Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Análisis, diseño e implementación de importador de desafíos para Pilas Bloques

Tesis de Licenciatura en Ciencias de la Computación

Tomás Freilij

Director: Ing. Alfredo Sanzo (UTN.BA, CONICET-UBA ICC, Fundación Sadosky)

Codirector: Dr. Fernando Schapachnik (CONICET-UBA ICC, Fundación Sadosky)

Buenos Aires, 2022

ABSTRACT

En la enseñanza de la programación se presentan propuestas de trabajo utilizando distintas herramientas de software. Pilas Bloques es una de ellas, y propone el trabajo orientado a desafíos y la programación por bloques. El hecho de que esté en español, que admita el trabajo o□ine, y que posea determinados aspectos estéticos, la convierten en una buena opción para la enseñanza en los niveles de primaria e inicios de la secundaria en la Argentina. Sin embargo, a pesar de ser de código abierto, Pilas Bloques no permite la producción de nuevos ejercicios por parte de les docentes que no estén dispuestos a modificar y recompilar el código de la aplicación o no sean capaces de hacerlo. Esta tesis propone encarar la resolución de este problema, dándole a docentes la posibilidad de elaborar nuevos ejercicios de forma independiente y sin necesidad de contar con conocimientos de la implementación de Pilas Bloques. Para llegar a esto, se elaboró un análisis, diseño y modificación de la herramienta. Las modificaciones desarrolladas están integradas ya en el código de la herramienta, y en un futuro se encontrará en el ambiente de producción y disponible para su uso por docentes y estudiantes.

Palabras claves: Pilas Bloques, introducción a la programación, primaria, programación por bloques, trabajo por desafíos, programación en argentina.

ABSTRACT

In the teaching of programming, work proposals are presented using di□erent software tools. Pilas Bloques is one of them, and it proposes scenario-based challenges and block programming. The fact that it is in Spanish, that it supports o□ine work, and that it has certain aesthetic aspects, makes it a good option for teaching in primary and early secondary levels in Argentina. However, Pilas Bloques doesn't have the feature that allows dynamically adding a new challenge. The tool is Open Source, but teachers can't add a new challenge without modifying the code and recompiling the app. This thesis' goal is to address the resolution of this problem, giving teachers the possibility of developing new exercises by themselves and without needing any programming skills or background on the implementation of Pilas Bloques. Therefore, an analysis, design and modification of Pilas Bloques was elaborated. The new features are already integrated in the tool code, and in a not far future they will be found in the production environment sand available for use by teachers and students.

Keywords: pilas bloques, computer programming foundations, primary school, block programming, scenario-based, computer programming in argentina.

AGRADECIMIENTOS

Quiero agradecer a toda mi familia y amigos. Va un reconocimiento particular a la Escuela Técnica de la UBA sin la cual creo que la propuesta de esta tesis no la hubiese tomado con la iniciativa que se merece.

Índice general

1.. Introducción 1

2.. Estado del arte y dimensiones de análisis 3
2.1. Características que analizaremos 3
2.2. Armado del programa: Lenguajes textuales y con bloques 4
2.3. Propuestas didácticas: herramientas “open ended” y orientadas a desafíos . . 8
2.4. Requerimientos de conectividad 13
2.5. Nivel educativo 14
2.6. Pilas Bloques en el contexto de estas herramientas 15

3.. Planteo del problema 16

4.. Estructura de un desafío en Pilas Bloques 17
4.1. Mecánica de un desafío en Pilas Bloques 17
4.2. Armado y ejecución de un programa 17
4.3. Objetos interactuables 17
4.4. Mensajes de error 19
4.5. Obstáculos 20
4.6. Mensaje de finalización/felicitación 21

5.. Clasificación de los desafíos 22
5.1. Estructura del escenario 23
5.1.1. Escenarios de grillas 23
5.2. Escenarios desestructurados 24
5.3. Disposición de obstáculos 25
5.3.1. Escenarios con obstáculos fijos 25
5.4. Escenarios con obstáculos variables 26
5.5. Disposición de los objetos interactuables 30
5.5.1. Objetos interactuables fijos 30
5.5.2. Objetos interactuables variables 31
5.6. Ejercicios con y sin precondiciones entre acciones 32
5.7. Cómo se cumple el objetivo (condición de finalización) 33
5.7.1. Caso de que todos los objetos fueron interactuados 34
5.7.2. Caso de “sucesión de estados” 34

6.. Hacia la implementación de la mejora 36
6.1. Preguntas de investigación 36
6.2. Recorte de ejercicios 37
6.3. Plantillas ya existentes en otras herramientas 37

7.. Retomando el análisis para un desafío nuevo 41
7.1. Instrucciones disponibles 42
7.2. Mensajes del autómata y felicitación 44
7.3. Información para le estudiante 45
7.4. Imágenes y animaciones 45
7.5. Tamaño y forma de la grilla 45

8.. Síntesis de los aspectos anteriores 46

9.. Siguiente etapa: contexto de trabajo con Pilas Bloques 47
9.1. De nición básica de un desafío 47
9.2. ¾Qué aspectos de este lenguaje de con guración se deberían extender? . . . 50
9.3. Extensión del lenguaje de con guración 52
9.4. Animaciones e imágenes 54
9.5. Formato del directorio 60

10..Primeros pasos de la implementación 62
10.1. Modi car el tipo de archivo que importa 62
10.2. Modi car los campos que toma del JSON 62
10.3. Detalles de implementación 64
10.4. Resultados obtenidos 66
10.5. Más avances en el desarrollo y nuevas actividades para agregar 69
10.6. Desafíos propuestos 74
10.7. Justi cación de las actividades propuestas 76
10.8. Armado de los desafíos con esta nueva mejora 76
10.8.1. Archivo desa o.json 77

11..Conclusión y próximos pasos 80

Anexo - 'La Programación y su didáctica - Plantilla de Diseño de Actividad' 84
11.1. Descripción de Ficha para Diseño de Actividades - La Programación y su didáctica - Plantilla de Diseño de Actividad 84
11.2. Ficha para Diseño de Actividad 1 86
11.3. Ficha para Diseño de Actividad 2 88

1. INTRODUCCIÓN

En el ambiente de la docencia primaria y secundaria existen nuevos desafíos que se introdujeron a raíz del crecimiento del mundo digital.

Surge la necesidad de introducir de manera cada vez más temprana una formación de base que contemple no sólo el manejo de herramientas de uso cotidiano en una computadora promedio (alfabetización digital, o mática, etc.) sino que también permita visualizar el trasfondo de lo que sucede en las transformaciones digitales de nuestra vida cotidiana. Esta necesidad se ve re ejada en programas a nivel nacional en diferentes países [3] [5] y en Argentina con la iniciativa Program.AR [17]. En Argentina particularmente, el Consejo Federal de Educación declaró mediante su resolución 263/15 que la enseñanza y el aprendizaje de programación es de importancia estratégica para fortalecer el desarrollo económico-social de la Nación, y estableció mediante su resolución 343/18 un plazo de dos años para que cada una de las provincias argentinas tome acciones tendientes a la incorporación de pensamiento computacional, programación y robótica a sus currículas.

Particularmente para enseñar programación existen herramientas que tienen como objetivo introducir a estudiantes en ese mundo y esta tesis se enfocará en trabajar en una propuesta de mejora para una de estas.

La herramienta que elegimos para el trabajo es Pilas Bloques [20], un software desarrollado por la iniciativa Program.AR [17] de la Fundación Sadosky [19]. La misma se presenta como apta para la enseñanza de la programación en el nivel de enseñanza primaria y media del sistema educativo formal, haciendo eje en los niveles en los que ya se cuenta con habilidades básicas de lectoescritura. A junio de 2022, Pilas Bloques cuenta con más de medio millón de usuarios anuales, con alrededor de cinco mil usuarios diarios (en días escolares), distribuidos en un 65 % en Argentina y el resto en otros países como Uruguay, México, Colombia, Costa Rica y más[1]. Además, posee versiones instalables de la aplicación para aquellas regiones sin conectividad. Propone una serie de desafíos y una estrategia didáctica asociada, materializada en un cuadernillo[10]. Cuenta además con un manual para docentes [8] y múltiples cursos de capacitación [16].

La tesis actual se sustenta en la experiencia de cuatro años de enseñanza con la herramienta en los talleres de programación del primer año de la Escuela Técnica de la UBA de Villa Lugano[2], donde surge hacia el nal de cada curso la necesidad de tener un conjunto mayor de actividades para la ejercitación con Pilas Bloques. Ya sea como ejercitación complementaria para aquelles estudiantes que lo necesiten o como mecanismo de evaluación con desafíos distintos a la práctica, es necesario tener nuevas guías y ejercicios en la herramienta.

Como parte de la realización de la tesis se mantuvo contacto estrecho con la iniciativa Program.AR y la Fundación Sadosky, y también con el equipo de desarrollo de Pilas Bloques, quienes aportaron especi cidad al problema arriba mencionado. Les docentes a lo largo de Argentina y otros países necesitan que sea quien usa la aplicación quien pueda crear los nuevos desafíos, de forma similar a otras herramientas existentes. Es decir, que sean estudiantes y docentes quienes puedan agregarlos, y que no sólo vengan dados .

Pilas Bloques es de código abierto, así que les docentes podrían modi car el código y

[1] Datos provistos por el equipo de Program.AR
[2] http://www.etec.uba.ar/

recompilarla para conseguirlo, pero muy pocos docentes tienen el tiempo o el conocimiento especí co de las tecnologías sobre las que está implementada. Pero sobretodo no deberían necesitar nada de eso para poder crear nuevos desafíos. Esta tesis es el resultado de dar un primer paso hacia la implementación de esta posibilidad.

Se hizo un análisis de requisitos, un relevamiento de las soluciones existentes a problemas similares en otras herramientas existentes, se analizó técnicamente cómo Pilas Bloques resuelve la creación de actividades, y con eso se de nió un conjunto de características necesarias mínimas para la implementación de un creador de desafíos. Se realizó nalmente la implementación de una parte de todo el proceso: la importación desde un archivo externo, con una estructura y un diseño propuestos por la tesis.

