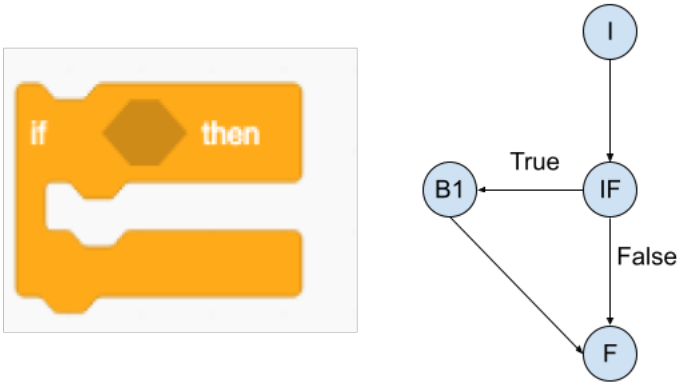



Figura 4.7: CFG para bloque if

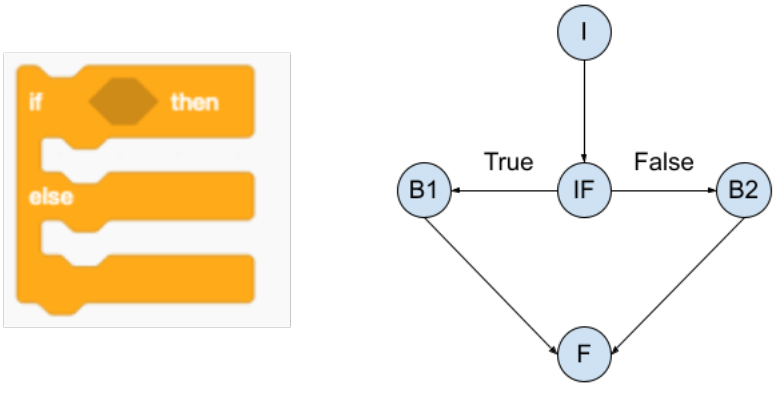



Figura 4.8: CFG para bloque if_else

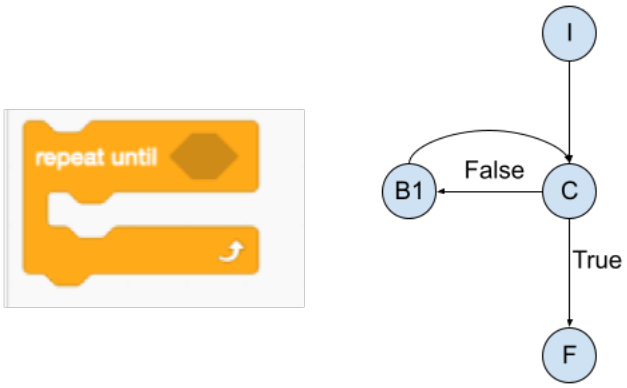



Figura 4.9: CFG para bloque repeat_until

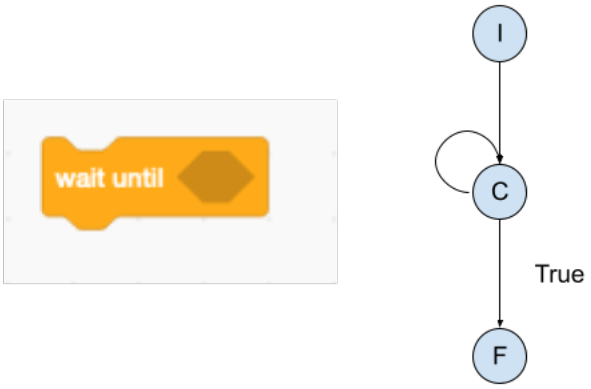



Figura 4.10: CFG para bloque wait_until

complejidad ciclomática en 1.

Un bloque que podría provocar confusión en el calculo de la complejidad es "wait_until", pues al igual que los previamente mencionados y como se observa en la Figura 4.10, genera un nuevo arco en el CFG. Sin embargo, en la misma figura se puede ver que no aporta con un camino independiente nuevo, pues finalmente la ejecución recorre siempre los mismos nodos en el mismo orden, por lo que este bloque no aumenta la complejidad ciclomática del programa.

Resumiendo los antecedentes presentados, solo 3 bloques pueden aumentar la complejidad ciclomática de un programa. Estos corresponden a "if", "if_else" y "repeat_until" y todos la aumentan en el mismo valor, 1. Por ello, para calcular la complejidad ciclomática bajo esta métrica, basta con recorrer todos los bloques asociados del proyecto y sumar aquellos que seam de los tipos mencionados a la complejidad inicial como se muestra en el Código 4.16

```
def calculate_cyclomatic_complexity():
        complexity = 1
        for block in project.blocks.all():
            if block.action in ['if', 'if_else', 'repeat_until']:
                complexity = complexity + 1
```

```
return complexity
```

Código 4.16: Código para calcular la complejidad ciclomática

## 4.4. Rúbrica

Al momento de incorporar la rúbrica de evaluación al proyecto, se consultó a una las investigadoras de los cursos de Scratch dictados por el DCC (y profesora guía de este trabajo de título) más información sobre esta parte del proceso actual de análisis de datos. La docente facilitó la rúbrica disponible Figura 4.11, donde que observan las distintas categorías evaluadas en la primera columna, seguidas por la especificación porcentual de cada nivel de completitud en las 4 columnas siguientes, finalizando con una columna vacía para comentarios por categoría. Esta rúbrica es utilizada durante la exploración individual de cada proyecto, donde la persona que evalúa debe ejecutar el programa, revisar los personajes y bloques y la documentación asociada y registrar sus hallazgos en este documento.

Al comparar los atributos de la entidad Rubric de la Figura 3.3 con las categorías estudiadas en la Figura 4.11, es directo deducir que para cada valor de la rúbrica existe un atributo en el cual se guarda el nivel de completitud en formato numérico. Además, se observa que para cada uno de estos campos se incluye un campo adicional de tipo texto para agregar los comentarios que sean necesarios.

El siguiente paso en la evaluación, una vez revisados todos los proyectos de un curso, es tabular los datos de cada rúbrica individual en una tabla como la presentada en la Tabla 4.1. En ella se registran los niveles de completitud asignados a cada categoría de cada proyecto del curso con la finalidad de estudiar el aprendizaje general de cada curso en su totalidad.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Proyecto_1 | 1 | 2 | 3 | 1 | 3 | 2 | 1 |
| Proyecto_2 | 3 | 3 | 3 | 1 | 2 | 2 | 0 |
| ... | | | | | | | |

Tabla 4.1: Tabulación de los datos de rúbricas

En capítulos previos se enfatizó reiteradamente que la riqueza que aporta una evaluación humana al análisis comprensivo de los proyectos no es automatizable. En particular, cuando se trata con programas en los que se narra una historia, coincidentemente los más frecuentes en este tipo de cursos, se vuelve aún más necesario un nivel de interpretación superior al que una máquina puede proporcionar.

Debido a lo anterior, la automatización del proceso nunca estuvo centrado en la rúbrica de la Figura 4.11, la cual se replicó de forma los más fiel posible en el frontend. La única optimización incoportada es que, en lugar de registrar el nivel de completitud de cada categoría, el evaluador puede hacer click sobre la casilla deseada cuando la tabla se encuentra en modo edición y de esta forma solo debe escribir para incluir comentarios de ser necesario.

El real proceso de optimización de las horas invertidas en la evaluación corresponden, en cambio, al proceso de tabulación. Considerando que toda la información tabulada ya fue

| | ninguno | principiante | intermedio | competente | |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | |
| **Siempre evaluar el delta con respecto a los proyectos de ejemplo** | | | | | |
| cohesion de los sprites | [0%, 10%] tiene código solo para controlarse a si mismo | (10%, 40%] tiene código solo para controlarse a si mismo | (40%, 70%] tiene código solo para controlarse a si mismo | (70%, 100%] sprites tiene código solo para controlarse a si mismo | |
| proposito de los bloques (solo codigo alcanzable) | [0%, 10%] de los bloques tiene un unico proposito | (10%, 40%] de los bloques tiene un unico proposito | (40%, 70%] de los bloques tiene un unico proposito | (70%, 100%] de los bloques tiene un unico proposito | |
| documentación | 0 comentarios | 1-2 comentarios | 3-6 comentarios | > 6 comentarios | |
| codigo alcanzable | solo [0%, 10%] bloques son alcanzables | solo (10%, 40%] de los bloques son alcanzables | solo (40%, 70%] de los bloques son alcanzables | solo (70%, 100%] de los bloques son alcanzables | |
| nombres de items (variables, sprites y backdrops) apropiados | [0%, 10%] de items tienen nombres apropiados | (10%, 40%] de items tienen nombres apropiados | (40%, 70%] de items tienen nombres apropiados | (70%, 100%] de items tienen nombres apropiados | |
| grado de modificación del prototipo | de las instrucciones tomadas de los ejemplos, el (70%, 100%] de los cambios son cosmeticos | de las instrucciones tomadas de los ejemplos, el (40%, 70%] de los cambios son cosmeticos | de las instrucciones tomadas de los ejemplos, el (10%, 40%] de los cambios son cosmeticos | de las instrucciones tomadas de los ejemplos, el [0%, 10%] de los cambios son cosmeticos | |
| grado de extensión del prototipo | no se entiende que tipo de proyecto es, que se puede hacer | muy similar a los ejemplos dados, o con muchos errores | se nota alguna diferencia con los ejemplos, y se puede ejecutar aunque hayan algunos errores | se diferencia de los ejemplos, o integra 2 o mas ejemplos. Intuitivamente, funciona de forma esperada | |

Figura 4.11: Rubrica de ejemplo facilitada por el equipo docente

ingresada una vez en la rúbrica disponible de cada proyecto no existe motivo por el cual el equipo docente debiese utilizar parte del tiempo dedicado a la evaluación de proyectos en tabular los resultados. Debido a esto se utilizó la librería nativa de Python csv para extraer los datos de cada rúbrica de cada proyecto de un curso y exportarlos utilizando el formato presentaod en la Tabla 4.1.

