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 offline, 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 different 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 offline 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. Definición básica de un desafío 47
9.2. ¿Qué aspectos de este lenguaje de configuración se deberían extender? . . . 50
9.3. Extensión del lenguaje de configuración 52
9.4. Animaciones e imágenes 54
9.5. Formato del directorio 60

10..Primeros pasos de la implementación 62
10.1. Modificar el tipo de archivo que importa 62
10.2. Modificar 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. Justificación de las actividades propuestas 76
10.8. Armado de los desafíos con esta nueva mejora 76
10.8.1. Archivo desafio.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, ofimática, etc.) sino que también permita visualizar el trasfondo de lo que sucede en las transformaciones digitales de nuestra vida cotidiana. Esta necesidad se ve reflejada 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 final 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 especificidad 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 modificar 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 específico 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 definió un conjunto de características necesarias mínimas para la implementación de un creador de desafíos. Se realizó finalmente 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 específica 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.

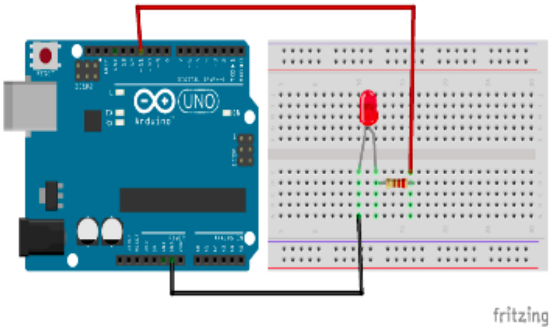


Fig. 2.1: Algunas herramientas educativas requieren conocimientos de electrónica y armado de circuitos (Diagrama hecho con Fritzing)

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.

```
1 Repetir por siempre {
2     Aplicar una tension de 5v en el pin
3     Esperar un segundo
4     Aplicar una tension de 0v en el pin
5     Esperar un segundo
6 }
```

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 final de cada instrucción, el balanceo de paréntesis y el uso de una sintaxis bien definida. 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.

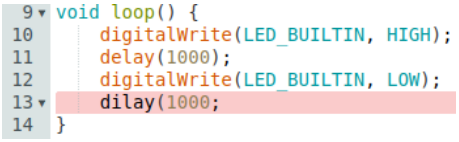


Fig. 2.2: Arduino IDE. Hay resaltado de sintaxis pero esto no evita errores.

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 específicamente, 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 affords 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-first 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, modifiers, 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 dificultad de la sintaxis y permitan enfocarse en la solución final 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 figura 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 definir 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 modificando los bloques ahí presentes. En la Fig. 2.4 hay un ejemplo de programa con bloques.

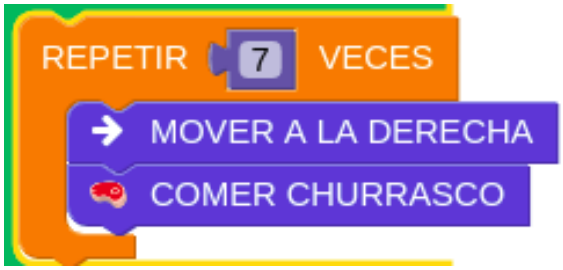


Fig. 2.4: Ejemplo de programa que se podría plantear para el desafío de los churrascos

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 gráfica e interactiva de un concepto de programación. Los bloques naranjas suelen estar asociados a lo que tradicional-

mente se considera control de flujo 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 configuraciones 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 definido 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.

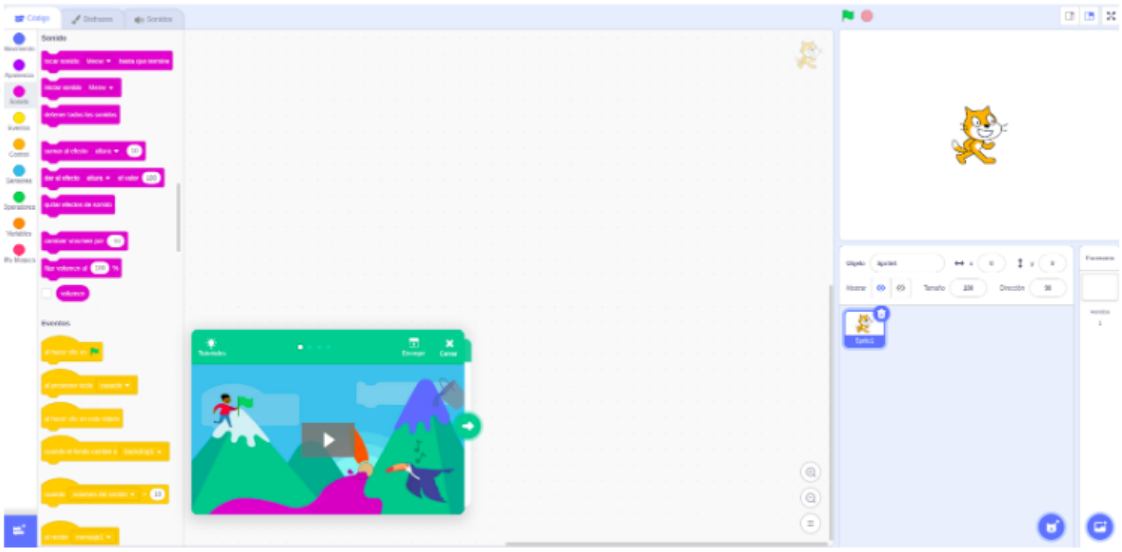


Fig. 2.5: Interfaz de Scratch. La misma presenta similitudes y diferencias con Pilas Bloques

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 modificar las imágenes de fondo, aportar animaciones y sonido a los objetos y sus acciones.