Gracias al trabajo de esta tesis ahora es posible agregar desafíos a Pilas Bloques con un simple conocimiento de edición de un JSON y creación de un archivo comprimido, conocimientos al alcance de un docente promedio.

La estructura de este documento es la siguiente:

- Primero se presentará un análisis que permitirá situar a Pilas Bloques en el contexto de la enseñanza de la programación a modo de estado del arte. Esto permitirá introducir algunos conceptos y nociones que luego usaremos en los análisis posteriores.
- Luego se hará un planteo de problema que aclara el objetivo al que queremos llegar en este trabajo. Se hará un análisis de Pilas Bloques, los tipos de problemas que se pueden abordar, y cuáles son sus características principales que nos permitirán aclarar mejor el panorama para avanzar en el desarrollo.
- Seguido de esto, se presentará un diseño y propuesta para la importación de nuevos desafíos. A partir de la misma se presentará una implementación que mostrará los grados de avances que se pudieron lograr con la presente tesis.
- Por último, haremos un cierre a modo de balance y para presentar posibles trabajos a futuro.

2. ESTADO DEL ARTE Y DIMENSIONES DE ANÁLISIS

A modo de contextualización de Pilas Bloques, lo primero que haremos es presentar un conjunto de características que consideramos centrales a la hora de evaluar qué herramienta de enseñanza usar en el aula. Como los ejes de análisis pueden ser muchos, nos remitiremos a los aspectos que nos interesan a la hora de pensar la enseñanza en los niveles para los que está pensado el uso de Pilas Bloques. Esto es, estudiantes de primaria y primeros años de secundaria que hayan alcanzado un nivel de alfabetización básica. Además estas escuelas podrán ser tanto de orientación técnica especí ca como de enseñanza media general.

En cada caso, el análisis irá acompañado de ejemplos de herramientas que tienen esas características descritas. Como el conjunto de aplicaciones educativas es tan grande, intentaremos utilizar un subconjunto que sea representativo.

A modo de anticipo, estas herramientas serán.

1. Arduino [2]
2. Scratch [18]
3. Pilas Bloques [20]
4. Alice [7]
5. Gobstones [15]

2.1. Características que analizaremos

Como ya dijimos anteriormente, existen muchas dimensiones a considerar a la hora de elegir una herramienta para la enseñanza de los conceptos básicos de programación. Cada una tiene ventajas y desventajas de acuerdo a las expectativas que tiene le docente para con su curso y también de la edad de les estudiantes.

Por el contexto que planteamos anteriormente, nosotros nos enfocaremos en las siguientes dimensiones.

- La forma en la que les estudiantes deben armar los programas.
- El idioma de la interfaz de la aplicación.
- La modalidad de trabajo que proponen estas herramientas.
- Las posibilidades de trabajo con o sin conexión a internet que proveen las mismas.
- El conjunto de estudiantes a los que se orienta la herramienta.

2.2. Armado del programa: Lenguajes textuales y con bloques

Los lenguajes textuales son los que presentan la necesidad de escribir sus instrucciones con determinadas reglas sintácticas. Un ejemplo de estos lenguajes son C++, Java o Python.

Arduino es el caso de una herramienta que incorpora este tipo de programación.

Fig. 2.1: Algunas herramientas educativas requieren conocimientos de electrónica y armado de circuitos (Diagrama hecho con Fritzing)

Arduino en particular propone además enseñar conceptos de electrónica. En la Fig. 2.1 se puede ver un circuito típico que se suele armar para introducir en el uso de la placa. La placa Arduino UNO irá conectada a una computadora mediante un cable USB para obtener la alimentación. Tendrá un conjunto de cables más chicos que están conectados a los pines de la placa en un extremo y en el otro a lo que suele llamarse protoboard (una placa con conexiones internas que permite conectar componentes entre ellos) que es muy utilizada en contextos educativos por la facilidad que provee para generar las conexiones.

Los pines del Arduino permiten, entre otras cosas, controlar la tensión que se le aplica al circuito. Sin meternos en el detalle de la electrónica, si se quiere que el led rojo titile, se podría hacer un programa de la siguiente forma escrito en pseudocódigo.

```
Repetir por siempre {
    Aplicar una tension de 5v en el pin
    Esperar un segundo
    Aplicar una tension de 0v en el pin
    Esperar un segundo
}
```

Para escribir este programa que acabamos de mostrar, podemos descargar de forma gratuita una IDE que nos facilita la escritura de este programa, pero que debe hacerse utilizando el lenguaje C++.

Aquí vemos el programa que implementa el pseudocódigo anterior. Les estudiantes deben respetar reglas sintácticas como el uso de ; al nal de cada instrucción, el balanceo de paréntesis y el uso de una sintaxis bien de nida. En muchos casos además algunas palabras serán macros de C++ que deberán ser escritas respetando las mayúsculas sin excepción.

Como se muestra en la Fig. 2.2, Arduino IDE cuenta con el resaltado de sintaxis, lo cual ayuda a les estudiantes, pero a veces esto no alcanza para que eviten los errores.

Fig. 2.2: Arduino IDE. Hay resaltado de sintaxis pero esto no evita errores.

Veamos lo que plantean varios autores respecto al uso de lenguajes con reglas sintácticas para la enseñanza de la introducción a la programación. En principio, se señala el problema del nivel de abstracción necesario para lidiar con editores de texto:

> For Greenfoot, we aim at users aged from 13 upwards. Below this age, many children do not have the necessary typing and abstraction capabilities to cope with the text-based editor and resulting syntax errors. Languages avoiding this type of syntax, most notably block-based languages, are usable at a much lower age range and can better focus on the initial learning of fundamental concepts. Using a standard text-based language has an advantage only when the learning of syntax and text-based programming is an explicit goal of the learning process [14]

Más especí camente, se señalan los problemas para desarrollar algoritmos y encontrar errores.

> Many students struggle with learning how to write software (Bonar & Soloway, 1983; Mow, 2008; Lahtinen, Ala-Mutka, Jarvinen, 2005; Robins, Roundtree, & Roundtree, 2003). Students need to learn the syntax and semantics of a programming language as well how to connect the two in order to solve a problem (Soloway, 1986). This process can be frustrating for students. Successful programmers tend to be logical, analytical, and detail-oriented (Daly, 2010); this a ords them the ability to develop algorithms, synthesize problems, and troubleshoot errors.
>
> In order to execute a program, you must adhere to the syntax of the language; any slight deviation from the rules of the language (improper capitalization, misspelling, missing or wrong number of braces, quotes, semicolons, brackets, and/or parenthesis) and the program will not run.
>
> This can be overwhelming for students just learning how to program. Ultimately, they will need to be problem solvers, to think abstractly, to be detail-oriented, and to be able to communicate with peers in order for their code to tie into someone else's code, but this is too much to take on at one time. [9]

Inclusive en casos donde contamos con librerías que nos resuelven necesidades, es necesario escribir código respetando reglas.

> The use of graphics libraries is likely the most complex approach. Even though libraries are provided, students still must write Java/C++ programs from scratch, mastering a non-trivial amount of syntax (regardless whether they understand the semantics of what they are writing). Then they need to understand the particulars of the graphics library. Karel J. Robot has a fair bit of Java that needs

> to be mastered before being able to write a program. The BlueJ and JPT approaches are somewhat simpler, as students only modify existing code. Yet, it is still necessary to write correct Java code, and certain errors (such as missing brackets or trying to place code in the wrong location, or invoking a method with a bad parameter) can lead to errors in the code provided to the student and the student may not know how to start debugging code that he/she did not write [7]

Por ejemplo, en el contexto de la enseñanza de la Programación Orientada a Objetos, proponer que deban asimilar conceptos propios del paradigma, sumado a la correcta escritura de código, puede ser un problema.

> The classic approach allows a somewhat gentle learning curve, providing time for the learner to assimilate and build knowledge incrementally. An objects- rst strategy is intended to have students work immediately with objects. This means students must dive right into classes and objects, their encapsulation (public and private data, etc.) and methods (the constructors, accessors, modi ers, helpers, etc.). All this is in addition to mastering the usual concepts of types, variables, values, and references, as well as with the often-frustrating details of syntax [7]

Es por esto que consideramos deseable trabajar con lenguajes que abstraen de la di cultad de la sintaxis y permitan enfocarse en la solución nal del problema. Veamos entonces un ejemplo de un ejercicio en Pilas Bloques, aplicación que admite la programación por bloques.

Fig. 2.3: Presentación de un desafío en Pilas Bloques

En la gura Fig. 2.3 a la izquierda podemos ver un escenario donde el personaje empieza en la casilla en el extremo izquierdo y debe avanzar comiendo los churrascos (objetos con los que puede interactuar el autómata). El objetivo es escribir un programa que permita comer todos los churrascos sin importar en dónde aparezcan los mismos.

En el medio hay un toolbox donde le estudiante puede ver las instrucciones con las que cuenta para resolver el ejercicio y a la derecha el programa que va armando.

Para de nir el programa que se ejecutará, Pilas Bloques les permite arrastrar desde el toolbox hasta el bloque principal e ir construyendo así el programa, agregando y modi cando los bloques ahí presentes. En la Fig. 2.4 hay un ejemplo de programa con bloques.

Fig. 2.4: Ejemplo de programa que se podría plantear para el desafío de los churrascos

Con la programación por bloques la posibilidad de errores por un incorrecto uso de la sintaxis disminuye drásticamente porque la modalidad de arrastrar bloques evita la necesidad de tener que escribir las instrucciones a mano. Esto permite entonces poder focalizar mejor la atención en la solución del problema.

Algunas ventajas adicionales que vemos en comparación a la sintaxis que se propone para trabajar con Arduino son.

1. Cada bloque tiene un color y es fácilmente distinguible del resto.

2. Sumado a eso, cada bloque es una representación grá ca e interactiva de un concepto de programación. Los bloques naranjas suelen estar asociados a lo que tradicional-

mente se considera control de ujo de ejecución, los bloques con enganche a izquierda son expresiones, los que tienen enganche superior e inferior son comandos, y así podemos seguir mencionando detalles. En esa asociación está la potencia didáctica de la enseñanza con bloques.