## 4.5. Resumen

En este capítulo se revisaron las decisiones tomadas en la implementación del prototipo propuesto por este trabajo de memoria, así como las tecnologías elegidas para cada capa del sistema. Uno de los puntos más importantes discutidos en este capítulo es el manejo de los datos originales del programa en Scratch y cómo se utilizan para generar información accesible y reutilizable por sí misma. Además, el diseño de la base de datos presentado en el Capítulo 3 acompañado de las consultas diseñadas para el análisis de los datos de proyecto asegura la extensibilidad del código si se quisieran agregar nuevos análisis. Finalmente, se discute sobre el diseño e implementación de la rúbrica de evaluación y su tabulación de forma que no pierda la esencia del proceso actual pero si reduzca tiempos de trabajo innecesarios.

# Capítulo 5

# Validación

Este capítulo pretende demostrar que el trabajo realizado efectivamente representa una solución efectiva a la problemática planteada. Para ello, en la Sección 5.1 se contrastarán los resultados obtenidos con los mockups e historias de usuario planteadas en el Capítulo 3. Además, en la Sección 5.2 se contrastarán los resultados de la evaluación de los proyectos en Dr. Scratch del Capítulo 2 con los resultados obtenidos en la plataforma desarrollada.

## 5.1. Interfaz web

No existieron modificaciones a la distribución de contenido de los mockups presentados en la Sección 3.4.2, pero sí hubo cambios relacionados al contenido mostrado o el enfoque del mismo. Estos cambios se detallarán a continuación

### 5.1.1. Vista de inicio

Esta vista, disponible en la Figura 5.1 y correspondiente al mockup de la Figura 3.5, tiene como funcionalidad el manejo de cursos en el sistema. En particular, en la HU-1 se hace referencia a la creación de cursos, la que se encuentra en la esquina superior derecha del sistema. Además se pueden visualizar, editar y borrar los cursos creados y acceder a ellos clickeando alguno de la lista.

### 5.1.2. Vista de un curso

Al ingresar a un curso de los expuestos en la Figura 5.1 se presenta la vista mostrada en la Figura 5.2, la cual corresponde al mockup de la Figura 3.6 y se divide en 2 secciones principales.

En la sección izquierda de la página se puede observar el módulo de carga de proyectos, donde se ofrece la opción de subir uno o múltiples proyectos (en formato comprimido) a la plataforma para ser analizados. Al presionar el botón “Analizar proyecto(s)” se recargará la página y los nuevos proyectos aparecerán incorporados a la tabla de la sección derecha de la plataforma. Estas acciones comprenden lo descrito en las Historias de Usuario HU-2, HU-3


Figura 5.1: Página de inicio de la plataforma

y parte de HU-4.

La sección derecha de la vista corresponde a una tabla con todos los proyectos vinculados a ese curso que han sido analizados hasta el momento. La tabla contiene:

- **Identificador**: Identificador asociado a cada proyecto por el evaluador para mantener coherencia con otros documentos que puedan existir. Es un campo editable que por defecto está en blanco.
- **Proyecto**: Nombre del proyecto analizado.
- **Descargar Proyecto**: Al presionar el botón de esta columna se descargará el proyecto original en formato .sb3 subido por el usuario a la plataforma.
- **Ver Análisis**: Hipervínculo que lleva a la vista de análisis para el proyecto individual.

Adicional a la tabla, al lado de nombre del curso se observan dos elementos: El primero corresponde al mismo icono de descarga presente en la tabla de proyectos y su funcionalidad es exportar una carpeta comprimida con todos los archivos de proyectos cargados en el curso. A su derecha se encuentra un botón destinado a exportar la tabulación de todas las rúbricas del curso. Estas acciones comprenden lo descrito en las Historias de Usuario HU-6 y HU-7.

Para cualquier tipo de descarga en esta vista, se gatilla el proceso de descarga clásico de cualquier navegador disponible en la Figura 5.3 que en particular representa la exportación de las rúbricas del curso.

La principal diferencia entre el mockup de la Figura 3.6 y la versión final radica en los datos de la tabla. Originalmente se habían diseñado filas clickeables en una tabla de 3 columnas que contemplara solo el nombre del proyecto, una descripción agregada por el evaluador y la columna de descarga del proyecto. Posteriormente se tuvo conocimiento de un identificador


Figura 5.2: Vista de un curso

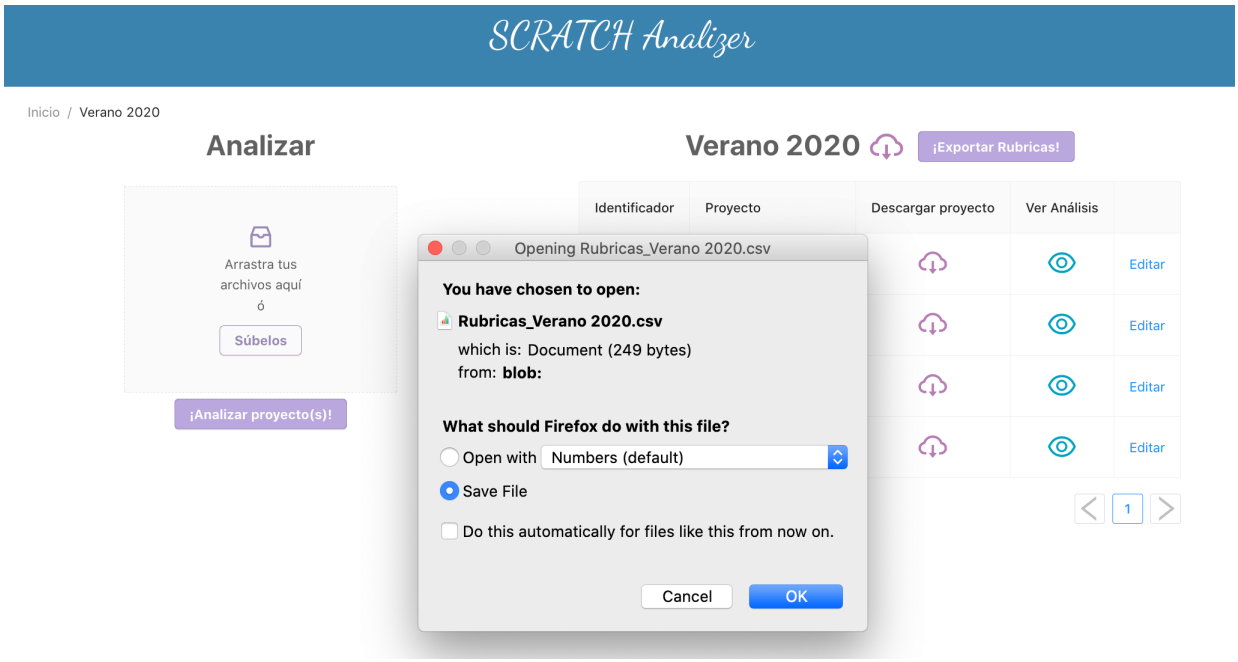



Figura 5.3: Vista de un curso al exportar las rúbricas

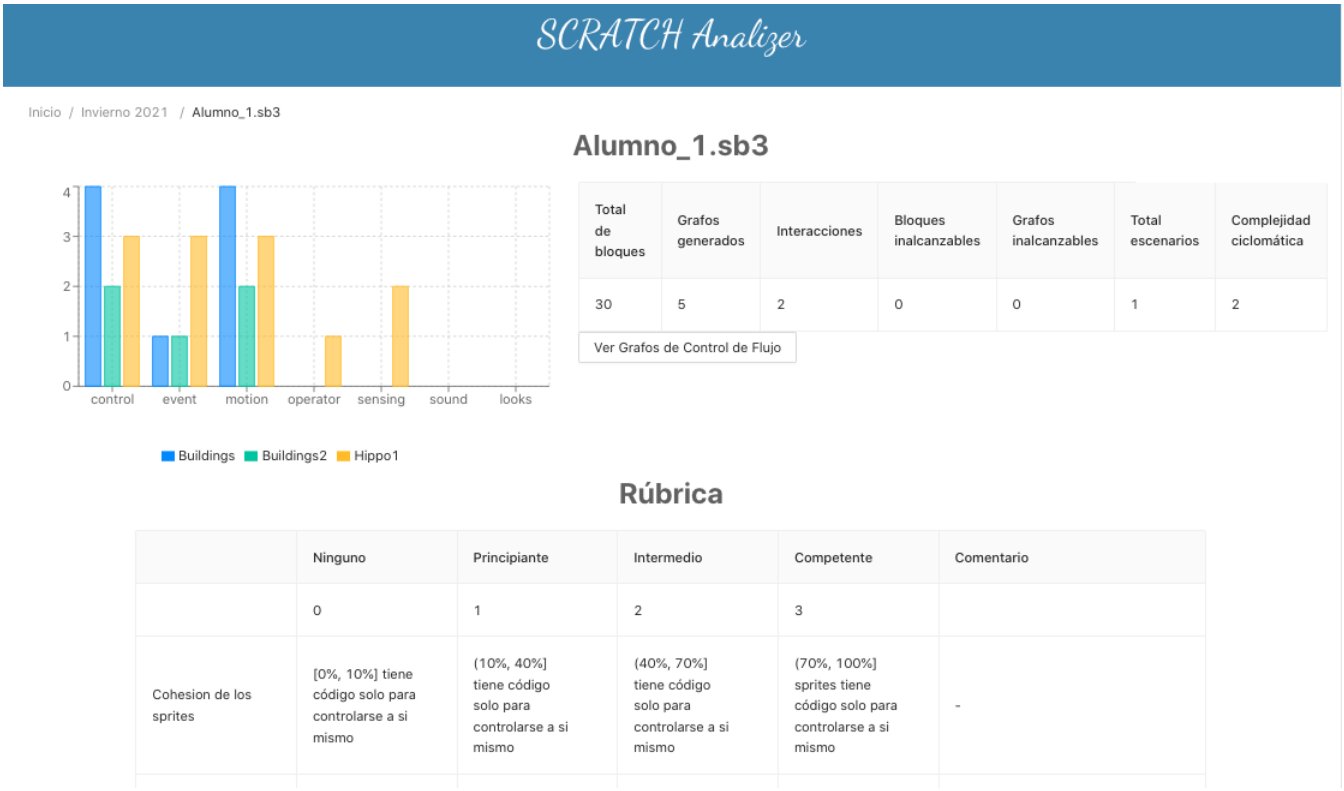



Figura 5.4: Vista de un proyecto: Análisis automatizados

único que se utiliza en el proceso de evaluación actual con el cual los docentes vinculan fácilmente los proyectos con las rúbricas y otras anotaciones, para mantener la compatibilidad con esto, se agregó la columna editable de Identificador (conectada al modelo de datos) para ingresar esta información. El resto de las columnas se modificaron con la finalidad de maximizar la usabilidad de la aplicación.