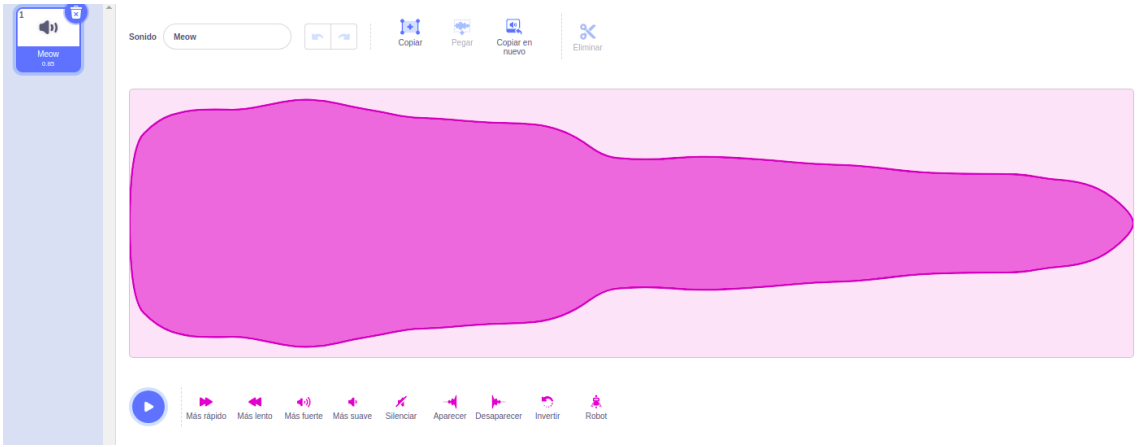


Fig. 2.6: Toolkit de edición de sonido en Scratch

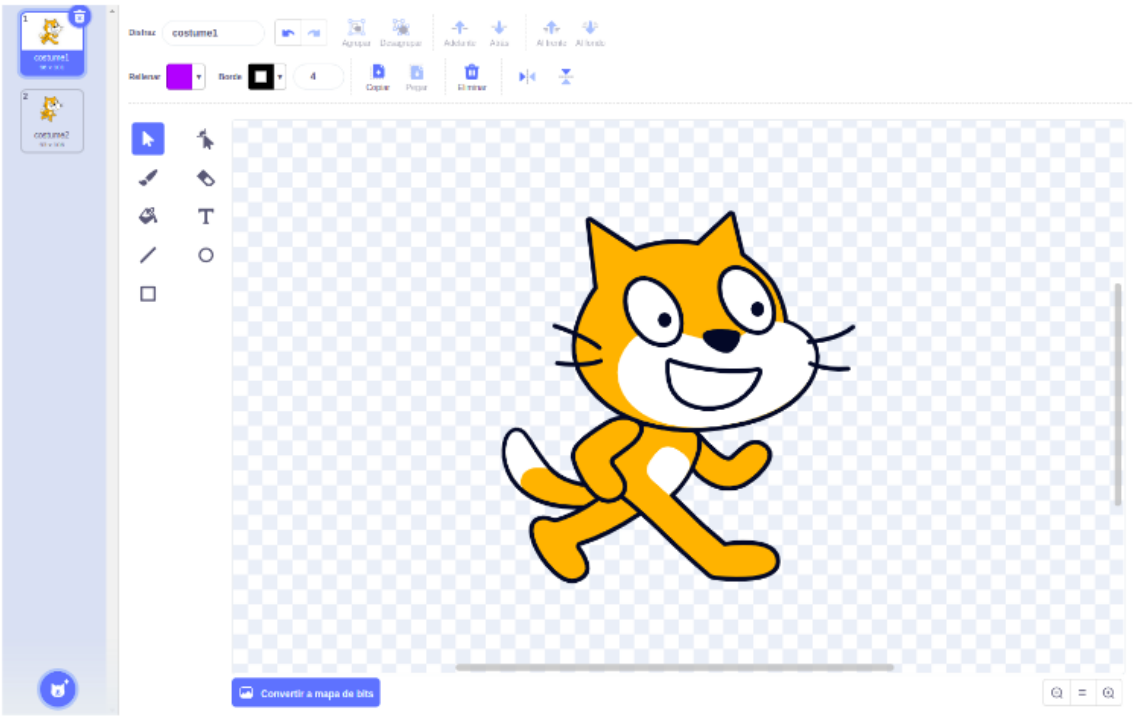


Fig. 2.7: En esta interfaz les estudiantes pueden dar nuevos disfraces o posiciones a los personajes/objetos, pudiendo así crear animaciones para los mismos.