3. Las instrucciones disponibles están en un toolbox desde donde pueden arrastrar, de modo que no deben acordarse de memoria cómo se escribe cada instrucción.

4. En algunos casos se pueden establecer con guraciones adicionales que permiten que el texto dentro de los bloques se adapte a problemáticas de lectoescritura que puedan tener les estudiantes.

2.3. Propuestas didácticas: herramientas open ended y orientadas a desafíos

Llamaremos aplicaciones open ended (o de mundo abierto) a las que proponen seguir la dinámica de trabajo por proyectos. En las mismas no hay un desafío/ecuación/pregunta de nido a priori que tenga una solución o soluciones ideales, sino que se dispone de múltiples opciones, personajes y comandos para crear un proyecto particular, partiendo en general de un lienzo en blanco o de un proyecto empezado.

Existen dos aplicaciones muy conocidas que permiten este enfoque, Scratch y Alice. En la Fig. 2.5 se muestra cómo se ve Scratch, una de las más utilizadas.

Fig. 2.5: Interfaz de Scratch. La misma presenta similitudes y diferencias con Pilas Bloques

1. En la derecha tenemos el escenario que por defecto empieza con un gato como autómata. A diferencia de Pilas Bloques, podemos agregar nuevos personajes que, en la aplicación, se llamarán Objetos.

2. A la izquierda contamos con un toolkit de instrucciones similar al de Pilas Bloques. Pero en este caso, les estudiantes tienen dominio de muchos más aspectos. Por ejemplo pueden modi car las imágenes de fondo, aportar animaciones y sonido a los objetos y sus acciones.

Fig. 2.6: Toolkit de edición de sonido en Scratch

Fig. 2.7: En esta interfaz les estudiantes pueden dar nuevos disfraces o posiciones a los personajes/objetos, pudiendo así crear animaciones para los mismos.

Fig. 2.8: En esta imagen mostramos el toolkit de instrucciones que permite usar Scratch.

Como podemos ver en las Fig. 2.6, Fig. 2.7 y la Fig. 2.8, la aplicación provee un conjunto bastante amplio de aspectos a manipular, lo que incluye movimientos, sonidos, interacciones entre personajes, interacción con el usuario, dibujo de escenarios de fondo, etc.

Diversas experiencias docentes mencionan el problema que resulta lo abrumador del conjunto de bloques disponibles.

Alice también presenta una modalidad de trabajo open-ended y permite elaborar escenarios y animaciones 3D. Suele además estar asociada a la enseñanza del Paradigma Orientado a Objetos utilizando programación por bloques, pero cuenta con un mecanismo para migrar al lenguaje textual.

> Alice is another well-known tool, very similar in approach to Scratch: it also allows the building of complex scenarios and animations (although in this case those are 3D), and experiment using programming with blocks. Consequently its similarities and di erences with Pilas Bloques are equivalent to those with Scratch. Besides the 3D scenarios, the main di erences with other tools is that Alice is leaned towards teaching object oriented programming and allows a migration path from blocks to textual code. [20]

Y por último, otra característica sumamente relevante para esta tesis es que ninguno de los dos acota los bloques y aspectos con gurables de cada proyecto. Es decir que en todo momento les estudiantes tendrán a disposición la totalidad de los bloques y aspectos con gurables que antes mencionamos.

Esto las convierte en opciones ideales para la enseñanza basada por proyectos. De hecho, en el trabajo Teaching Introductory Programming Concepts: A Comparison of Scratch and Arduino [4] se hace una comparación de la enseñanza con Scratch y Arduino en una escuela. Las encuestas que realizaron lanzaron que les estudiantes consideran que Scratch es ideal para programar juegos.

Como alternativa a la propuesta open ended, tenemos las herramientas orientadas a desafíos . En estas, el trabajo está organizado como una secuencia de actividades en las que se presenta un enunciado y les estudiantes desarrollan una solución a la misma. En este caso, los objetivos son mucho más puntuales y acotados: no es necesario construir o pensar un dominio, sino que está dado, y hay un goal como comer todas las manzanas , o resolver el problema de prender todas las luces independientemente de la posición donde se encuentren . Estas actividades con objetivos acotados pueden pensarse como desafíos .

En el siguiente extracto se puede ver los lineamientos que siguió Fundación Sadosky para el desarrollo de Pilas Bloques

> For the concrete implementation of our inquiry-based strategy, we provide a set of xed scenarios, each one with a de ned goal. In order to complete each goal the student needs not only to apply previous knowledge, but also needs an ideally-small new concept that she does not still possess. Once the student reaches the problem, tries to solve it with her current knowledge, and formulates the question, then and not before the teacher introduces the concept[14]

Lo anterior se puede resumir en.

1. Pilas Bloques ofrece escenarios prede nidos y con propiedades jas.
2. Cada escenario tendrá un objetivo claro y bien de nido.

Existen dos elementos más que se suelen contemplar a la hora de trabajar con herramientas orientadas a desafíos.

1. La posibilidad de que las instrucciones disponibles para cada problema sean un conjunto acotado que permita inducir al estudiante en el armado de determinadas soluciones. Esto lo contraponemos con las open ended en el sentido de que les estudiantes cuentan con un conjunto de instrucciones que probablemente excede a lo que se necesita para resolver el problema.
2. Esta mecánica permite además que la herramienta provea feedback automatizado sobre la solución (por ejemplo si resuelve o no el problema), en contraposición con las herramientas open ended que por su naturaleza no pueden hacerlo.

Esta última estrategia la adoptaron les desarrolladores de Pilas Bloques en el trabajo antes citado:

> Task-completion, i.e., the tool notifying the student that she nished a challenge, is key. In the case of Pilas Bloques, to stress that achieving the goal might not mean that the program is fully correct (i.e., the student could have used a sentence many times instead of using a repetition control structure), we output a carefully crafted message. It reads: Goal achieved! However, your program might or might not be right. Discuss it with your classmates and teachers to know if you really found an idea that is useful for other tasks. . We plan to use program analysis techniques to provide more accurate feedback in the future. [20]

Además de Pilas Bloques, otros ejemplos de herramientas que admiten el trabajo orientado a desafíos son Gobstones [15], Mumuki [1] y Code.org [13].

Gobstones [15] merece una consideración especial pues presenta en algunos ejercicios la modalidad con Desafíos y también admite la posibilidad de armar ejercicios open ended para que les estudiantes puedan experimentar.

2.4. Requerimientos de conectividad

Otro elemento importante a la hora de evaluar características de una aplicación es la posibilidad de poder trabajar sin una conexión permanente a internet.

La relevancia de esto radica en que en Argentina todavía existe una gran población y escuelas que no cuentan con una conectividad estable y de calidad, lo cual no brinda un escenario propicio de aprendizaje en algunos casos.

En este aspecto, podremos encontrar distintos tipos de aplicaciones, algunas exigen una conexión rápida y permanente, mientras que otras pueden trabajar de forma completamente o ine, descargando previamente un cliente instalable. Ejemplo de esto último es la versión de escritorio de Scratch que, una vez instalada, permite tener los proyectos

y desarrollos alojados en la computadora local sin necesidad de contar con conectividad necesariamente. Otra aplicación que permite un trabajo similar es Pilas Bloques.

Lo que nos parece relevante, es lo que plantea el artículo de Fundación Sadosky sobre ciertas características que debe cumplir una aplicación que pretenda tener alcance masivo en la Argentina.

> The CS teaching e ort had to be built on top of Conectar Igualdad, the Argentine nationwide one-to-one computer program, with more than 5.5 million netbooks delivered to high-school students, and many local initiatives also making netbooks available at primary schools. Although these 1 to 4 GB netbooks can be considered widely available in Argentine schools, Internet connectivity cannot, meaning that an o ine mode was mandatory. Also, the tool had to be in Spanish, and be available free of charge (being open-source was a plus). More importantly, it had to support the pedagogical methodology developed by the team over the years. [20]

2.5. Nivel educativo

La enseñanza en los primero niveles del sistema educativo implica una adaptación acorde a las capacidades y habilidades adquiridas hasta ese momento en la vida de les estudiantes. El acceso a la formación en el idioma inglés desde edades tempranas es algo muy heterogéneo en la población argentina. Por consiguiente, debemos tomar una visión general que asuma que el público al que está dirigido estas herramientas no cuenta con un nivel de inglés razonable para trabajar con cierta autonomía.

Así como evaluamos que la programación textual puede ser una barrera para el aprendizaje en los niveles iniciales, de la misma manera el idioma puede serlo, como plantea Fundación Sadosky en el trabajo ya citado anteriormente:

> Some of the pillars of our pedagogical strategy are well known criteria like inquiry based learning and native language use. In our case that means that whatever tool is used needs to be available in Argentine Spanish, so that language does not become a learning barrier. It also has to have a visually welcoming and children-suited design, which includes graphics and animations, and has to allow for programming through blocks, an increasingly popular strategy for avoiding syntax-related and conceptually shallow mistakes.[20]

Sumamos además como elemento la estética de la interfaz, el diseño de los personajes, y de los diálogos. Pilas Bloques además fue diseñado con la intención de contemplar estudiantes que todavía cuenten con problemas de lectoescritura, siendo más apto para el nivel primario.

2.6. Pilas Bloques en el contexto de estas herramientas

	Arduino	Gobstones	Alice	Scratch	Pilas Bloques
Propuesta didáctica orientada a desafíos	No	Sí	No	No	Sí
Admite programación por bloques	Sí	Sí	Sí	Sí	Sí
Admite trabajo o ine	Sí	Sí	Sí	Sí	Sí
Admite el trabajo con idioma español	No	Sí	Sí	Sí	Sí
Orientado a primaria y primeros años de secundaria	No	No	Sí	Sí	Sí

Tab. 2.1: Resumen de contextualización de Pilas Bloques

En la Tab. 2.1 se puede ver un resumen de dónde se para Pilas Bloques en relación a otras herramientas [6] [12] para enseñar a programar. El diseño propuesto más adelante en esta tesis está guiado por esta caracterización. Pilas Bloques es un lenguaje por bloques, o ine, en español y pensado para ser utilizado en las escuela primarias y secundarias de Argentina.