Por otro lado, se removió la posibilidad de cargar un proyecto haciendo uso del enlace generado por la plataforma de Scratch, pues debido a la falta de documentación de la API pública de la página quedó fuera de los alcances del proyecto.

### 5.1.3. Vista de un proyecto

Al presionar el hipervínculo para ver el análisis de un proyecto en la vista anterior, la plataforma se redireccionará a la página del proyecto.

Esta vista corresponde al mockup de la Figura 3.7 y se divide en 2 secciones: En la parte superior, disponible en la Figura 5.4, muestra la visualización de los análisis aplicados al proyecto. En la izquierda se muestra un gráfico de barras que representa la cantidad de bloques por tipo de bloque asignados a cada personaje del proyecto. En la izquierda de muestra una tabla con la información general del proyecto como sus bloques, CFGs generados, código inalcanzable, etc. Esta visualización corresponde a lo descrito en la Historia de Usuario HU-4.

En la parte inferior, disponible en la Figura 5.5 se observa la rúbrica de evaluación manual.

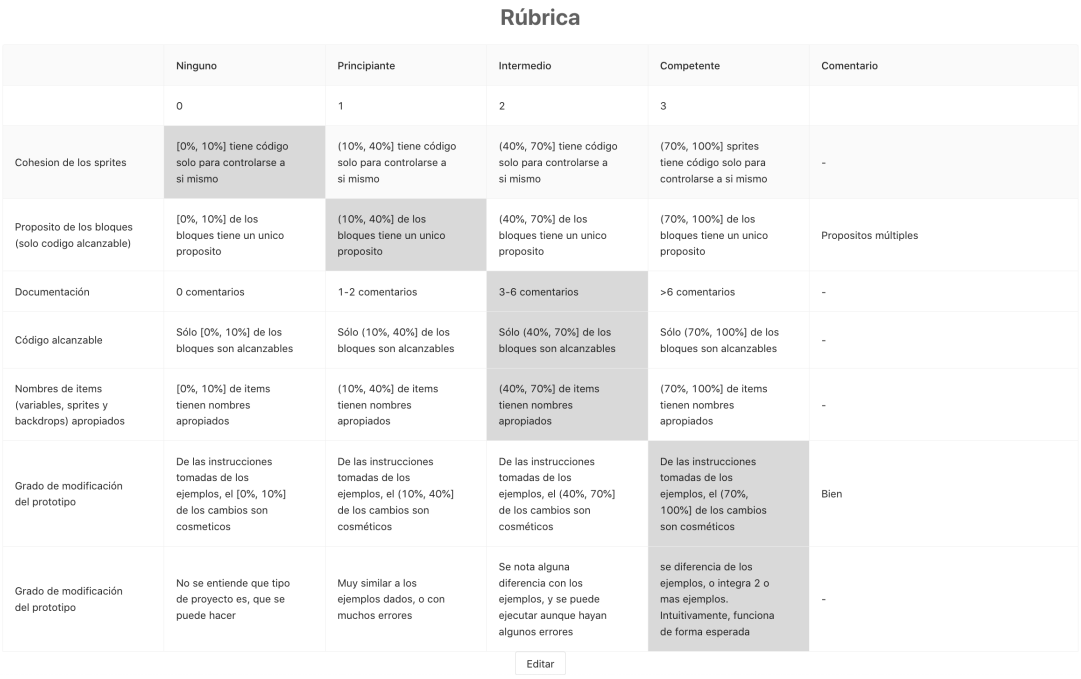



Figura 5.5: Vista de un proyecto: Rúbrica de evaluación manual

Rubricas_Verano 2020

| Identificador | Proyecto | Cohesion | Proposito | Documentación | Codigo alcanzable | Nombres apropiados | Modificacion del prototipo | Extensión del prototipo |
|---|---|---|---|---|---|---|---|---|
| | Alumno_1.sb3 | 1 | 1 | | 3 | 2 | 2 | 3 |
| | Alumno_2.sb3 | 0 | 1 | 2 | 3 | 2 | 1 | 0 |
| | Alumno_3.sb3 | | | | | | | |
| | Proyecto_ejemplo.sb3 | | | | | | | |

Figura 5.6: Archivo generado al exportar las rúbricas de un curso

Como se mencionó en la sección 4.4 del Capítulo de Implementación, el equipo docente de los cursos de Scratch facilitó la rúbrica actual utilizada en el proceso de evaluación y esta se replicó lo más fielmente posible en el proyecto. En ella, se puede seleccionar el nivel de competencia de cada categoría haciendo click sobre la celda cuando la tabla se encuentre en modo edición. Además, dentro del mismo modo, se pueden añadir comentarios para cada categoría según se requiera. Esta funcionalidad comprende lo descrito en la Historia de Usuario HU-5.

Al exportar las rúbricas desde la vista de un Curso, todos los datos de todas las rubricas de dicho curso se tabulan automáticamente siguiendo el formato de la Tabla 4.1 siendo el archivo exportado como el de la Figura 5.6.

### 5.1.4. Vista CFGs de un proyecto

Finalmente, al presionar el botón “Ver Grafos de Control de Flujo”, se llega a la vista presentada en la Figura 5.7, donde se muestran los CFGs generados para ese proyecto, separados por actor.

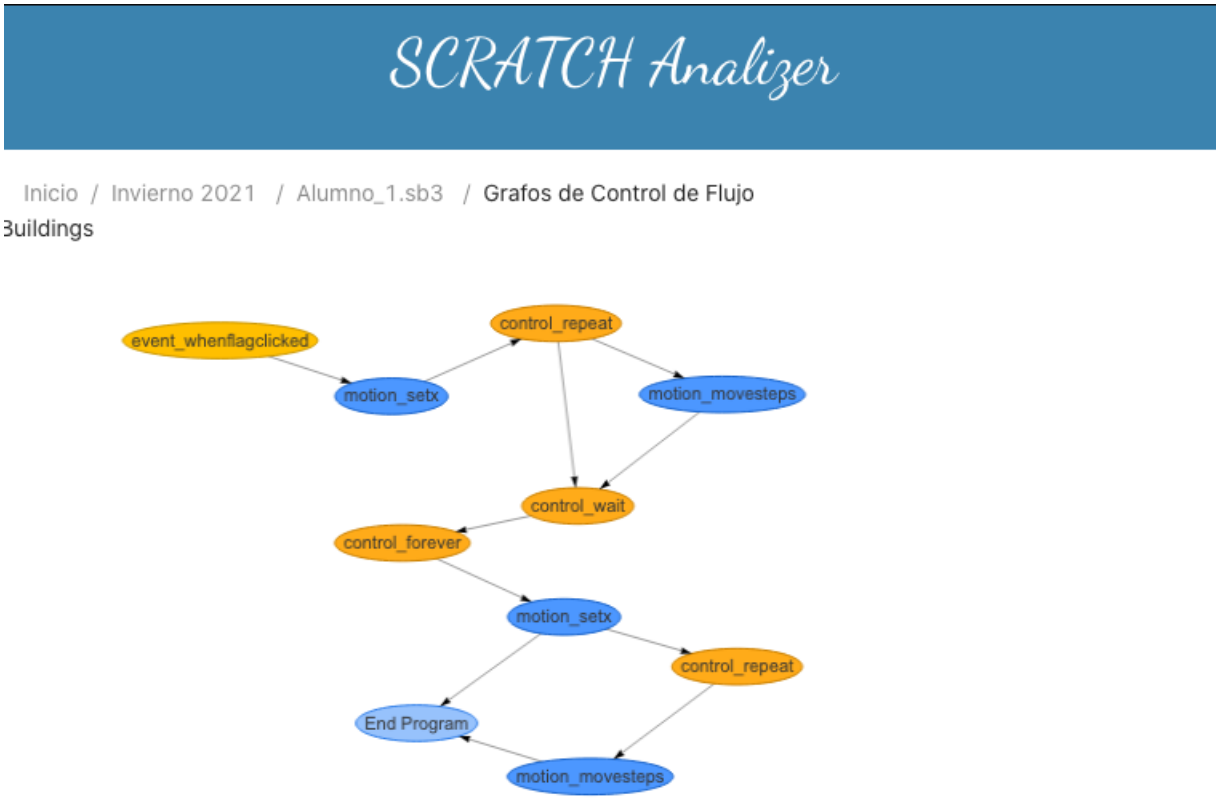



Figura 5.7: Fragmento de la Vista de CFGs generados para el proyecto de la Figura 5.4.

## 5.2. Comparación de análisis automatizados

En el Capítulo 2 se analizaron dos proyectos en Dr. Scratch, con la finalidad de ejemplificar su funcionamiento. Las Figuras relacionadas (2.3, 2.6, 2.7, 2.8) han sido replicadas nuevamente en este capítulo para facilitar la presentación de resultados.

En la Figura 5.8a se muestra un programa simple de un personaje y dos secuencias de bloques, una lineal y otra anidada. Al analizar este proyecto en la herramienta Dr. Scratch, se obtiene un puntaje final de 10 de un total de 21 puntos. El desglose de los puntos asignados por análisis y otros datos están disponibles en la Figura 5.8b.

Luego, en la Figura 5.9a se muestra un programa de 3 personajes con secuencias de bloques lineales y anidados asociados a todos ellos, es decir, un proyecto al menos 3 veces más grande. El problema que se planteó en el marco teórico y que se evidencia en la Figura 5.9b es que para un programa al menos 3 veces más grande y de mayor complejidad Dr. Scratch solo arroja una diferencia de 2 puntos del total de 21.