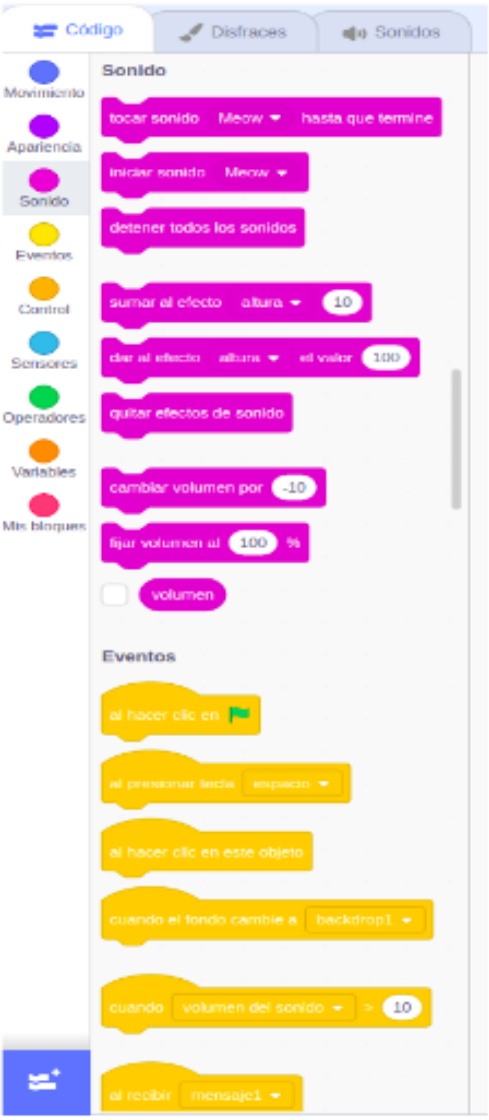


Fig. 2.8: En esta imagen mostramos el toolkit de instrucciones que permite usar Scratch.

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 differences with Pilas Bloques are equivalent to those with Scratch. Besides the 3D scenarios, the main differences 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 configurables de cada proyecto. Es decir que en todo momento les estudiantes tendrán a disposición la totalidad de los bloques y aspectos configurables 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 fixed scenarios, each one with a defined 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 predefinidos y con propiedades fijas.
2. Cada escenario tendrá un objetivo claro y bien definido.

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 finished 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 offline, 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 effort 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 offline 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 offline	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, offline, 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 fija 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 dificultad para resolver un conjunto de ejercicios y los mismos los pudo terminar con ayuda docente.¿Cómo podemos evaluar o medir el grado de avance al llegar los momentos de evaluación? ¿Tiene 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 significa 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 definir estos nuevos ejercicios sin necesidad de que tengan que hacer modificaciones 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 definir 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 definir 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 fila 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.

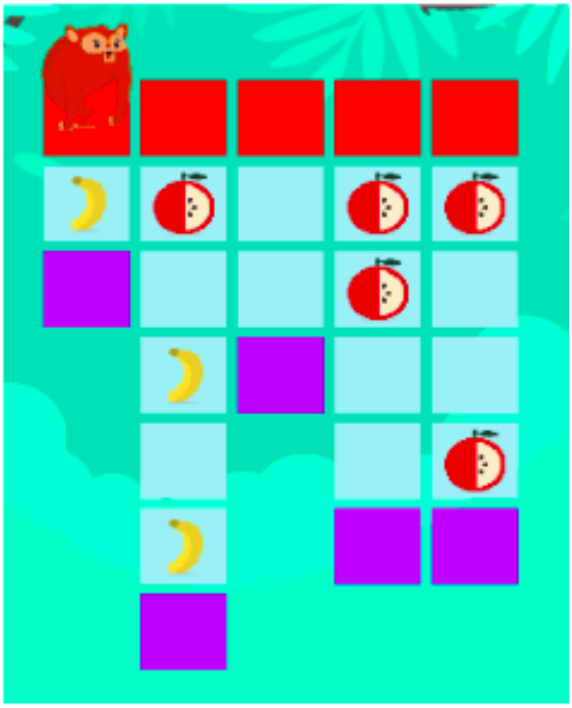


Fig. 4.2: El desafío le exige a le estudiante que desarrolle un programa para que el mono cuentes las bananas y las manzanas. Las manzanas y las bananas son objetos con los que puede interactuar el autómata mediante sus respectivos bloques “contar manzana” y “contar banana”.

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 específico 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 significa 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, configurar 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.