3. PLANTEO DEL PROBLEMA

Retomando experiencias de trabajo con Pilas Bloques, esta es una herramienta que a un grupo de colegas y a mí nos permitió encarar la enseñanza de la programación en los primeros años de la escuela secundaria técnica. Al recorrer la guía de desafíos, les estudiantes desarrollan un pensamiento computacional que luego se podrá traducir en producciones de mayor complejidad en años posteriores.

Sin embargo, propuestas como una guía ja de ejercicios pueden ser limitantes a la hora de introducirse en un espacio áulico donde además se tienen que poder medir y evaluar avances. Por ejemplo, si une estudiante presenta mucha di cultad para resolver un conjunto de ejercicios y los mismos los pudo terminar con ayuda docente.¾Cómo podemos evaluar o medir el grado de avance al llegar los momentos de evaluación? ¾Tiene valor que une estudiante muestre que puede resolver un desafío para el cual ya vio una solución?.Sería ideal presentar un ejercicio distinto a los ya vistos, pero similar en estructura y forma de solución, y de esta forma podremos ver si logró o no un avance en su autonomía para resolver ese tipo de ejercicios.

Con esto presentado, el trasfondo del problema es que la aplicación carece de la posibilidad de que un docente sin conocimientos de programación avanzada pueda crear cualquier tipo de desafío nuevo. Se pierde entonces la posibilidad de poder generar de manera simple actividades que sirvan como refuerzo para estas instancias antes descritas. Esto signi ca que, de ser necesarios nuevos ejercicios, deben esperar al menos un nuevo release. A eso sumemos el hecho de que en un nuevo release, les estudiantes siempre tendrán a disposición todos los ejercicios que se les podría pedir que resuelvan, dejando de lado la posibilidad de usarlos como evaluación.

Como mejora, sería deseable contar con un mecanismo que le permita a les docentes de nir estos nuevos ejercicios sin necesidad de que tengan que hacer modi caciones al código de Pilas Bloques y de forma tal que le estudiante no haya visto el desafío de forma previa. El objetivo de esta tesis es sentar las bases para lograr esto, abordando una primera etapa del desarrollo de esta mejora. Para lograr esto, es necesario entonces hacer un análisis previo que permita de nir alcances y poner en contexto las complejidades que pueden surgir durante este proceso.

Dado que los desafíos en Pilas Bloques son muy variados y cuentan con muchas características que los pueden diferenciar mucho entre sí, empezaremos primero por hacer un análisis de las mismas.

4. ESTRUCTURA DE UN DESAFÍO EN PILAS BLOQUES

4.1. Mecánica de un desafío en Pilas Bloques

En esta sección lo que haremos es hacer un análisis de un desafío de Pilas Bloques con un poco más de profundidad. Se introducirán conceptos que utilizaremos en el resto del trabajo e iremos aclarando algunos aspectos relevantes que se deberán tener en cuenta a la hora de de nir un ejercicio nuevo.

Para eso retomemos un caso antes visto como el que se muestra en la siguiente imagen.

Fig. 4.1: El autómata se llama Duba. El mismo empieza siempre a la izquierda de la la y el objetivo es desarrollar un programa que le permita comerse todos los churrascos que aparecen.

4.2. Armado y ejecución de un programa

En el centro hay un toolbox que provee los bloques que pueden utilizar. Dependiendo del desafío, el conjunto de bloques irá variando, intentando guiar a le estudiante en el tipo de solución que puede buscar.

Esta solución se puede ir armando dentro del bloque al empezar a ejecutar y cada vez que clickeen sobre el botón verde Ejecutar (ver arriba al centro) podrán ver al autómata seguir las instrucciones que se le indicaron. De esta forma, pueden ver en acciones concretas lo que desarrollaron y evaluar el nivel de precisión de la solución que propusieron.

4.3. Objetos interactuables

Los churrascos en este caso son objetos interactuables. Llamaremos objetos interactuables a los elementos de un desafío con el que puede interactuar el autómata mediante una acción que se ve expresada en un bloque. En este caso, contamos con la instrucción Comer churrasco que le permite ejecutar una acción cuando está parado sobre el objeto.

El concepto de objeto interactuable aparece en varios desafíos de la aplicación. Veamos otro ejemplo.

Fig. 4.2: El desafío le exige a le estudiante que desarrolle un programa para que el mono cuentes las bananas y las manzanas. Las manzanas y las bananas son objetos con los que puede interactuar el autómata mediante sus respectivos bloques contar manzana y contar banana .

4.4. Mensajes de error

Sin embargo, no siempre desarrollarán una solución correcta al primer intento. Veamos entonces situaciones que se pueden presentar.

Fig. 4.3: Ejemplo de un error indicado por el autómata cuando intentó comer un churrasco donde no había uno.

En este caso, el error le indica a le estudiante que intentó interactuar con un objeto especí co de forma incorrecta. La herramienta evalúa que se intenta comer un churrasco en una casilla donde no había uno y le avisa de forma explícita mediante un globo de diálogo como se muestra en la foto.

Por otro lado, otro posible error es que el personaje se vaya fuera del mapa . Esto signi ca que estando en la última casilla, intente avanzar nuevamente a la derecha.

Fig. 4.4: Aquí se indicó repetir 8 veces el movimiento a la derecha y se obtuvo el siguiente error. El bloque violeta del medio permite, mediante una entrada por teclado, con gurar cuántas veces se repetirá una secuencia de instrucciones.

En ambos casos, la forma que tiene la herramienta para indicar a le estudiante la presencia de un error es mediante un globo diálogo del autómata. Este mensaje podrá variar de acuerdo al desafío o el tipo de error.

En otras situaciones de programas incorrectos no habrá ningún mensaje y le estudiante deberá detectar la falla por su propia cuenta sin que se le avise de forma explícita.

Fig. 4.5: Aquí se le indicó que repita 6 veces la secuencia de acción y el programa terminó sin llegar a comer el último churrasco. Esto se considera un error, pero el desafío no lo informa de manera explícita.

4.5. Obstáculos

En algunos desafíos tendremos elementos por los que no puede pasar el autómata. Los llamaremos obstáculos.

Fig. 4.6: Aquí el autómata debe comer el churrasco, pero para eso debe llegar sin chocar previamente con las piedras o los arbustos. Es decir que si en esta posición se mueve para arriba, o da un paso a la izquierda y luego para arriba, en ambos casos chocará con los obstáculos.

Este choque se puede considerar de alguna forma como un caso similar a lo que sucede cuando intenta moverse fuera de la grilla y se le indica mediante un diálogo que hubo un error.

4.6. Mensaje de nalización/felicitación

Imaginemos que le estudiante resolvió de forma exitosa un desafío. En algunos casos, la herramienta no cuenta con un mensaje de felicitaciones para le estudiante y este debe por su cuenta evaluar si el programa es correcto.

Pero en otros se muestra un cartel como el siguiente

Fig. 4.7: En algunos casos la herramienta puede evaluar que se resolvió el desafío e indicarlo de forma explícita como el caso de esta imagen.

5. CLASIFICACIÓN DE LOS DESAFÍOS

Habiendo presentado estos aspectos de Pilas Bloques, estamos en condiciones de avanzar en una clasi cación de desafíos que nos permitan enmarcar mejor el alcance de la tesis.

Veremos que, más allá de que existen muchos desafíos distintos en Pilas Bloques, a los mismos podemos agruparlos según ciertas propiedades que de niremos y que serán relevantes a la hora de pensar la mejora en la herramienta.

Los clasi caremos de acuerdo a los siguientes aspectos:

1. La estructuración del mapa por el que se puede mover el autómata.
2. La forma en la que se disponen los obstáculos con los que se puede chocar el autómata.
3. La disposición de los objetos con los que debe interactuar el autómata.
4. La existencia o no de precondiciones para las acciones del autómata. Este punto lo desarrollaremos mejor más adelante.
5. La forma en la que se decide si un desafío está correctamente resuelto.

Como agregado, veremos además que cada tipo de ejercicio refuerza el trabajo con determinados conceptos de la programación (repeticiones, condicionales, etc). Teniendo esto en claro, les docentes tendrán una mejor idea de qué tipo de ejercicios son los que necesitan como refuerzo y evaluación de los contenidos sobre los que quieran enfocarse.

5.1. Estructura del escenario

5.1.1. Escenarios de grillas

Los ejercicios de grillas son los que tendrán como escenario una matriz o grilla de N las y M columnas.

Fig. 5.1: En este caso el autómata (marciano verde) deberá comerse todas las manzanas (objetos).

En este tipo de escenarios los movimientos podrán ser entre casillas. El autómata podrá moverse desde la casilla en la que esté el autómata hacia alguna que esté a la izquierda, derecha, arriba o abajo.

Los límites de los movimientos que se pueden hacer estarán dados por.

- La existencia de obstáculos en el escenario.
- La imposibilidad de irse por fuera de la grilla.
- Los bloques que se le proveen a le estudiante. Por ejemplo, en este desafío del marciano se le podría dar únicamente la opción de moverse a la derecha, izquierda o arriba.

Estos ejercicios nos permiten introducir el uso de repeticiones y procedimientos para evitar repetir código y trabajar la secuencialidad entre instrucciones de un programa.

5.2. Escenarios desestructurados

En este tipo de escenarios los movimientos posibles del autómata no estarán prede nidos por grillas como comentamos antes.

Fig. 5.2: Aquí se debe lograr que el marciano pinte de negro la gura sombreada en gris.

En el desafío de la Fig. 5.2, el autómata podrá avanzar una cantidad de unidades dibujando una línea negra en ese recorrido.

El mover derecha , mover izquierda , mover abajo o mover arriba será reemplazado por pintar una cantidad de unidades en la dirección en la que esté mirando el personaje y girar una cantidad de grados .