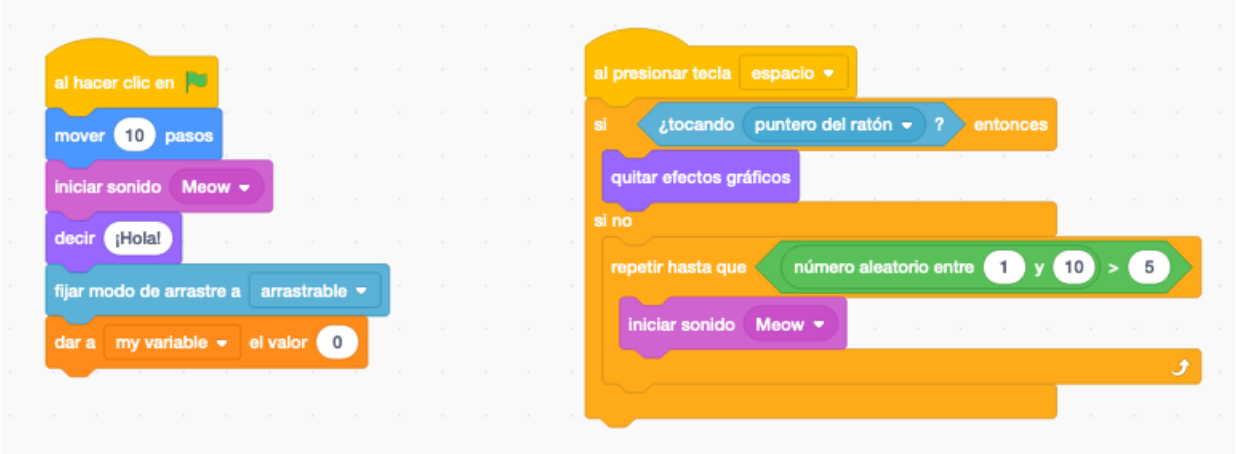
Tener un margen de sólo 2 puntos entre dos programas de complejidades tan diferentes es un problema a la hora de determinar el nivel de aprendizaje de un alumno, pues la plataforma de análisis no entrega la información necesaria para discernir la dimensión real de cada uno. Si bien se puede extraer mayor información desde los puntajes individuales de cada categorías, se generan algunas discordancias. Por ejemplo, el programa simple pareciera tener mejor paralelismo, sin tomar en consideración que todas las acciones están asociadas a un solo personaje. Por otro lado, se da bajo puntaje el programa complejo que debe coordinar las acciones de tres actores.

Como contraparte, se someten los mismo proyectos al análisis automatizado de la plataforma desarrollada en este trabajo de título, Scratch Analizer. Para el proyecto simple de la Figura 5.8a se obtienen los resultados presentados en las Figura 5.10a y Figura 5.10b: Aquí se indica un total de 13 bloques generados, separados en 2 grafos y con una interacción (correspondiente al bloque de interacción entre el personaje y el puntero del ratón).

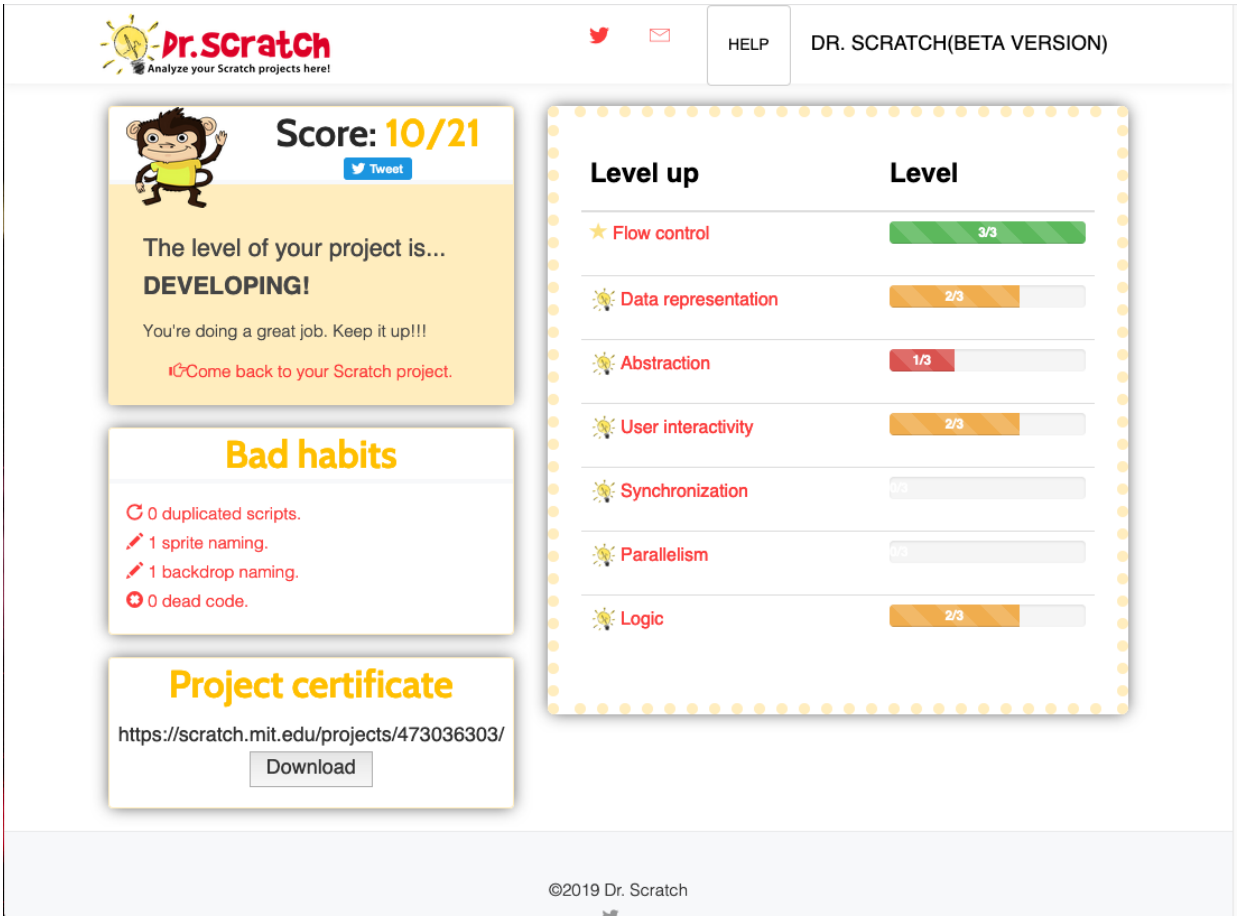
Por otro lado, en la Figura 5.12a y Figura 5.12a, correspondientes al ejemplo de la Figura 5.9, se observa inmediatamente al comparar ambos gráficos que existe un mayor número de personajes que en el proyecto anterior pero que la distribución de tipos de bloques es más amplia en el proyecto simple. En los datos, además, se representa que el proyecto posee 30 bloques distribuidos a lo largo de 5 CFGs independientes, con dos interacciones entre personajes. Al comparar con los proyectos, ambos resultados son acertados.

Luego, al acceder a la pestaña de Grafos de Control de Flujo, se puede verificar el número de grafos presentes y tener una visión más completa de la estructura del proyecto. En la Figura 5.11 se observan los 2 grafos generados para el proyecto de la Figura 5.8a y en las Figuras 5.13, 5.14 y 5.15 los 5 CFGs generados para el proyecto de la Figura 5.9a.

Al comparar los resultados de Scratch Analizer a los que se poseen en el sistema de corrección actual, se puede concluir que esta alternativa provee información más transparente al evaluador quien podrá determinar más rápidamente la magnitud y distribución de cada proyecto con la información presentada.