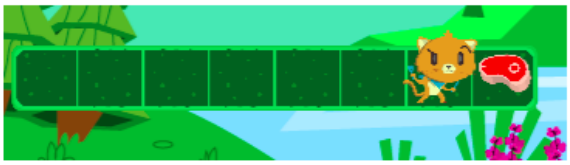


Fig. 4.5: Aquí se le indicó que repita 6 veces la secuencia de acción y el programa terminó sin llegar a comer el último churrasco. Esto se considera un error, pero el desafío no lo informa de manera explícita.

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 finalización/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

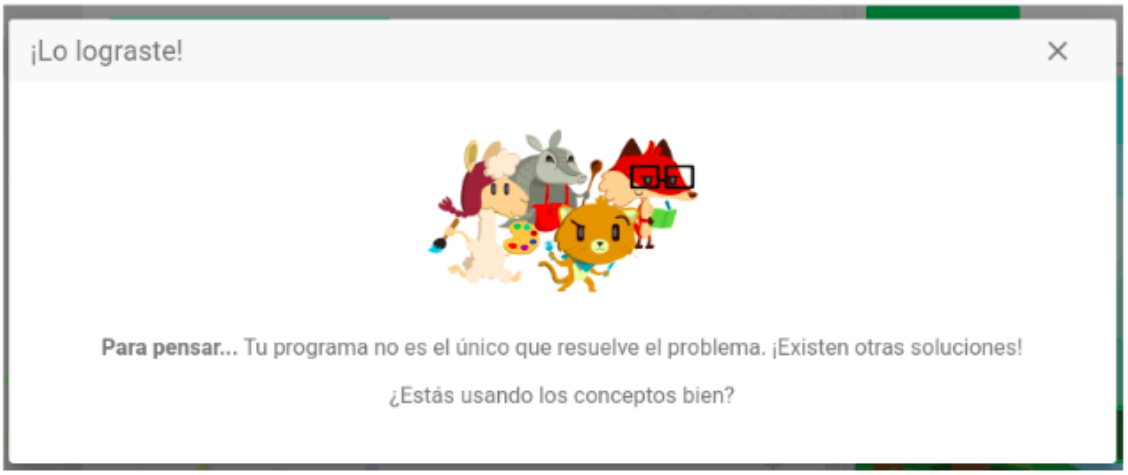


Fig. 4.7: En algunos casos la herramienta puede evaluar que se resolvió el desafío e indicarlo de forma explícita como el caso de esta imagen.

5. CLASIFICACIÓN DE LOS DESAFÍOS

Habiendo presentado estos aspectos de Pilas Bloques, estamos en condiciones de avanzar en una clasificación 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 definiremos y que serán relevantes a la hora de pensar la mejora en la herramienta.

Los clasificaremos 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 filas y M columnas.

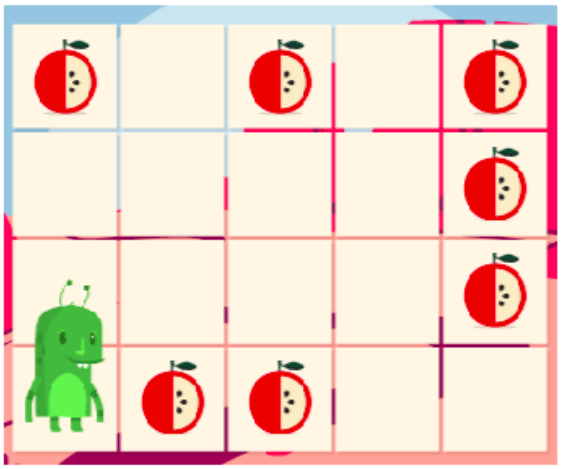


Fig. 5.1: En este caso el autómata (marciano verde) deberá comerse todas las manzanas (objetos).

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 predefinidos por grillas como comentamos antes.

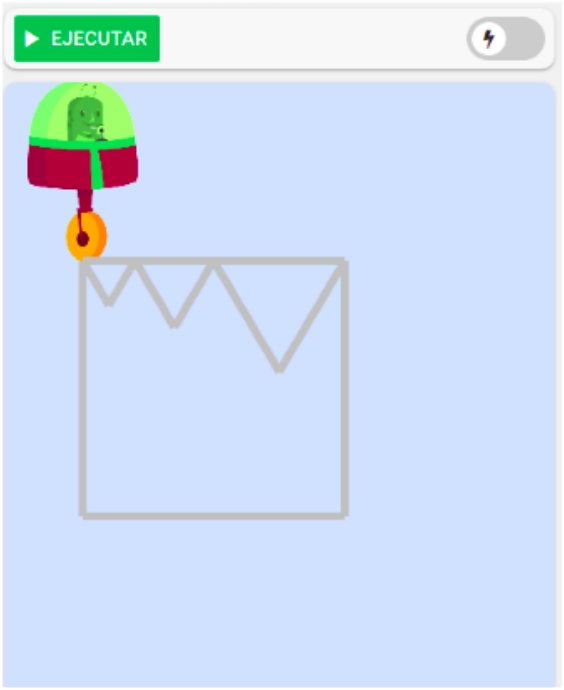


Fig. 5.2: Aquí se debe lograr que el marciano pinte de negro la figura sombreada en gris.