En este tipo de desafíos el avance no será estructurado por casillas como en las grillas sino que se le deberá indicar la cantidad de píxeles que avanzará el marciano.

En estos ejercicios se puede trabajar fuertemente el uso de procedimientos con y sin parámetros. Por ejemplo, se podría armar un procedimiento dibujar triángulo donde tendrá un parámetro con el que indicamos el tamaño de los lados del triángulo. De esta forma ya tenemos resuelto una parte del problema sin necesidad de repetir el código para dibujar cada una de esas guras de distintos tamaños.

5.3. Disposición de obstáculos

5.3.1. Escenarios con obstáculos jos

En este tipo de desafíos tendremos obstáculos por los que el autómata no podrá transitar. Lo relevante es que cada vez que reiniciemos el desafío, la disposición de los mismos será la misma.

Fig. 5.3: En el desafío de la gura tenemos una grilla de 8x8. Nuestro autómata empieza en la posición (3,1) y debe llegar hasta el churrasco para comerlo.

En el desafío de la gura Fig. 5.3, las piedras y arbustos son obstáculos y el autómata no puede moverse a través de ellos. La disposición de los mismos será siempre la misma.

Este tipo de ejercicios tienen objetivos similares a los de las grillas antes presentados.

5.4. Escenarios con obstáculos variables

En este tipo de desafíos, la disposición de los obstáculos podrá variar. Veamos un ejemplo.

Fig. 5.4: Aquí Duba debe llegar a comer el churrasco que siempre estará en el mismo lugar. Sin embargo, los obstáculos cambiarán de posición, teniendo que le estudiante desarrollar un programa que generalice estas dos situaciones.

En otros casos, tendremos casillas que estarán dentro de la grilla pero que serán inaccesibles. A pesar de que no hay un obstáculo representado de forma explícita por una imagen o ícono, si podremos pensar a estas casillas como tales. Veamos algunos ejemplos.

Fig. 5.5: En este desafío el robot debe recorrer cada la hasta el nal de la misma y patear la pelota. Cuando se apriete la opción de ejecutar , la cantidad de las será siempre la misma, pero cada una cambiará su longitud.

Las casillas que están después de la pelota no son accesibles y pueden ser pensadas entonces como obstáculos.

Fig. 5.6: Con esta imagen queremos ejempli car cómo podríamos ver a este ejercicio como de obstáculos variables.

Si se compara con la Fig. 5.5, se verá que las casillas por las que puede transitar el autómata no son las mismas. Las casillas que aparentan estar vacías en realidad serán consideradas obstáculos pero no mostrarán ningún ícono, dando la idea de que la grilla va variando su tamaño en cada la.

Estos tipos de ejercicios tienden a agregar una variabilidad en los escenarios, lo que induce la necesidad de utilizar repeticiones y alternativas condicionales de modo de poder generalizar el comportamiento de los autómatas.

Otro ejemplo para representar esta idea es el siguiente

Fig. 5.7: En este escenario el ratón debe llegar hasta la casilla negra. El camino está compuesto por muchas casillas verdes que le indican al ratón cuál debe ser la dirección de su próximo movimiento, pudiendo ser para la derecha o para abajo. Cada vez que apretemos ejecutar , si bien la cantidad de casillas será la misma, se modi cará la dirección que indicará cada una.

Como en el ejemplo del robot que patea pelotas, podemos pensar a estas casillas inaccesibles como obstáculos que no están representados por imágenes o íconos. Cada vez que apretemos ejecutar , la distribución de casillas con obstáculos cambiará, de modo que también se verá modi cado el camino que puede transitar el ratón.

Aquí mostramos cómo se puede pensar el escenario como una grilla y las casillas en las que no hay camino pueden considerarse como obstáculos.

5.5. Disposición de los objetos interactuables

5.5.1. Objetos interactuables jos

Fig. 5.8: En este caso el fueguito será nuestro autómata y deberá avanzar por cada la y recolectar cada estrella. El objetivo es que el programa termine sin errores y que no hayan quedado estrellas por recolectar.

En este tipo de escenarios, los objetos con los que interactúa el autómata estarán siempre en el mismo lugar y su disposición no cambiará. Cada vez que apretemos ejecutar , el autómata volverá a la posición inicial (la mostrada en la imagen) y las estrellas volverán a disponerse como en la foto. Lo relevante es que esta disposición será la misma en cada intento.

5.5.2. Objetos interactuables variables

En este tipo de problemas, los objetos con los que interactúa el autómata cambiarán sistemáticamente de disposición.

Fig. 5.9: En este caso el autómata deberá comerse todos los churrascos y se debe hacer un programa que resuelva esto sin importar la disposición de los mismos. En cada reinicio del problema la grilla se mantendrá igual pero cambiará la disposición de los objetos.

Fig. 5.10: En este ejemplo, lo que debemos lograr es que el mono cuente la cantidad de manzanas y bananas que hay. La disposición de estos objetos irá variando en cada reinicio.

El desafío de la Fig. 5.10 combina las características de los escenarios de objetos variables con obstáculos variables ya que en cada intento cambiará la disposición de las casillas por las que puede transitar el autómata como también los objetos con los que puede interactuar en la grilla.

Este tipo de ejercicios exige el uso de repeticiones simples (repetir una cantidad ja de veces) o condicionales, combinadas con condicionales anidados para tomar decisiones sobre si interactuar o no con objetos en cada paso del autómata.

5.6. Ejercicios con y sin precondiciones entre acciones

Fig. 5.11: En este problema, la princesa (la que tiene un escudo rosa) debe rescatar al príncipe que es capturado por el caballero (el que tiene escudo violeta).

En este desafío, cada acción que realizamos sobre algún objeto se debe hacer posterior a otra acción en particular. Por ejemplo, para abrir el cofre debemos previamente haber encontrado la llave. Dentro del cofre hay una espada que necesita para poder enfrentar al caballero.

Podemos decir así que ciertas acciones cuentan con la precondición de que alguna en particular se haya realizado antes.

Fig. 5.12: Aquí el marciano debe reparar la nave para escapar en ella. Para hacerlo debe juntar las piezas de hierro y carbón.

En total hay tres unidades de hierro y de carbón en el escenario. Como condición adicional, el marciano puede llevar una unidad de cada material por vez, por ende debe ir y volver a buscar los materiales seis veces en total. Una vez resuelto eso, podrá irse en la nave.

Así es como tenemos desafíos donde existen precondiciones para las acciones mientras que otros no.

Por ejemplo, en los desafíos de las Fig. 5.9 y Fig. 5.10, las acciones sobre los objetos son independientes de lo que hicieron previamente. Es decir, se puedo contar una banana, contar una manzana o comer un churrasco sin importar lo que sucedió previamente. Sólo se deberá contemplar realizar la acción cuando se esté efectivamente sobre alguno de los objetos con los que se puede interactuar.

5.7. Cómo se cumple el objetivo (condición de nalización)

Como ya dijimos, en algunos casos se indicará explícitamente a le estudiante que realizó con éxito el desafío. Pero en otros la herramienta no cuenta con un mensaje de felicitaciones para le estudiante y este debe por su cuenta evaluar si el programa funciona de forma correcta. En general, podemos detectar dos escenarios fácilmente distinguibles en los que se puede mostrar un mensaje de felicitación como el que decimos.

- Uno en el que todos los objetos interactuables fueron interactuados por el autómata (por ejemplo que no quedan churrascos por comer).
- Uno en el que el autómata pasa por una sucesión de estados hasta llegar al nal. Para haber llegado a la misma se tuvieron que haber cumplido un conjunto de condiciones previas (esto es pensando en ejercicios con acciones que tengan precondiciones).

Tenemos otros ejemplos pero que no representan a la mayoría de desafíos de Pilas Bloques. Por ejemplo el del ratón en el laberinto, desafío en el que el ratón debe llegar hasta la última casilla sin salirse del camino, pero donde no tendrá objetos para interactuar.

5.7.1. Caso de que todos los objetos fueron interactuados

Estos por ejemplo son los desafíos como el de Duba en el que debe comerse todos los churrascos, o el fútbol para robots en el que el autómata debe patear todas las pelotas. Los churrascos y las pelotas son los objetos interactuables, y Pilas Bloques debe analizar que no quedó ninguno por interactuar.

5.7.2. Caso de sucesión de estados

Ya vimos dos ejemplos, el de La aventura del mar encantado y el de Reparando la nave . Ambos desafíos se completan exitosamente cuando se logra realizar una acción nal. En el primer caso es cuando logró escapar con el unicornio luego de haber rescatado al príncipe y en el otro cuando logró irse con la nave. Cada acción sobre determinados objetos necesitará que previamente se haya realizado otra en particular.

Podemos pensar así a estos desafíos como una sucesión de estados por los que va pasando el autómata. En el caso de Reparando la nave , luego de traer todos los materiales, pasa a un estado de nave reparada que le permite realizar la acción de escapar. En el caso de La aventura del mar encantado , la princesa debe primero obtener la espada, para con eso poder rescatar al príncipe y luego irse ambos en el unicornio.

En todos estos casos contamos con varios estados, debiendo transitar por los mismos en un determinado orden para poder realizar la última acción y así saber que se resolvió exitosamente el desafío.

Fig. 5.13: Este es otro ejemplo de sucesión de estados. El objetivo del desafío es Hacer que el gato avance un paso, se duerma, se despierte, salude y vuelva a su lugar .

Fig. 5.14: Estos son los bloques con los que se dispone.

Para resolver el desafío, el gato debe realizar las acciones pero respetando el orden pedido por el enunciado.

El desafío se puede considerar resuelto cuando ejecute la acción de volver a su lugar habiendo pasado por los estados anteriores y respetando ese orden.

6. HACIA LA IMPLEMENTACIÓN DE LA MEJORA

Una vez hecha esta clasi cación de ejercicios de Pilas Bloques, nos dispusimos empezar la investigación necesaria para de nir la hoja de ruta que tenga como objetivo nal este generador de ejercicios en Pilas Bloques.