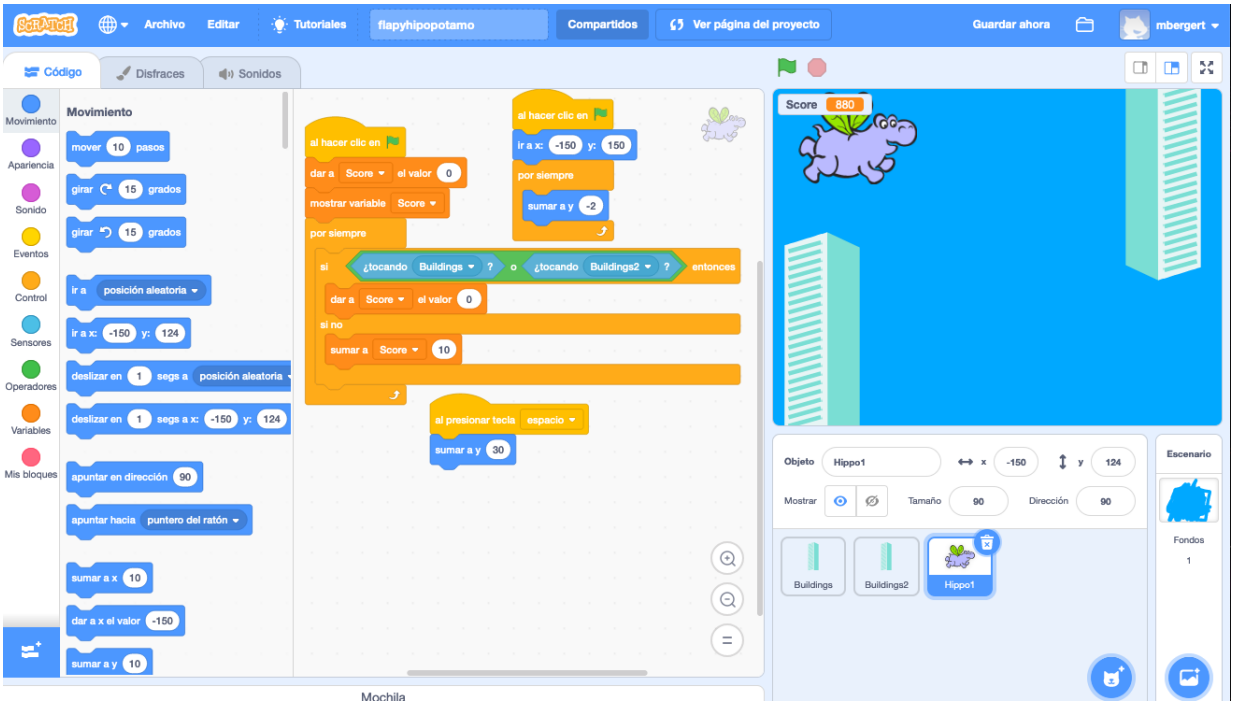

(a) Bloques



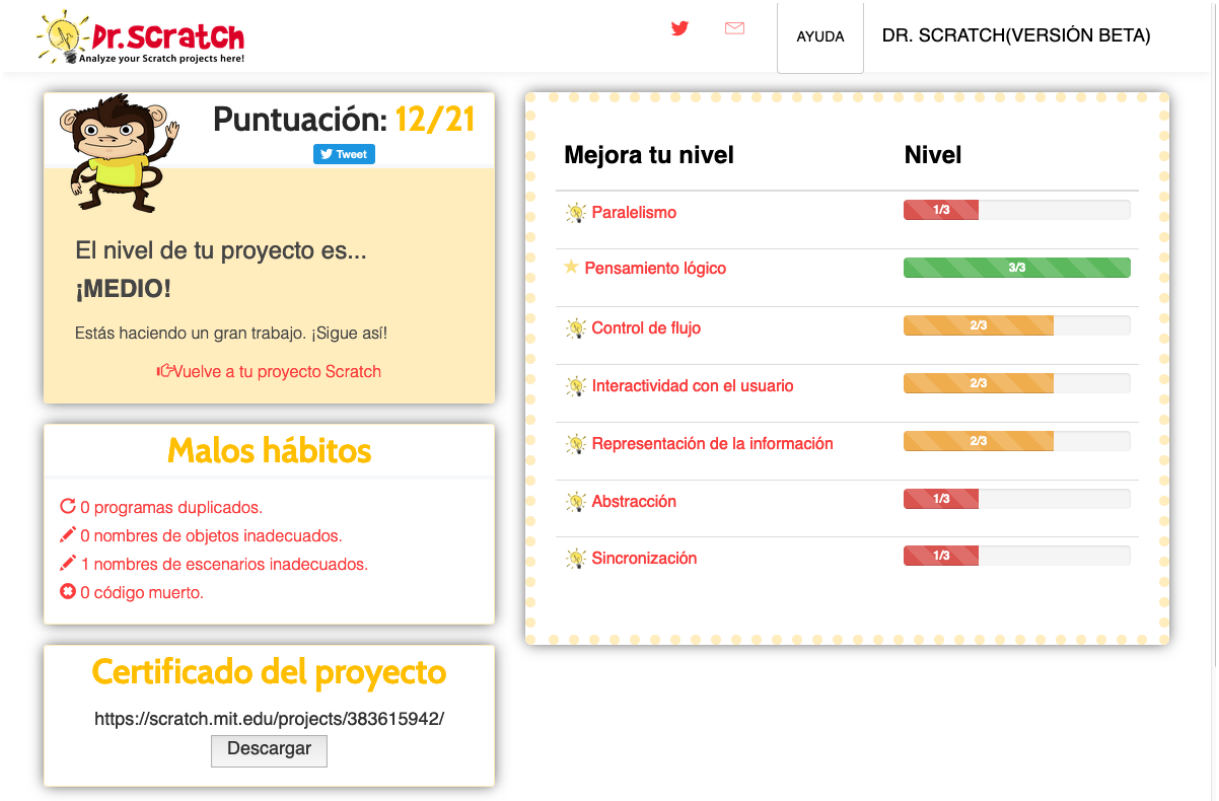

(b) Análisis en Dr. Scratch

Figura 5.8: Programa de ejemplo simple en Scratch

En condiciones normales, el sistema desarrollado podría no ser la mejor alternativa para un usuario que no tenga un basto conocimiento de cómo funciona la plataforma de Scratch, pues utiliza lenguaje técnico que no necesariamente es de conocimiento público. Este tipo de usuarios claramente se vería más beneficiado utilizando una plataforma como Dr. Scratch pues no entra en detalle sobre los análisis. Sin embargo, el público general no es el usuario



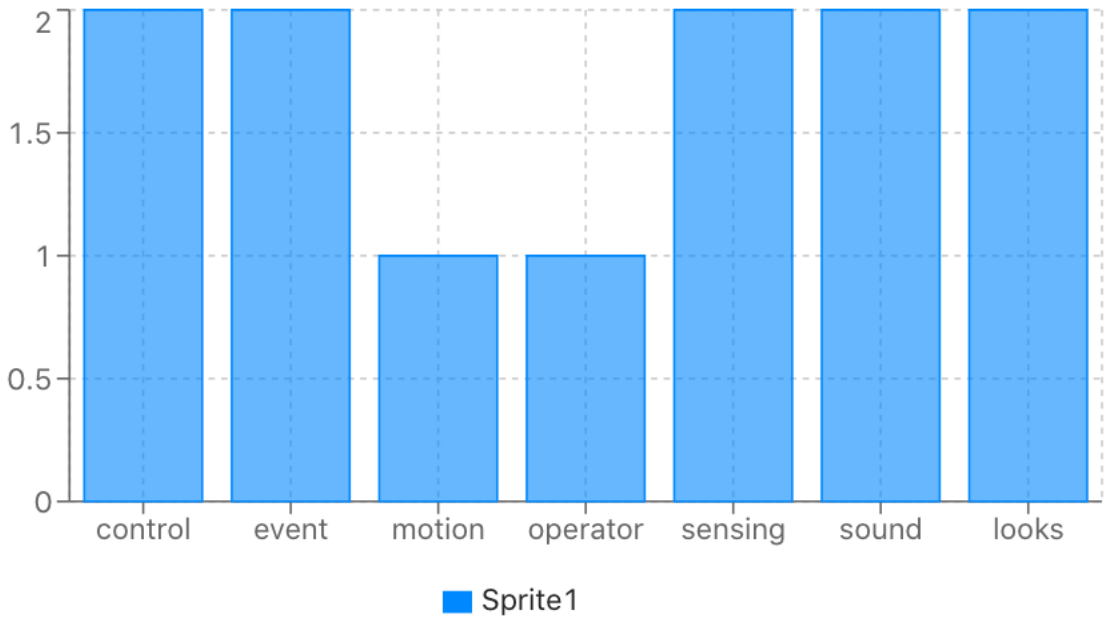

(a) Bloques del programa de ejemplo complejo




(b) Análisis de programa complejo en Dr. Scratchh

Figura 5.9: Programa de ejemplo complejo en Scratch

objetivo de la plataforma, pues esta fue diseñada específicamente para el uso interno del equipo docente de los cursos de Scratch dictados por el DCC, por lo que se puede concluir




(a) Gráfico

| Total de bloques | Grafos generados | Interacciones | Bloques inalcanzables | Grafos inalcanzables | Total escenarios | Complejidad ciclomática |
|---|---|---|---|---|---|---|
| 30 | 5 | 2 | 0 | 0 | 1 | 2 |

Ver Grafos de Control de Flujo

(b) Tabla

Figura 5.10: Resultados de someter el programa de la Figura 5.8a al análisis de Scratch Analizer

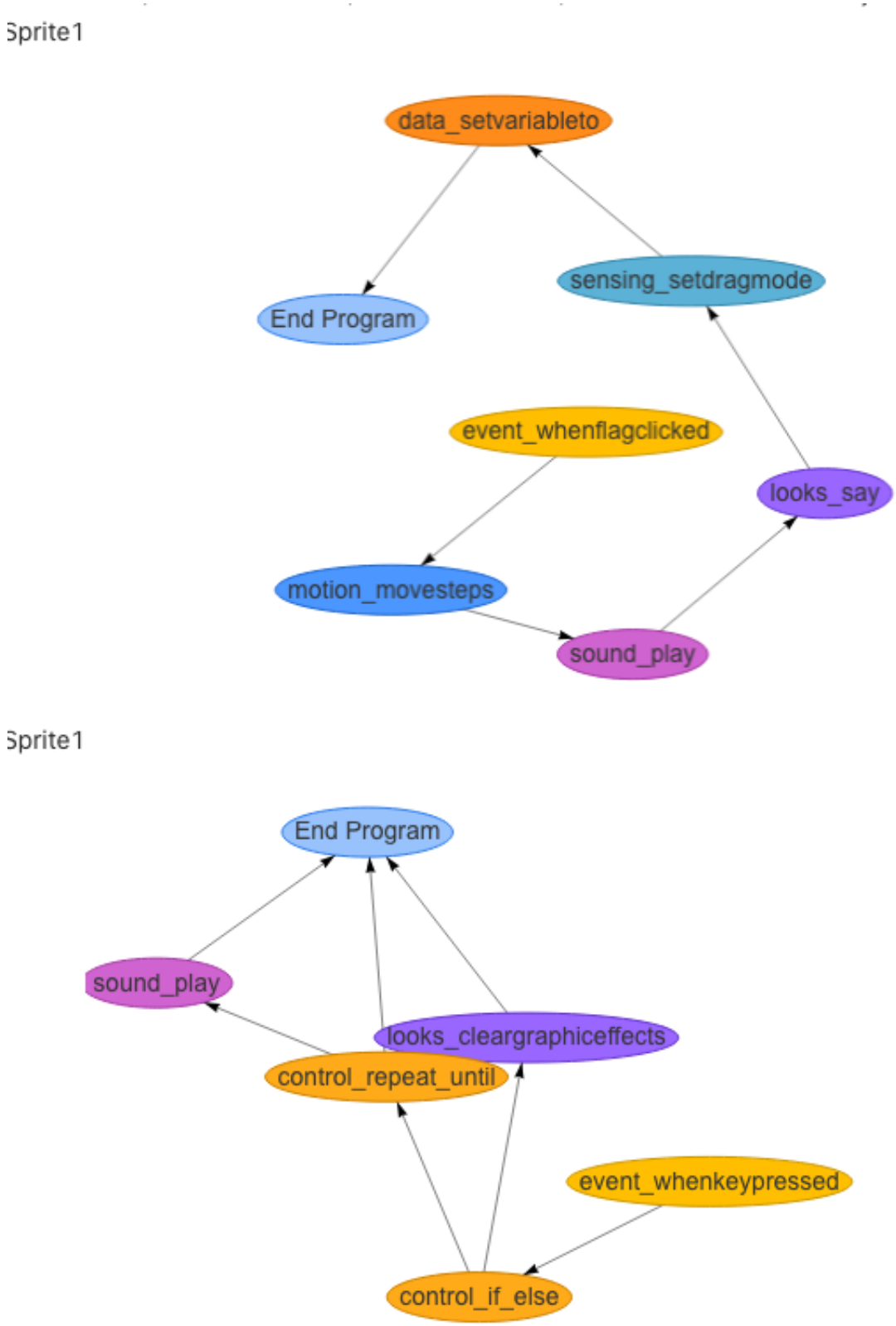
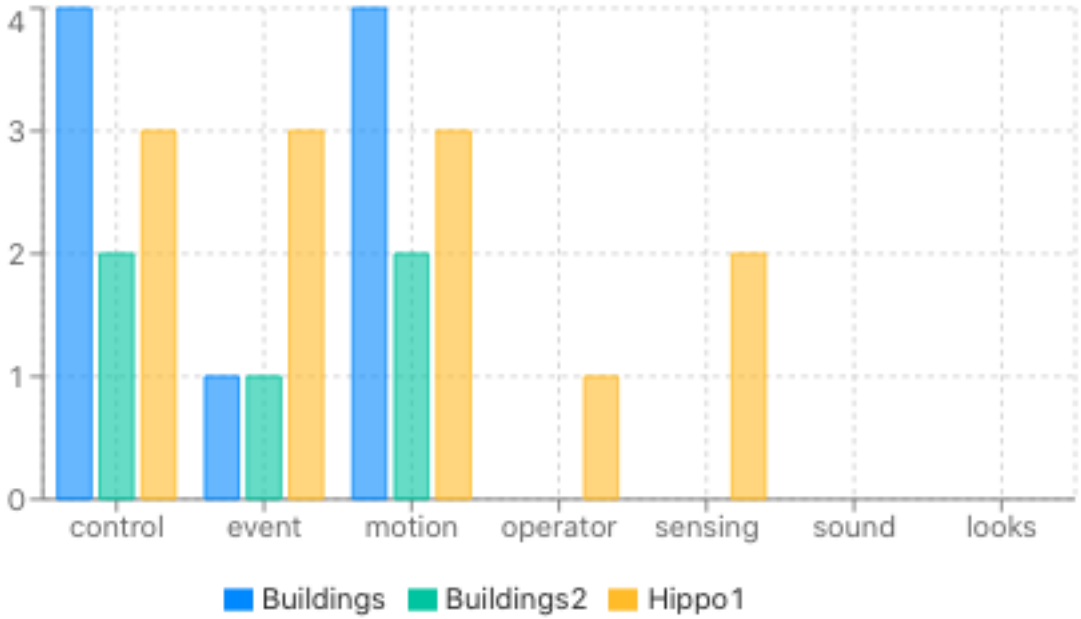