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 figuras de distintos tamaños.

5.3. Disposición de obstáculos

5.3.1. Escenarios con obstáculos fijos

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.

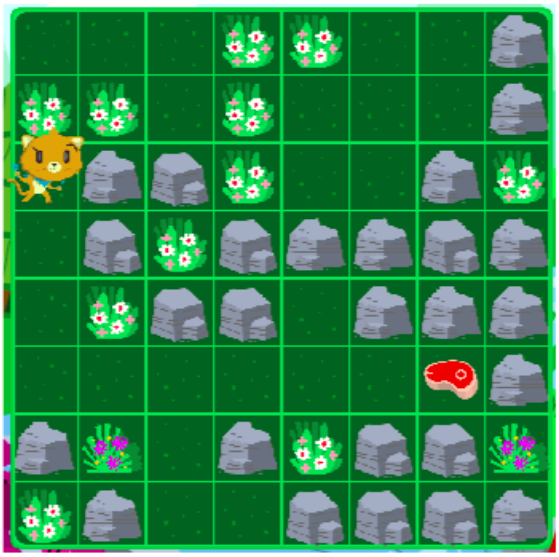


Fig. 5.3: En el desafío de la figura tenemos una grilla de 8x8. Nuestro autómata empieza en la posición (3,1) y debe llegar hasta el churrasco para comerlo.

En el desafío de la figura 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 fila hasta el final de la misma y patear la pelota. Cuando se apriete la opción de "ejecutar", la cantidad de filas 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.

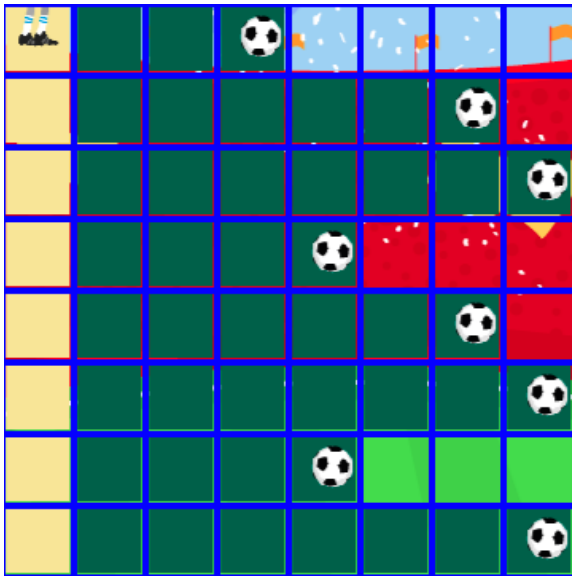


Fig. 5.6: Con esta imagen queremos ejemplificar cómo podríamos ver a este ejercicio como de obstáculos variables.

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 fila.

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

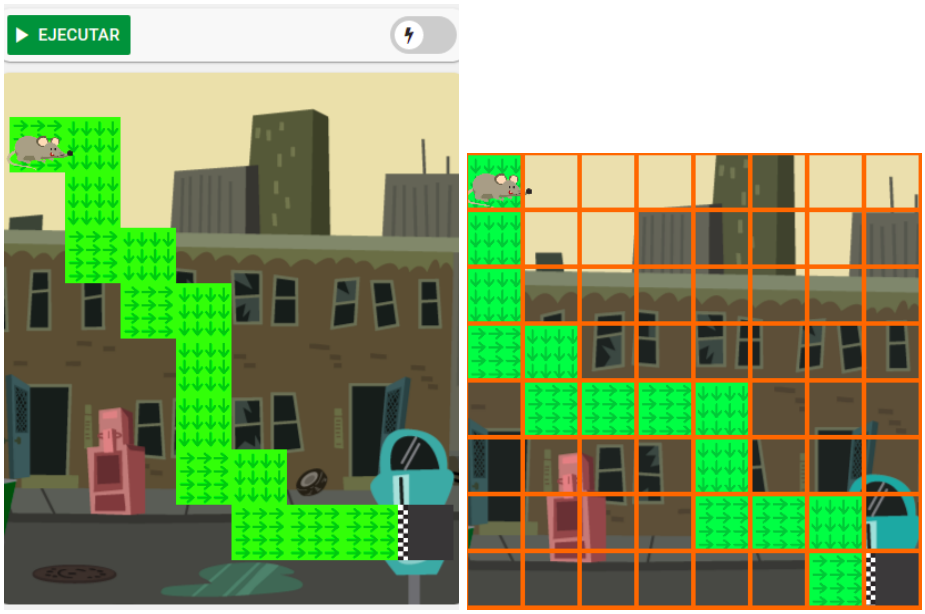


Fig. 5.7: En este escenario el ratón debe llegar hasta la casilla negra. El camino está compuesto por muchas casillas verdes que le indican al ratón cuál debe ser la dirección de su próximo movimiento, pudiendo ser para la derecha o para abajo. Cada vez que apretemos "ejecutar", si bien la cantidad de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