6.1. Preguntas de investigación

Para poder pensar cómo desarrollar la mejora, las primeras preguntas que consideramos necesarias de responder fueron:

- ¾Se podrá diseñar una plantilla genérica y única para de nir cualquier desafío? Caso contrario, ¾Cuál es la mínima plantilla que maximiza la variedad de desafíos que se pueden construir?
- ¾Hoy cómo de ne un programador en Pilas Bloques un desafío nuevo?

Cuando hablamos de plantilla genérica nos referimos a la posibilidad de, con una interfaz simple y sin tecnicismos propios de la tecnología con la que se desarrolló Pilas Bloques, se pueda generar cualquier tipo de ejercicio que hoy aparece en la herramienta. Si pudiéramos visualizar con claridad la posibilidad de generar cualquier desafío de los hoy existentes con esa plantilla, entonces modi cando ciertos parámetros de la misma podríamos ya crear los nuevos que quisiéramos.

Ahora bien, para tener una primera aproximación de cómo responder a esa pregunta, evaluamos necesario responder la segunda. Adentrándonos en el código de la aplicación, deberíamos ver si los ejercicios generados actualmente responden a una implementación adaptable a las necesidades planteadas en la tesis.

Quizás resulte necesario, previo a ver ese nivel de adaptabilidad, responder la siguiente pregunta:

- ¾Es técnicamente sencillo agregar un desafío previamente inexistente, dada la arquitectura actual de la aplicación?
- En otras palabras, ¾cuál es el límite técnico de la exibilidad de la arquitectura actual? .
- Y posteriormente ¾está al alcance de docentes de escuelas sin conocimientos de la implementación de Pilas Bloques agregar un desafío de esa forma?

En este punto es necesario meternos de lleno en cuestiones de índole técnica sobre el lenguaje de programación con el que está creado Pilas Bloques. El mismo está desarrollado con Typescript, un lenguaje que admite el paradigma de programación orientada a objetos.

Un desafío en Pilas Bloques es una clase que se instancia cada vez que se carga un ejercicio. El autómata y sus acciones serán objetos con los que colaborará el desafío.

El diseño actual no admite en principio la posibilidad de parametrizar todos los comportamientos o características que nos interesan en los desafíos. Es por esto que vemos a la

actual implementación poco exible para resolver la problemática de la creación de nuevos ejercicios sin que haya que modi car el código de la aplicación.

En cuanto a los Actores, sus acciones están en principio disociadas de sus animaciones y veremos más adelante cómo esto aporta cierta exibilidad para trabajar esta parte, presentando una propuesta de diseño que resuelva este aspecto a la hora de de nir un desafío nuevo.

Pensamos entonces que, si bien los actores y sus acciones serían relativamente fáciles de poder generar de forma accesible para docentes, no lo serían los escenarios en sí junto a sus objetos, obstáculos y objetivos.

Nos planteamos entonces las siguientes preguntas.

- ¾Qué sucede si pensamos en la posibilidad de recortar el espectro de ejercicios que podamos generar con esta mejora?
- ¾Con ese recorte podríamos tener un conjunto de ejercicios que, en términos del diseño, sea fácilmente extensible?
- ¾Tendríamos así una forma fácil de poder personalizarlos sin necesidad de que el docente deba ponerse a programar?

Lo que nos propusimos entonces fue pensar en la posibilidad de que, haciendo ese recorte, podamos encontrar un conjunto de propiedades generales a todo ese subconjunto de ejercicios que permita visibilizar esa idea de plantilla uni cada que comentamos antes.

El siguiente paso sería analizar los desarrollos necesarios para darle a Pilas Bloques la capacidad de admitir nuevos ejercicios generados con esta herramienta en potencia.

6.2. Recorte de ejercicios

Decidimos entonces encarar la tarea de trabajar con ejercicios de grillas con obstáculos jos y objetos interactuables variables. Las acciones no tendrán precedencia entre ellas y la condición de nalización será que no queden objetos interactuables en el escenario.

La razón por la que elegimos este tipo de ejercicios es porque aportan la su ciente exibilidad para trabajar los conceptos asociados a Pilas Bloques de la propuesta de Program.AR.

6.3. Plantillas ya existentes en otras herramientas

Comenzamos por revisar la herramienta Gobstones [15] pues el estilo de desafíos es similar al de Pilas Bloques. Tomemos como ejemplo el desafío La música del loro .

Fig. 6.1: El desafío consiste en que el loro debe moverse desde la casilla inicial hasta algún instrumento que haya, y tocarlo. Tanto el instrumento como la posición del mismo pueden cambiar, de modo que se debe escribir un programa que generalice a cualquier instrumento y que sea independiente de la posición en la que aparezca el mismo.

Fig. 6.2: Podemos ver que la interfaz de usuario del desafío tiene una estructura similar a Pilas Bloques, con un toolbox para elegir los bloques, un bloque principal donde armamos la solución al desafío, y por último el escenario.

Mediante un directorio local o de un repositorio en github (según si usa modalidad o ine u online), Gobstones obtiene la información necesaria para de nir un conjunto de ejercicios. Por ejemplo, en ese directorio están de nidos los escenarios iniciales y la disposición de los obstáculos y objetos interactuables que habrán en el mismo. También estarán incorporadas las imágenes, entre otras cosas.

Fig. 6.3: Aquí podemos ver algunos directorios y contenido que hay del desafío en Github. Por ejemplo, los archivos .gbb que se ven a la derecha son las distintas con guraciones iniciales de la grilla. El equipo de gobstones decidió que la forma de aportar aleatoriedad al escenario es mediante la rotación de estos archivos.

Fig. 6.4: De la misma manera, hay un directorio donde se encuentran todas las imágenes que representan grá camente al desafío. El autómata cuenta con una imagen propia para ser representado en la aplicación.

Esta estrategia de trabajar con un directorio tiene la ventaja de también contemplar el trabajo o ine ya que los archivos y de niciones podrían tomarse de un directorio local en lugar de un repositorio online.

Sumado a eso, podríamos generar un ambiente colaborativo entre docentes en el que subiendo los ejercicios a un repositorio común podrán reutilizar ejercicios elaborados por otres colegas.

Para saber entonces qué debería tener ese directorio en la mejora de Pilas Bloques, pasemos a analizar en detalle cómo se podría de nir un ejercicio representativo del conjunto elegido. Debemos reconocer las partes que conforman un desafío y así poder detallar mejor el conjunto de aspectos que deberá de nir el docente.

7. RETOMANDO EL ANÁLISIS PARA UN DESAFÍO NUEVO

En esta sección retomamos y analizamos otros aspectos de un desafío de Pilas Bloques que tendremos que contemplar para de nir uno como tal. Esto permite nalmente captar los aspectos que deberá tener en cuenta un docente para de nir un nuevo desafío y poder a nar así los requerimientos. Retomemos con un desafío de Pilas Bloques que entra en la categoría que elegimos previamente.

Fig. 7.1: Nuestro autómata (Duba) es la puma que quiere llegar hasta el churrasco y comerlo. Pilas Bloques nos permite ejecutar el ejercicio varias veces para probar (de manera empírica) que la solución es correcta, pero con la característica de que el churrasco va cambiando de posición en cada intento.

En este desafío, la premisa es que el churrasco siempre se mantendrá en la misma la de Duba y en alguna casilla que esté a la derecha del inicio del personaje.

7.1. Instrucciones disponibles

Una solución posible es

Fig. 7.2: El personaje se irá moviendo paso a paso hasta llegar al churrasco. Una vez que está sobre el mismo, dejará de moverse, lo comerá y el desafío estará terminado.

En este caso le estudiante cuenta con las siguientes instrucciones o estructuras de control

1. Repetir hasta que se cumpla una condición.
2. Sensores para saber si estoy sobre un churrasco (que permiten controlar la condición de la repetición).
3. Mover a la derecha.
4. Comer el churrasco.

Una solución alternativa podría ser

Fig. 7.3: En esta solución, el autómata irá caminando hasta llegar a una pared y se detendrá. Pero como en cada paso irá viendo si hay un churrasco y se lo comerá de ser así, tenemos la garantía de que al terminar se lo habrá comido.

Lo que cambia entra una solución y otra es el conjunto de bloques con los que cuenta.

1. Repetir hasta (condición).
2. Sensores para saber si:
3. Tengo una pared (u obstáculo) a la derecha.
   a) Estoy sobre un churrasco.
   b) Mover a la derecha.
   c) Comer el churrasco.

Como podemos ver, no es algo menor el conjunto de instrucciones que se decida proveer en el desafío. Si por ejemplo retiramos la posibilidad de ver si hay obstáculos a la derecha, estaremos recortando el espacio de soluciones. Como también podríamos optar por darle todas las antes mencionadas y el ejercicio admitirá muchas y más variadas soluciones.

En ese sentido, la elección de instrucciones disponibles para le estudiante es un aspecto central en el armado de un ejercicio.

Y retomando lo planteado en la discusión de open ended vs orientados a desafíos, puede ser deseable que el set de instrucciones pueda ser acotable por le docente.

7.2. Mensajes del autómata y felicitación

Como dijimos anteriormente, cada desafío deberá tener mensajes propios para cada momento en el que el autómata choca contra un obstáculo o interactúa de forma incorrecta con algún objeto.

Fig. 7.4: Acá el autómata indica que se intentó comer un churrasco donde no había una. Si en este desafío en lugar de churrascos hubiesen manzanas, sería deseable que el mensaje del autómata cambie.

Para lograr esto, se podría permitir que le docente pueda con gurar los mensajes que le aparecen le estudiante ante cada situación como estas.

De la misma manera, se podría pensar que el mensaje de felicitación sea con gurable de modo que, entre otras cosas, se pueda dejar pistas o recordatorios de cómo mejorar el programa que armó le estudiante.

Por ejemplo: ½Felicitaciones! ½Lograste el objetivo! Si tu programa no usó repetición condicional, podrías pensar la forma de utilizarlo para mejorar la solución! .