Figura 5.11: CFGs generados para proyecto de la Figura 5.8a



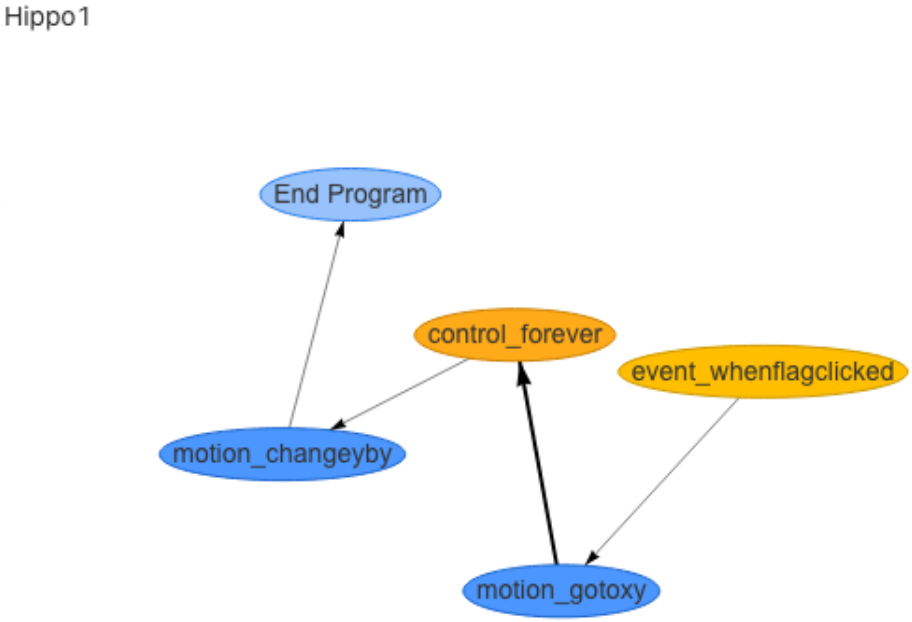

(a) Gráfico

| Total de bloques | Grafos generados | Interacciones | Bloques inalcanzables | Grafos inalcanzables | Total escenarios | Complejidad ciclomática |
|---|---|---|---|---|---|---|
| 14 | 2 | 1 | 0 | 0 | 1 | 3 |

Ver Grafos de Control de Flujo

(b) Tabla

Figura 5.12: Resultados de someter el programa de la Figura 5.9a al análisis de Scratch Analizer



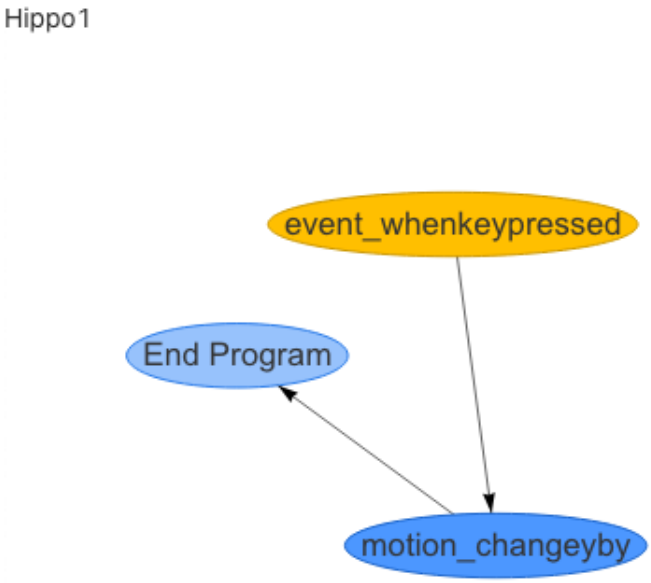




Figura 5.13: CFGs 1 y 2 generados para proyecto de la Figura 5.9a.


Figura 5.14: CFGs 3 y 4 generados para proyecto de la Figura 5.9a.

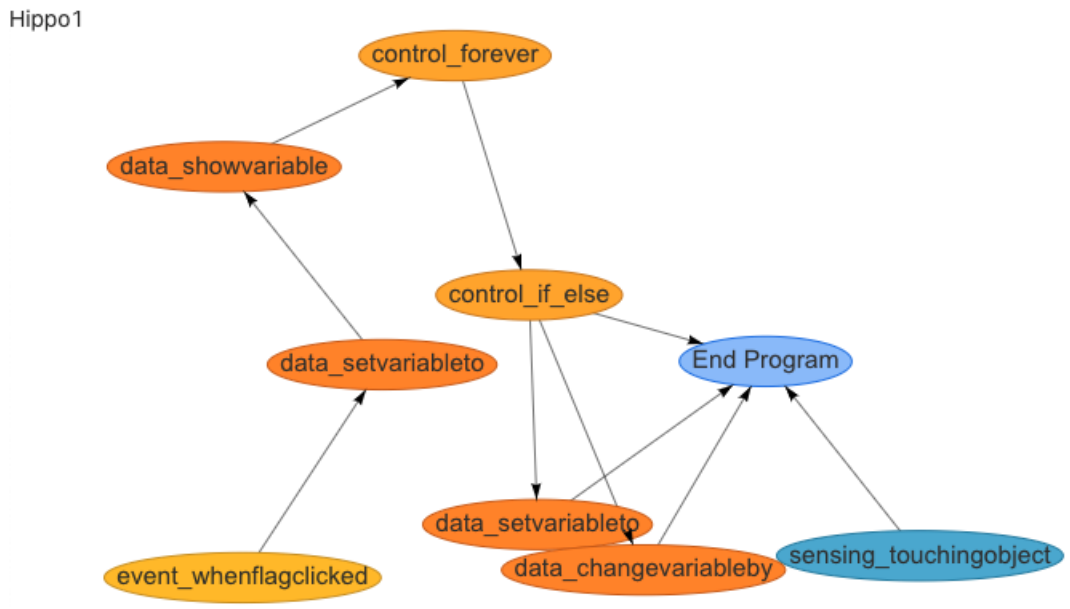



Figura 5.15: CFG 5 generado para proyecto de la Figura 5.9a.

que esta plataforma sí es una mejora en comparación al proceso actual.

## 5.3. Resumen

De lo expuesto en este capítulo, se satisfacen las historias de usuario presentadas Sección 3.1: con la interfaz mostrada en la Sección 5.1. Además se dan por cumplidos el objetivo general y los objetivos específicos de este trabajo de título, pues se construyó el sistema de acuerdo al diseño planteado, y en este capítulo fue validado según los parámetros propuestos. Finalmente, se realizan comparaciones entre el sistema de evaluación actual para verificar que la nueva plataforma suple las falencias detectadas en el Capítulo 2.

# Capítulo 6

# Conclusión

El trabajo realizado en esta memoria permitió unificar, optimizar y por lo tanto mejorar el proceso de evaluación de proyectos generados en los cursos de Scratch del Departamento de Ciencias de la Computación de la Facultad de ciencias Físicas y Matemáticas de la Universidad de Chile. Para llevarlo a cabo se estudió a fondo la dinámica de evaluación actual, identificando qué elementos aportaban valor por sí mismos y cuales presentaban falencias o requerían ser optimizados, identificando oportunidades de mejora.