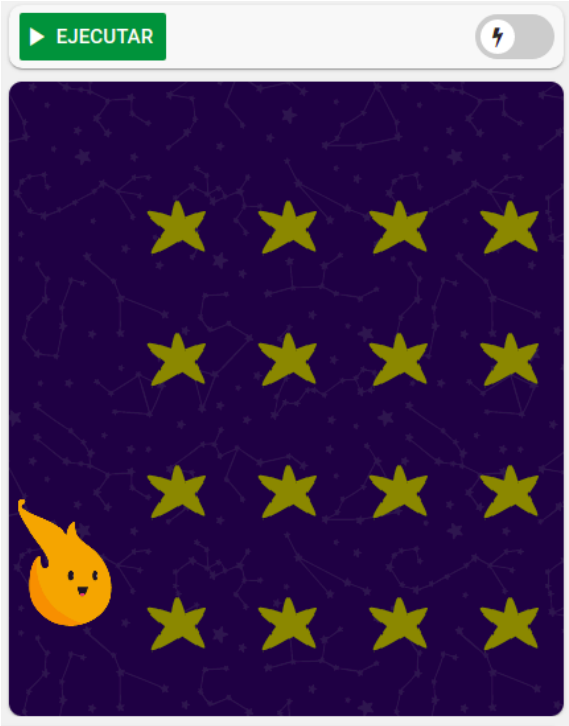


Fig. 5.8: En este caso el "fueguito" será nuestro autómata y deberá avanzar por cada fila y recolectar cada estrella. El objetivo es que el programa termine sin errores y que no hayan quedado estrellas por recolectar.

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.

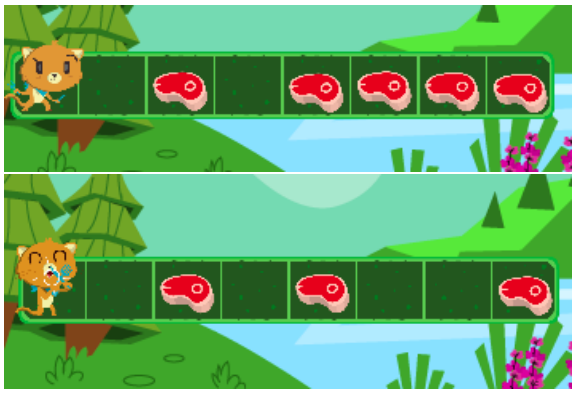


Fig. 5.9: En este caso el autómata deberá comerse todos los churrascos y se debe hacer un programa que resuelva esto sin importar la disposición de los mismos. En cada reinicio del problema la grilla se mantendrá igual pero cambiará la disposición de los objetos.

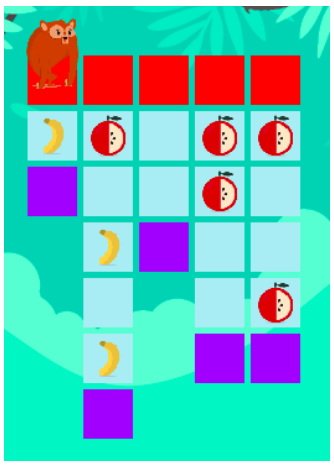


Fig. 5.10: En este ejemplo, lo que debemos lograr es que el mono cuente la cantidad de manzanas y bananas que hay. La disposición de estos objetos irá variando en cada reinicio.

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 fija 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.

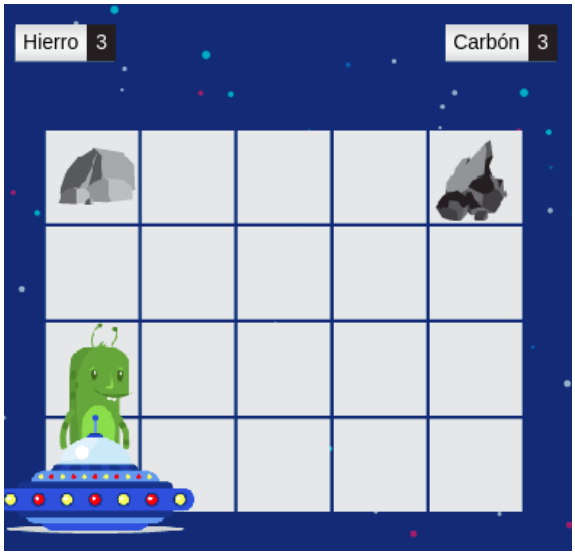


Fig. 5.12: Aquí el marciano debe reparar la nave para escapar en ella. Para hacerlo debe juntar las piezas de hierro y carbón.

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 finalización)

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 final. 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 final. 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.