7.3. Información para le estudiante

Cada desafío tendrá un título, enunciado y posiblemente un texto con pistas para le estudiante.

7.4. Imágenes y animaciones

Además de los bloques disponibles con los que contará le estudiante, debemos de nir las imágenes que le aportarán la estética al desafío.

Desde esta óptica, debería estar de nido

1. Una imagen que de na el fondo del escenario.
2. Imágenes que le den forma a los obstáculos.
3. Imágenes que le den forma a cada objeto con los que interactúa el autómata.
4. Una imagen o sprites que den forma y animación al autómata.
5. Imágenes que le den forma a las casillas de la grilla por las que se mueve el autómata

7.5. Tamaño y forma de la grilla

En todos los desafíos habrá un autómata, de modo que en relación al escenario por el cual este se mueve, debería ser necesario especi car lo siguiente

1. Cantidad de las y columnas de la grilla.
2. Posición inicial del autómata.
3. Posición de los obstáculos.
4. Posición de los objetos con los que el autómata deberá interactuar.

8. SÍNTESIS DE LOS ASPECTOS ANTERIORES

De este modo, sin ser la única opción posible, llegamos a resumir que debería alcanzar con que le docente de na lo listado a continuación.

1. Cantidad de las y columnas de la grilla.
2. Posición inicial del autómata.
3. Posición de los obstáculos.
4. Posición de los objetos interactuables.
5. Imágenes del fondo, obstáculos y objetos interactuables.
6. Sprites para animar las acciones del autómata.
7. Título del ejercicio.
8. Enunciado para le estudiante.
9. Bloques con los que contará le estudiante junto a los mensajes ante algún error de las mismas y, en caso de que sea necesario, indicar el objeto con el que interactúan.
10. Mensaje de ayuda o sugerencia para la resolución.
11. Mensaje de felicitación cuando resuelva el desafío.

Recordemos que para el conjunto de ejercicios elegidos no será necesario indicar alguna Condición de Finalización ya que será siempre que no deben quedar objetos interactuables.

9. SIGUIENTE ETAPA: CONTEXTO DE TRABAJO CON PILAS BLOQUES

9.1. De nición básica de un desafío

La herramienta cuenta con un mecanismo medianamente estandarizado que parece resolver de forma parcial el problema. Este método consiste en un .json donde las claves son palabras reservadas que hacen referencia a algunos aspectos parametrizables de cada ejercicio junto a los valores que toman en el momento de mostrar y ejecutar el ejercicio.

Veamos un ejemplo:

```
{
    id: 201,
    nombre: '3.1.2a',
    titulo:   'Desafio 1',
    escena: `new EscenaDuba("\
            [O,O,O,O,O,O],\
            [O,-,-,-,O,-],\
            [-,A,-,-,P,-],\
            [-,-,-,O,-,-],\
            [O,O,O,O,-,O],\")   `,
    enunciado: 'Ayuda a la puma Duba a comer su churrasco.
                Evita los obstaculos.  ',
    debeFelicitarse: true,
    estiloToolbox:   'sinCategorias',
    bloques: [
        'MoverACasillaAbajo',
        'MoverACasillaArriba  ',
        'MoverACasillaIzquierda  ',
        'MoverACasillaDerecha',
        'ComerChurrasco'
    ]
}
```

Cuando la aplicación recibe el pedido de cargar un ejercicio en particular (indicado por un id), busca en este archivo por este id y obtiene cierta información de base para saber qué desafío mostrarle a le estudiante. Podemos ver cómo algunos aspectos analizados anteriormente son mapeados en este .json

1. En el campo enunciado se especi ca el enunciado inicial que indica a le estudiante el objetivo a alcanzar en el desafío.

2. En el campo bloques están de nidos los bloques (instrucciones) que tendrá a disposición le estudiante para resolver el ejercicio. Estos bloques son, por el momento, sólo los que ya existen en la aplicación (desarrollaremos mejor este aspecto más adelante).

El campo escena merece una explicación en mayor detalle. Podemos ver que tiene el string

```
"new EscenaDuba(\
    [O,O,O,O,O,O],\
    [O,A,*,*,-,*],\
    [O,O,O,O,O,O],\
  `,)"
```

El substring new EscenaDuba.. indica el desafío que se tiene que instanciar, en este caso el de Duba. Pero por poner ejemplos, en el mismo campo también podríamos poner new EscenaCoty.. , new EscenaCotySonrisa() , new EscenaLita.. , obteniendo así distintos desafíos.

En el caso de la escena de Duba, seguida del substring, tenemos una grilla de tres las y seis columnas. La cantidad de las está de nida por la cantidad de arrays, y cada la será un arreglo de caracteres.

Estos caracteres de nirán cómo será esa la. Veamos cada uno.

1. `O' indica dónde habrá obstáculos.
2. `A' indica dónde empezará el autómata.
3. `*' es un caracter especial que indica que en esa posición podrán haber objetos interactuables o no, y de esa forma se pueden armar los escenarios variables en donde los objetos aparecen en distintos lugares.

De esta manera, podemos tener un mismo tipo de desafío con distintos tipos de escenarios, permitiendo tener cierta exibilidad para generar nuevos problemas para les estudiantes.

Ahora bien, al n y al cabo, la que se encargará de hacer ese trabajo nal será una clase en Typescript, que hoy por hoy no es con gurable en muchos aspectos. Entonces si bien tenemos exibilidad en de nir la grilla y la disposición de sus elementos; los objetos y obstáculos que utilizará el desafío están hardcodeados en la clase que instancia el desafío. Esto signi ca que se puede generar una escena de Duba pero no que, en lugar de churrascos, tenga otro objeto para interactuar; y de la misma manera no se puede decidir los obstáculos o el fondo del mapa.

A su vez, podemos indicar los bloques que tendrá a su disposición le estudiante, pero estos serán sólo los prede nidos en la aplicación. Como ya comentamos previamente, nosotros queremos apuntar a que le docente pueda generar nuevos de su preferencia, con nombre y mensaje de error con gurables, y que pueda indicar el objeto con el que interactúa.

En conclusión, podemos decir que hoy por hoy no nos alcanza solamente con modi car valores de este archivo .json para poder de nir un desafío completamente nuevo. Hay aspectos que queremos con gurar que están programados en Typescript y que no son customizables como a nosotros nos gustaría.

Como para dar una foto general del proceso actual que implica generar un nuevo desafío, tenemos que.

Fig. 9.1: Cómo se genera un ejercicio hoy en Pilas Bloques.

Se resume la idea de que para que Pilas Bloques tenga un nuevo desafío hay que proveer un input de nido en el lenguaje de con guración (representado en el json antes descrito) y algunos aspectos adicionales deberán estar programados de forma ad-hoc.

Un detalle más para aclarar es que este archivo que contiene estos diccionarios con la información de uno y cada uno de los desafíos está encapsulado en la aplicación y la misma lo consume de forma interna. De modo que también deberemos hacer modi caciones para que la aplicación tenga la exibilidad de consumir uno externo que provea le docente.

De ser así, para el conjunto de desafíos que elegimos, queremos que la foto general de trabajo para docentes pueda ser algo como.

Fig. 9.2: Cómo se propone en esta tesis generar ejercicios.

Pensamos entonces que tenemos las siguientes tareas para abordar.

1. Extender este lenguaje de con guración para que cada ejercicio pueda ser autocontenido en algún tipo de formato compatible con Pilas Bloques.
2. Adaptar el código de Pilas Bloques para que pueda contemplar esta mejora.

9.2. ¾Qué aspectos de este lenguaje de con guración se deberían extender?

Tomando los elementos que conforman a un desafío, y viendo lo que está actualmente incluido en el lenguaje, se llegó a la conclusión de que, en un primer abordaje, debería ser su ciente con poder agregar la especi cación de los siguientes elementos.

1. Objetos interactuables: Los objetos con los que el autómata puede interactuar mediante una acción de nida. No nos referimos a la posición, sino a los nombres de distintos obstáculos.

2. Obstáculos: Similar a los Objetos Interactuables pero con estos no puede interactuar el autómata. Se debería poder con gurar el mensaje de error que emite el autómata cuando choque con los mismos. El mensaje de error de cuando intentamos irnos de la grilla del escenario podría ser el mismo.

3. Las acciones con las que contará le estudiante. Recordemos que hablamos de poder de nir inclusive las que no están de forma preexistentes en Pilas Bloques. Esto incluye por ejemplo el mensaje de error ante la interacción con el objeto incorrecto, con qué objeto interactúa, etc.

4. Mensaje de felicitación para le estudiante cuando resuelve el desafío.

Notar que aquí estamos omitiendo la de nición del Autómata ya que el mismo es único en el desafío y siempre habrá uno por lo que resultaría redundante tener información del mismo en este .json.

Sumado a esto, dado que los ejercicios pueden contener imágenes que actualmente no están incorporadas en Pilas Bloques, también deberá haber un mecanismo para poder proveer de las mismas para que se puedan cargar y mostrar. En un apartado posterior veremos el formato de archivo que podrá importar Pilas Bloques y ahí se especi cará dónde irán estas imágenes.

Y por último, dado que entendemos que esta mejora de Pilas Bloques tiene como norte que les docentes puedan crear nuevos ejercicios con mayor autonomía, no podemos esperar que puedan escribir el detalle de este archivo de con guración. Creemos entonces que sería deseable que haya una interfaz que abstraiga estos detalles y que sea la misma la que termine generando este input que luego consumirá Pilas Bloques, llegando entonces al siguiente esquema de trabajo.

Fig. 9.3: Cómo debería ser el esquema ideal de creación de ejercicios (hoy factible gracias al trabajo de esta tesis)

Esta GUI podría ofrecer una forma interactiva para que les docentes (o inclusive desarrolladores de la aplicación) de nan las características del desafío y que luego esta sea la que genera un .zip en donde esté el diccionario que contenga la de nición del ejercicio y lo necesario para representar las animaciones del autómata, objetos, obstáculos y fondo del mapa.

La idea de esta GUI se presenta a modo de deseo de lo que debería estar al nal del camino de la mejora, pero será un trabajo pendiente que no se abordará en esta tesis.