Para los análisis automatizados que se utilizaban como complemento a la evaluación se propuso una versión más técnica, que se ajustara al nivel de comprensión del sistema que tienen los docentes dedicados al tema y no el público general, evitando así que tuviesen que deducir información en lugar de tenerla explícitamente. Por otro lado, se buscaron optimizaciones al sistema de rúbricas que cubrieran las necesidades de los investigadores pero que a su vez redujeran los tiempos de tabulación, reduciendo el tiempo total de evaluación. Para esto se propuso e implementó una aplicación web en la que se ejecutan estos procesos, con la que los usuarios pueden interactuar, incluyendo en ella un algoritmo dedicado a parsear el lenguaje de bloques Scratch el cual no posee una especificación formal.

En base a lo expuesto en esta memoria, se cumplieron los objetivos específicos en su totalidad. Se creó un parser de datos con la capacidad de convertir archivos .sb3 a instancias modelos de bases de datos, dándoles una estructura similar a la de un CFG. Con estas estructuras se definieron y aplicaron análisis en forma de consultas a la base de datos para determinar su complejidad en distintas dimensiones. Estos resultados se usaron como insumo para el diseño de visualizaciones que permitieran exponer la naturaleza de un proyecto de forma explícita que posteriormente fueron utilizadas en el prototipo de una aplicación web donde los docentes pueden llevar registro de sus cursos con sus respectivos proyectos. Además, se incluyó la rúbrica de evaluación diseñada por los docentes que permite mantener consistencia con los resultados de sus investigaciones previas, pero reduciendo los tiempos asociados a tabulación tanto de proyectos individuales como a nivel de un curso. Permitiendo también tener todas las herramientas de evaluación, antes separadas, interconectadas en un solo lugar. En cuanto a la validación del sistema, la docente quién también es la profesora guía de este trabajo de título quedó conforme con el primer prototipo, pues cumple con sus expectativas y las necesidades planteadas.

Considerando lo anterior, el objetivo general de mejorar y expeditar el análisis de programas desarrollados en los cursos de Scratch del DCC se cumplió. Ahora los docentes cuentan con un sistema unificado de todas sus necesidades que permite llevar registro de los datos que requieren para su investigación y exportarlos según necesiten. El sistema además, permite cargar masivamente proyectos, por lo que podrían cargar todos los datos de los 2 años de retraso de evaluación de proyectos que tienen hasta el momento y contar con los análisis automáticos de los mismos.

La metodología de desarrollo utilizada en esta memoria dio buenos resultados. Llevar de forma incremental la etapa de desarrollo del parser de datos permitió construir de forma ordenada un algoritmo para un lenguaje sobre el cual no existía mayor documentación o especificación formal, por lo que ir abarcando cada vez proyectos más grandes facilitó la detección oportuna de errores y su pronta corrección. Considerando que el desarrollo de esa etapa terminó tomando mayor tiempo al que se esperaba, el mantener una metodología incremental paralela a cada etapa del sistema ayudó a disminuir el riesgo de que quedarán partes inutilizables del sistema, porque cada vez que se agregaba una nueva funcionalidad, se incluía la interfaz de usuario para ejecutarla. Esto llevó al desarrollo casi completo de la interfaz idealmente planificada, quedando pendientes solo la ejecución los análisis complejos comentados en el Marco Teórico de este trabajo.

Finalmente, es importante destacar que considerando que los cambios de versiones de Scratch han significado cambios radicales en la estructura del json que genera, el sistema fue diseñado de tal manera que de ocurrir esto en una potencial versión 4 del lenguaje, bastaría extender el algoritmo que recorre el json para que considere la nueva estructura sin afectar esto al resto de la plataforma.

### Trabajo futuro

Entre los pasos a seguir después de este trabajo de título se encuentra la incorporación de todos los análisis mencionados en la Sección 2.3 del Marco Teórico, los cuales podrían aumentar considerablemente la completitud de la exploración de los proyectos, permitiendo mejores visualizaciones y por ende una mejor comprensión para quien las ve de la dimensión real del proyecto estudiado. Se espera corregir las limitaciones mencionadas en la generación de CFGs, con la finalidad de utilizarlos no solo como soporte visual sino que como base de nuevas métricas de evaluación. A su vez, mejorar la representación visual de los mismos para que el usuario deje de depender, en parte, de la exploración visual de los proyectos en la plataforma de Scratch. En esta misma línea también se sugiere incorporar la funcionalidad de cargar archivos para ser analizados con su link de la página de Scratch, evitando así que los docentes tengan que volver a cargar el proyecto en el sistema después de haberlo cargado en la página.

Con respecto a la docencia, otra mejora que se puede incluir al sistema es la incorporación de más datos referentes al curso o a los proyectos, como rango etario de los alumnos, procedencia e información sobre los proyectos utilizados en las clases para poder compararlos con los que produjeron los alumnos. Con esto se incorporaría mayor dimensión humana a los datos, pudiendo incorporarse a la vez esta información a la investigación.

Una tercera mejora, que también se consideró en el planteamiento original del proyecto pe-

ro que quedó fuera de alcance es levantar el proyecto en un servidor online, creando usuarios asociados a los cursos que puedan evaluar los proyectos de manera colaborativa. Esto permitiría a los docentes imitar su proceso actual de evaluación que generalmente es realizado en conjunto y no de manera individual. Esto toma vital importancia considerando que, debido a la pandemia actual, cada vez existen menos instancias presenciales y un sistema colaborativo facilitaría la evaluación sin necesidad de compartirse archivos de forma asíncrona.

Otras mejoras que se sugieren, pero que tienen un nivel de riesgo mayor, son embeber la interfaz de Scratch dentro del sistema para que se pueda visualizar la ejecución del programa dentro de la misma ficha de un proyecto. Esto evitaría depender de la página de Scratch para ver los programas y podría efectuarse todo el proceso de evaluación unificado de manera local.

Este trabajo de memoria es una primera aproximación a un sistema de generación automática de análisis de proyectos desarrollados en Scratch unificado capaz de incorporarse a una metodología de investigación activa, mejorando el proceso sin necesidad de cambiarlo y obligando a la pérdida de consistencia con investigaciones anteriores. Se espera que este prototipo sea desarrollado hasta llegar a ser una plataforma web completa, en que distintos usuarios puedan administrar los datos que se suben y luego los analicen usando las distintas visualizaciones generadas, sin tener que invertir tiempo en post procesamiento de datos o en recopilar datos obtenidos en herramientas separadas.

# Bibliografía


[1] Análisis de grafos de control de flujo. http://www.cs.columbia.edu/~suman/secure_sw_devel/Basic_Program_Analysis_DF.pdf. Accessed: 2020-05-09.

[2] Definición de complejidad ciclomática por wikipedia. https://en.wikipedia.org/wiki/Cyclomatic_complexity. Accessed: 2020-05-09.

[3] Dr. scratch. http://www.drscratch.org/. Accessed: 2020-05-09.

[4] Mulang. https://mumuki.github.io/mulang/. Accessed: 2020-07-1.

[5] Programadores: Escasez en chile y en el mundo. https://crealab.cl/programadores-escasez-chile-y-el-mundo/. Accessed: 2020-07-17.

[6] Scratch website. https://scratch.mit.edu/. Accessed: 2020-05-09.

[7] Bryce Boe, Charlotte Hill, Michelle Len, Greg Dreschler, Phillip Conrad, and Diana Franklin. Hairball: Lint-inspired static analysis of scratch projects. pages 215–220, 03 2013.

[8] Francisco Gutierrez, Jocelyn Simmonds, Cecilia Casanova, Cecilia Sotomayor, and Nancy Hitschfeld. Coding or hacking?: Exploring inaccurate views on computing and computer scientists among k-6 learners in chile. pages 993–998, 02 2018.

[9] Francisco J. Gutierrez, Jocelyn Simmonds, Nancy Hitschfeld, Cecilia Casanova, Cecilia Sotomayor, and Vanessa Peña Araya. Assessing Software Development Skills among K-6 Learners in a Project-Based Workshop with Scratch. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*, page 98–107. Association for Computing Machinery, 2018.

[10] Felienne Hermans and Efthimia Aivaloglou. Teaching software engineering principles to k-12 students: A mooc on scratch. pages 13–22, 05 2017.

[11] Jocelyn Simmonds, Francisco Gutierrez, Cecilia Casanova, Cecilia Sotomayor, and Nancy Hitschfeld. A teacher workshop for introducing computational thinking in rural and vulnerable environments. pages 1143–1149, 02 2019.

# Apéndice A

# Scratch

En la Figura A.1 se muestran las instrucciones disponibles en Scratch.


Figura A.1: Bloques disponibles en Scratch. Fuente: `https://www.dummies.com/programming/coding-with-scratch-blocks-and-scripts/`