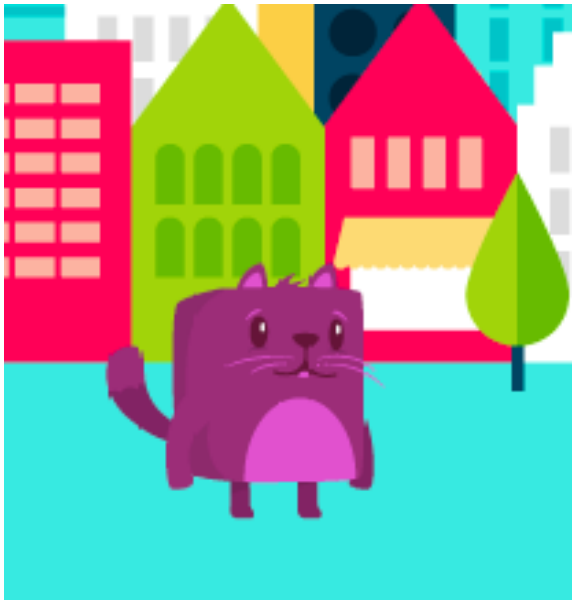


Fig. 5.13: Este es otro ejemplo de sucesión de estados. El objetivo del desafío es "Hacer que el gato avance un paso, se duerma, se despierte, salude y vuelva a su lugar".

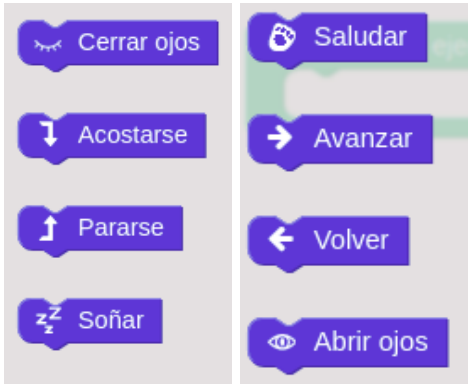


Fig. 5.14: Estos son los bloques con los que se dispone.

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 clasificación de ejercicios de Pilas Bloques, nos dispusimos empezar la investigación necesaria para definir la hoja de ruta que tenga como objetivo final 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:

- **¿Se podrá diseñar una plantilla genérica y única para definir cualquier desafío? Caso contrario, ¿Cuál es la mínima plantilla que maximiza la variedad de desafíos que se pueden construir?**
- **¿Hoy cómo define 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 modificando 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:

- **"¿Es técnicamente sencillo agregar un desafío previamente inexistente, dada la arquitectura actual de la aplicación?"**
- En otras palabras, **"¿cuál es el límite técnico de la flexibilidad de la arquitectura actual?".**
- Y posteriormente **"¿está 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 flexible para resolver la problemática de la creación de nuevos ejercicios sin que haya que modificar 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 flexibilidad para trabajar esta parte, presentando una propuesta de diseño que resuelva este aspecto a la hora de definir 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.

- **¿Qué sucede si pensamos en la posibilidad de recortar el espectro de ejercicios que podamos generar con esta mejora?**
- **¿Con ese recorte podríamos tener un conjunto de ejercicios que, en términos del diseño, sea fácilmente extensible?**
- **¿Tendríamos 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 unificada" 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 fijos y objetos interactuables variables. Las acciones no tendrán precedencia entre ellas y la condición de finalización será que no queden objetos interactuables en el escenario.

La razón por la que elegimos este tipo de ejercicios es porque aportan la suficiente flexibilidad 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" .

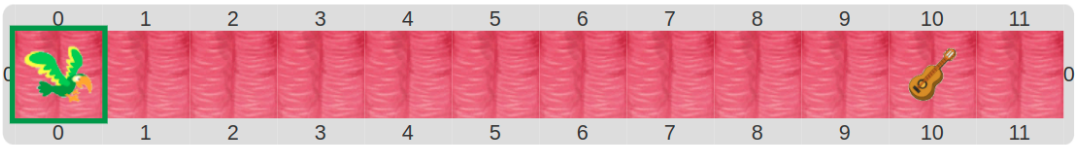


Fig. 6.1: El desafío consiste en que el loro debe moverse desde la casilla inicial hasta algún instrumento que haya, y tocarlo. Tanto el instrumento como la posición del mismo pueden cambiar, de modo que se debe escribir un programa que generalice a cualquier instrumento y que sea independiente de la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