9.3. Extensión del lenguaje de con guración

Presentamos entonces la estructura de cómo quedaría este nuevo formato de json. Se verá que es muy similar al archivo visto antes pero con algunos parámetros agregados o modi cados. Se eligió hacer una adaptación al inglés porque el equipo de desarrollo está privilegiando el código en este idioma, contemplando futuras colaboraciones a la herramienta por parte de desarrolladores/as no hispanohablantes (recordemos que la herramienta es de código abierto).

Pondremos valores a modo de ejemplo de cómo se deberán de nir los valores. Explicaremos luego qué representa cada campo y sus respectivos valores.

```
1 {
2   title:    ' Desafio  1 ' ,
3   scene:  `"\
4     [O,O,O,O,O,O],\
5     [O,A,*,*,-,*],\
6     [O,O,O,O,O,O],\
7   "` ,

9   task:  ' Ayuda  a  la  puma  Duba  a  comer  su  churrasco.  Evita  los
        obstaculos.  ' ,
10 Suggestion:  ' Podes  resolver  el  desafio  utilizando  repeticiones
      condicionales  ' ,
11   toolboxStyle:    ' sinCategorias  ' ,

13   blocks: [
14      { name: ' MoverACasillaAbajo  '  },
15      { name: ' MoverACasillaArriba  '  },
16      { name: ' MoverACasillaIzquierda  '  },
17      { name: ' MoverACasillaDerecha ' },
18      { name: ' RecogerMedia' , description:    ' Recoger  media' , type:  ' action
           ' , interactsWith:    ' media' , errorMessage: ' Aca  no  hay  media!'  },
19      { name: ' RecogerPapel' , description:    ' Recoger  papel' , type:  ' action
           ' , interactsWith:    ' papel' , errorMessage: ' Aca  no  hay  papel!'  },
20      { name: ' hayMedia' ,  description:    ' Hay  media' , type:  ' sensor' ,
           object:  ' media'  }
21   ],

23   colisionWithObstaclesMessage:    ' Ouch,  eso  dolio! ' ,
24   congratulationsMessage:  ' Felicitaciones,  Duba  se  comio  todos  los
        churrascos! ' ,
25 }
```

Describimos cada uno de los campos a continuación

1. El campo title representa el nombre del desafío.

2. El campo scene toma como valor un string que representa la disposición de obstáculos, objetos y punto de inicio del autómata (el formato de este string está explicado en una sección anterior).

3. El campo task es el mensaje que le aparecerá a le estudiante para que sepa lo que tiene que hacer. Por ejemplo en este ejercicio tenés que lograr que el perro se coma todos los pedazos de carne .

4. toolboxStyle : Les docentes podrían preferir presentar el toolbox de bloques bien separados y clasi cados según sus roles en el problema (movimiento, acciones sobre objetos, sensores, etc) o bien puede decidir que se presenten todos en un mismo grupo. Este campo originalmente se llamaba `estiloToolbox', y con el valor `sinCategorias' indicaba que sea una lista en un solo grupo y no subdividido, y con `conCategoria' es el caso contrario. Este campo decidimos dejarlo y por defecto tendrá este último valor para simpli car lo más posible la plantilla de la interfaz con la que interactúa el docente.

5. Suggestion : será un mensaje de ayuda o guía para le estudiante.

6. colisionWithObstaclesMessage indica el mensaje que mostrará el autómata cuando choque con un obstáculo. Podemos suponer que para todos los obstáculos tendremos el mismo mensaje de colisión.

7. congratulationsMessage es el mensaje que se le dará a le estudiante cuando resuelva el desafío.

8. Blocks indicará los bloques con los que cuenta le estudiante. Veamos más en detalle las posibilidades.

   - En el caso de bloques preexistentes tales como `MoverACasillaAbajo' sólo será necesario indicar el nombre.
   - En el caso de bloques nuevos que sean acciones o sensores sobre objetos tendremos además:
     - Description será el texto que aparecerá en el bloque con el n de que le estudiante pueda entender lo que hace.
     - Type puede tener como valores `sensor' o `actions' para saber qué tipo de acción podrá hacer sobre el objeto.
     - interactsWith indica el objeto con el que debe interactuar. Por ejemplo, el sensor hay manzana? interactuará con una manzana y dará verdadero o falso según corresponda. En el caso de un bloque de acción comer manzana el autómata consumirá el objeto o arrojará un error según corresponda.
     - errorMessage es el mensaje de error que tendrá el bloque en caso de realizar la acción sobre el objeto inadecuado. Esto debería ser aplicable sólo a bloques de tipo `acción' y no a sensores.

Ya explicamos anteriormente por qué no debería ser necesario hacer referencia alguna al autómata, de ahí que no hay nada que especi que información sobre el mismo. Los sprites de su animación estarán en un archivo con un nombre jo que no será con gurable, y la forma en la que se deben interpretar los sprites para animar sus acciones será desarrollado en la siguiente sección.

Todas las imágenes e íconos referidas a objetos interactuables u obstáculos se podrán inferir de este .json. Por ejemplo, si tenemos bloques de acción que interactúan con objetos distintos, los mismos se podrán inferir del campo interactsWith de la especi cación del bloque. Los íconos de los obstáculos estarán en un directorio propio, de modo que la aplicación podrá ir cargando los distintos íconos al azar en los lugares en los que aparezca el caracter `O' en el campo scene.

9.4. Animaciones e imágenes

En el caso de los obstáculos u objetos, los mismos estarán representados por íconos que serán estáticos. Para los objetos, lo más cercano a una animación será que cuando el autómata interactúe con los mismos, estos desaparezcan de la escena para que le estudiante lo pueda ver con claridad de forma grá ca. Podemos decir a su vez que el fondo no cambiará, de modo que con aportar una única imagen para el mismo será su ciente.

Pero para el caso del autómata, decidimos que el mismo sea animado. Esto signi ca que debemos proveer un conjunto de imágenes que representen los distintos estados de la animación del mismo. Veamos un ejemplo con Duba.

Fig. 9.4: En este caso, vemos un archivo que contiene cada sprite representando un estado de la animación de Duba. Tenemos las animaciones de cuando Duba se mueve, come un churrasco, o se choca con un obstáculo, entre otros.

Pilas Bloques de ne, por ejemplo, la animación de comer churrasco como una secuencia de imágenes que podemos ver en este archivo, desde que toma el churrasco hasta que se lo come y pone una sonrisa como expresión de felicidad.

Es decir, cada objeto animado tendrá asociado un conjunto de sprites para cada una de sus animaciones.

Podemos pensar al archivo como un arreglo indexado de sprites donde el primero será el de índice 0 y el último el de la cantidad de sprites - 1.

Fig. 9.5: En este último caso tenemos el ejemplo de un autómata con la animación de muchos menos sprites que Duba.

Ahora bien, para saber qué sprites representan cada una de las animaciones del autómata, Pilas Bloques cuenta con un código como el que vemos más abajo.

```
2 class Duba extends ActorAnimado {
3  static _grilla =     'actor.duba.png'

5     constructor() {
6     super(0,0,{cantColumnas: 10, cantFilas: 15});
7     this.definirAnimacion("parado",
8      new Cuadros(0).repetirVeces(16)
9      .concat([33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52
          ])
10     .concat(new Cuadros(0).repetirVeces(30))
11     .concat([33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52
          ])
12     .concat(new Cuadros(0).repetirVeces(30))
13     .concat([33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52
          ])
14     .concat(new Cuadros(0).repetirVeces(16))
15     .concat([0,1,2,3,4,5,6,7,8,9,10,11,11,12,12,13,14,15,16,17,18,19,19,
          20,21,22,23,23,23,24,25,26,27,27,28,29,30,31,31,31,31,31,32])
16     .concat(new Cuadros(0).repetirVeces(5)),
17   12, true);
18    this.definirAnimacion("correr", [0,53,54,55,56,57,58,59,60,0], 36);
19    this.definirAnimacion("correrChocando", [0,53,54,55,56,57,58,59,60,76
          ,77,78,79,80,81,82,83,84,85,86,87,88,88,89,90,90], 12)
20    this.definirAnimacion("comerChurrasco",[0,0,61,62,63,64,65,65,65,65,6
          6,67,68,69,70,71,72,73,74,75], 12);
21    this.definirAnimacion("obstaculo", [91,92,93,94,95,96,97,98,99,100,10
          1,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117
          ,118,119,120,121,122,123,124,125,126,127,128,128,128,128,128], 12
          )
22    this.definirAnimacion("error",[129,130,131,132,133,134,135,136,137,13
          8,139,140,141,142,142,142,142], 12)
23  }
24 }
```

Duba por ejemplo es una clase de Typescript en la que hay un método llamado definirAnimacion().

El mismo recibe tres parámetros

1. El nombre de la animación.
2. Un arreglo que indica los índices de los sprites (en el archivo de la animación del autómata) puestos en el orden correcto.
3. La velocidad a la que se hará la transición entre cada imagen de la animación.

Por ejemplo, para de nir la animación de comer churrasco se llama al método de la siguiente forma

```
1 this.definirAnimacion("comerChurrasco",[0,0,61,62,63,64,65,65,65,65,66,6
    7,68,69,70,71,72,73,74,75], 12);
```

Si tuviésemos que hacer un mapeo entre el trabajo que tendría que realizar le docente y esta implementación, el mismo debería decir cuál será cada sprite de la animación. Lo cierto es que esto nos parece muy complejo y debemos simpli carlo.

Lo que proponemos es que cada animación tenga una cantidad de nida de sprites y las animaciones estén siempre representadas en el mismo orden entre ellas. De esta forma, todos los desafíos creados con esta interfaz docente deberán tener la misma estructura en los sprites del autómata.

A modo de ejemplo, a continuación ponemos los sprites de dos autómatas en el release de Pilas Bloques y luego mostramos cómo proponemos simpli carlo siguiendo la propuesta de diseño de la tesis.

Fig. 9.6: Diseño original de Pilas Bloques (sprites distintos para distintos desafíos)