# Apéndice B

# Project.json

En este apéndice, se incluye el código correspondiente al proyecto mostrado en el capitulo 2 de esta memoria, específicamente el ejemplo que aparece en la Figura 2.3.


```
{
  "targets": [
    {
      "isStage": true,
      "name": "Stage",
      "variables": {
        "`jEk@4|i[#Fk?(8x)AV.-my variable": [
          "my variable",
          "0"
        ]
      },
      "lists": {},
      "broadcasts": {},
      "blocks": {},
      "comments": {},
      "currentCostume": 0,
      "costumes": [
        {
          "assetId": "cd21514d0531fdffb22204e0ec5ed84a",
          "name": "backdrop1",
          "md5ext": "cd21514d0531fdffb22204e0ec5ed84a.svg",
          "dataFormat": "svg",
          "rotationCenterX": 240,
          "rotationCenterY": 180
        }
      ],
      "sounds": [
        {
          "assetId": "83a9787d4cb6f3b7632b4ddfebf74367",
          "name": "pop",
          "dataFormat": "wav",
          "format": "",
          "rate": 48000,
```

```
          "sampleCount": 1124,
          "md5ext": "83a9787d4cb6f3b7632b4ddfebf74367.wav"
        }
      ],
      "volume": 100,
      "layerOrder": 0,
      "tempo": 60,
      "videoTransparency": 50,
      "videoState": "on",
      "textToSpeechLanguage": null
    },
    {
      "isStage": false,
      "name": "Sprite1",
      "variables": {},
      "lists": {},
      "broadcasts": {},
      "blocks": {
        "N!%}4tQ@,:jXea=:GH!7": {
          "opcode": "event_whenflagclicked",
          "next": ")7G_L](k[kUxacvnU6NS",
          "parent": null,
          "inputs": {},
          "fields": {},
          "shadow": false,
          "topLevel": true,
          "x": 333,
          "y": 169
        },
        ")7G_L](k[kUxacvnU6NS": {
          "opcode": "motion_movesteps",
          "next": "KbYtt-N.jT#_Vq/k=dgI",
          "parent": "N!%}4tQ@,:jXea=:GH!7",
          "inputs": {
            "STEPS": [
              1,
              [
                4,
                "10"
              ]
            ]
          },
          "fields": {},
          "shadow": false,
          "topLevel": false
        },
        "KbYtt-N.jT#_Vq/k=dgI": {
          "opcode": "sound_play",
          "next": "%[fh2/M{r01ybjo^mZ.q",
          "parent": ")7G_L](k[kUxacvnU6NS",
          "inputs": {
            "SOUND_MENU": [
```

```
              1,
              "w:QDybqyktIed(Ab+MIa"
            ]
          },
          "fields": {},
          "shadow": false,
          "topLevel": false
        },
        "w:QDybqyktIed(Ab+MIa": {
          "opcode": "sound_sounds_menu",
          "next": null,
          "parent": "KbYtt-N.jT#_Vq/k=dgI",
          "inputs": {},
          "fields": {
            "SOUND_MENU": [
              "Meow",
              null
            ]
          },
          "shadow": true,
          "topLevel": false
        },
        "%[fh2/M{r01ybjo^mZ.q": {
          "opcode": "looks_say",
          "next": "$dK.C,Zwrk,zVTENIoI(",
          "parent": "KbYtt-N.jT#_Vq/k=dgI",
          "inputs": {
            "MESSAGE": [
              1,
              [
                10,
                "Hola!"
              ]
            ]
          },
          "fields": {},
          "shadow": false,
          "topLevel": false
        },
        "$dK.C,Zwrk,zVTENIoI(": {
          "opcode": "sensing_setdragmode",
          "next": "E%fOSdKj3s=,BD+cg+kz",
          "parent": "%[fh2/M{r01ybjo^mZ.q",
          "inputs": {},
          "fields": {
            "DRAG_MODE": [
              "draggable",
              null
            ]
          },
          "shadow": false,
          "topLevel": false
```

```
        },
        "E%fOSdKj3s=,BD+cg+kz": {
          "opcode": "data_setvariableto",
          "next": null,
          "parent": "$dK.C,Zwrk,zVTENIoI(",
          "inputs": {
            "VALUE": [
              1,
              [
                10,
                "0"
              ]
            ]
          },
          "fields": {
            "VARIABLE": [
              "my variable",
              "`jEk@4|i[#Fk?(8x)AV.-my variable"
            ]
          },
          "shadow": false,
          "topLevel": false
        },
        ".L[~cJ4?8M:+A:+:uDJ0": {
          "opcode": "event_whenkeypressed",
          "next": ";W(xESf15Ei|!msEa^?p",
          "parent": null,
          "inputs": {},
          "fields": {
            "KEY_OPTION": [
              "space",
              null
            ]
          },
          "shadow": false,
          "topLevel": true,
          "x": 833,
          "y": 159
        },
        "Ia`T],6KZ!jzdh.h~.O;": {
          "opcode": "sensing_touchingobject",
          "next": null,
          "parent": ";W(xESf15Ei|!msEa^?p",
          "inputs": {
            "TOUCHINGOBJECTMENU": [
              1,
              "k$f`:A:N$TV1!Mbb3(;,"
            ]
          },
          "fields": {},
          "shadow": false,
          "topLevel": false
```

```
        },
        "k$f`:A:N$TV1!Mbb3(;,": {
          "opcode": "sensing_touchingobjectmenu",
          "next": null,
          "parent": "Ia`T],6KZ!jzdh.h~.O;",
          "inputs": {},
          "fields": {
            "TOUCHINGOBJECTMENU": [
              "_mouse_",
              null
            ]
          },
          "shadow": true,
          "topLevel": false
        },
        "~jjTN-s9:z*:%.+Fk*A^": {
          "opcode": "looks_cleargraphiceffects",
          "next": null,
          "parent": ";W(xESf15Ei|!msEa^?p",
          "inputs": {},
          "fields": {},
          "shadow": false,
          "topLevel": false
        },
        ";W(xESf15Ei|!msEa^?p": {
          "opcode": "control_if_else",
          "next": null,
          "parent": ".L[~cJ4?8M:+A:+:uDJ0",
          "inputs": {
            "CONDITION": [
              2,
              "Ia`T],6KZ!jzdh.h~.O;"
            ],
            "SUBSTACK": [
              2,
              "~jjTN-s9:z*:%.+Fk*A^"
            ],
            "SUBSTACK2": [
              2,
              "fL+Wr.h]}$9!1Mh(~(mr"
            ]
          },
          "fields": {},
          "shadow": false,
          "topLevel": false
        },
        "fL+Wr.h]}$9!1Mh(~(mr": {
          "opcode": "control_repeat_until",
          "next": null,
          "parent": ";W(xESf15Ei|!msEa^?p",
          "inputs": {
            "CONDITION": [
```

```
              2,
              "%!6u#lYJcO$Ek2r`g9^S"
            ],
            "SUBSTACK": [
              2,
              "jO_!{tzv}5oeL.tW]L|i"
            ]
          },
          "fields": {},
          "shadow": false,
          "topLevel": false
        },
        "%!6u#lYJcO$Ek2r`g9^S": {
          "opcode": "operator_gt",
          "next": null,
          "parent": "fL+Wr.h]}$9!1Mh(~(mr",
          "inputs": {
            "OPERAND1": [
              3,
              "(YU6_zDf?3-d7)8`tG%-",
              [
                10,
                ""
              ]
            ],
            "OPERAND2": [
              1,
              [
                10,
                "5"
              ]
            ]
          },
          "fields": {},
          "shadow": false,
          "topLevel": false
        },
        "(YU6_zDf?3-d7)8`tG%-": {
          "opcode": "operator_random",
          "next": null,
          "parent": "%!6u#lYJcO$Ek2r`g9^S",
          "inputs": {
            "FROM": [
              1,
              [
                4,
                "1"
              ]
            ],
            "TO": [
              1,
              [
```

```
                4,
                "10"
              ]
            ]
          },
          "fields": {},
          "shadow": false,
          "topLevel": false
        },
        "jO_!{tzv}5oeL.tW]L|i": {
          "opcode": "sound_play",
          "next": null,
          "parent": "fL+Wr.h]}$9!1Mh(~(mr",
          "inputs": {
            "SOUND_MENU": [
              1,
              "pr*hP,Eo2BqKaSSr|yyK"
            ]
          },
          "fields": {},
          "shadow": false,
          "topLevel": false
        },
        "pr*hP,Eo2BqKaSSr|yyK": {
          "opcode": "sound_sounds_menu",
          "next": null,
          "parent": "jO_!{tzv}5oeL.tW]L|i",
          "inputs": {},
          "fields": {
            "SOUND_MENU": [
              "Meow",
              null
            ]
          },
          "shadow": true,
          "topLevel": false
        }
      },
      "comments": {},
      "currentCostume": 0,
      "costumes": [
        {
          "assetId": "bcf454acf82e4504149f7ffe07081dbc",
          "name": "costume1",
          "bitmapResolution": 1,
          "md5ext": "bcf454acf82e4504149f7ffe07081dbc.svg",
          "dataFormat": "svg",
          "rotationCenterX": 48,
          "rotationCenterY": 50
        },
        {
          "assetId": "0fb9be3e8397c983338cb71dc84d0b25",
```

```
          "name": "costume2",
          "bitmapResolution": 1,
          "md5ext": "0fb9be3e8397c983338cb71dc84d0b25.svg",
          "dataFormat": "svg",
          "rotationCenterX": 46,
          "rotationCenterY": 53
        }
      ],
      "sounds": [
        {
          "assetId": "83c36d806dc92327b9e7049a565c6bff",
          "name": "Meow",
          "dataFormat": "wav",
          "format": "",
          "rate": 48000,
          "sampleCount": 40682,
          "md5ext": "83c36d806dc92327b9e7049a565c6bff.wav"
        }
      ],
      "volume": 100,
      "layerOrder": 1,
      "visible": true,
      "x": 20,
      "y": 0,
      "size": 100,
      "direction": 90,
      "draggable": true,
      "rotationStyle": "all around"
    }
  ],
  "monitors": [],
  "extensions": [],
  "meta": {
    "semver": "3.0.0",
    "vm": "0.2.0-prerelease.20201125065300",
    "agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:84.0) Gecko
       /20100101 Firefox/84.0"
  }
}
```


Código B.1: Archivo “project.json” para bloque “if else”

# Apéndice C

# Endpoints

En la Tabla C.1 se presentan todos los endpoints expuestos de la API, los cuales son consumidos por la Interfaz Web. En la columna parámetros, aquellos que inician con mayúscula corresponden a instancias del modelo de datos.

| Url | Request | Parámetros | Acción/Respuesta |
|---|---|---|---|
| /courses | GET | - | Listar Cursos |
| /courses | POST | semester, year | Crear un Curso |
| /courses/id/ | DELETE | - | Borrar un Curso |
| /projects | GET | - | Listar Proyectos |
| /upload | POST | Course, file(s) | Analizar uno o más proyectos |
| /data | GET | Project | Obtener análisis de un proyecto |
| /plot | GET | Project | Obtener datos de GFGs de un proyecto. |
| /course/<id>/all_files | GET | - | Obtener todos los archivos de un curso |
| /course/<id>/export_rubrics | GET | - | Obtener la tabulación de todas las rúbricas de un curso. |
| /project/<id>/rubric | GET | - | Obtener los datos de la rúbrica de un curso |
| /project/<id>/file | GET | - | Obtener el archivo original de un proyecto |

Tabla C.1: Endpoints expuestos de la aplicación.