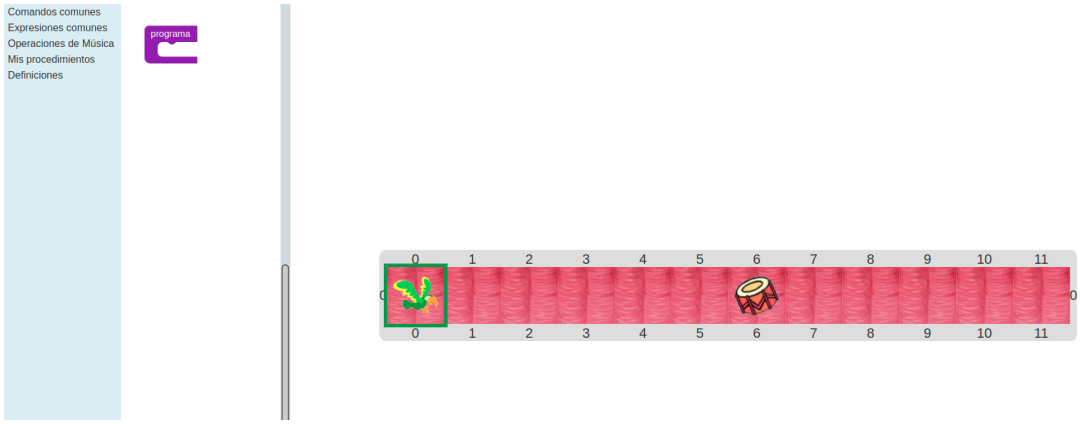


Fig. 6.2: Podemos ver que la interfaz de usuario del desafío tiene una estructura similar a Pilas Bloques, con un toolbox para elegir los bloques, un bloque principal donde armamos la solución al desafío, y por último el escenario.

Mediante un directorio local o de un repositorio en github (según si usa modalidad offline u online), Gobstones obtiene la información necesaria para definir un conjunto de ejercicios. Por ejemplo, en ese directorio están definidos 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.

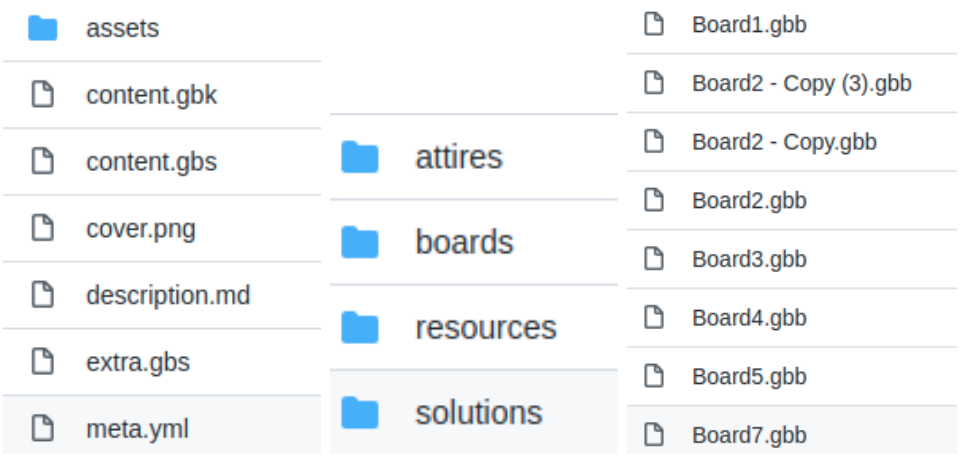


Fig. 6.3: Aquí podemos ver algunos directorios y contenido que hay del desafío en Github. Por ejemplo, los archivos .gbb que se ven a la derecha son las distintas configuraciones iniciales de la grilla. El equipo de gobstones decidió que la forma de aportar aleatoriedad al escenario es mediante la rotación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

Fig. 6.4: De la misma manera, hay un directorio donde se encuentran todas las imágenes que representan gráficamente 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 offline ya que los archivos y definiciones 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 definir 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 definir 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 definir uno como tal. Esto permite finalmente captar los aspectos que deberá tener en cuenta un docente para definir un nuevo desafío y poder afinar 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 fila de Duba y en alguna casilla que esté a la derecha del inicio del personaje.

7.1. Instrucciones disponibles

Una solución posible es

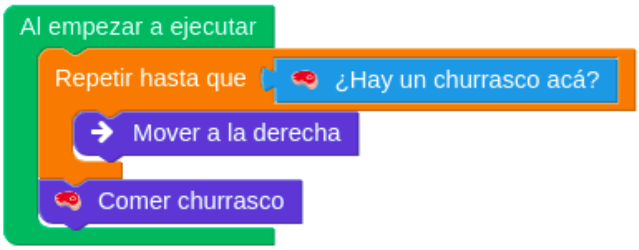


Fig. 7.2: El personaje se irá moviendo paso a paso hasta llegar al churrasco. Una vez que está sobre el mismo, dejará de moverse, lo comerá y el desafío estará terminado.

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

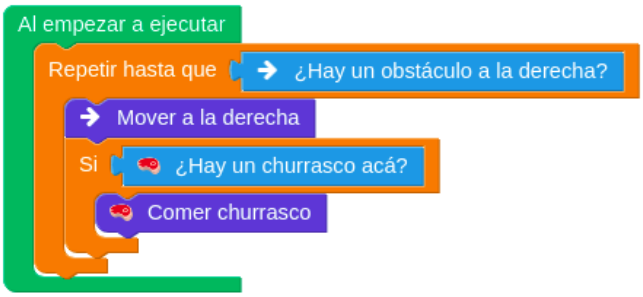


Fig. 7.3: En esta solución, el autómata irá caminando hasta llegar a una pared y se detendrá. Pero como en cada paso irá viendo si hay un churrasco y se lo comerá de ser así, tenemos la garantía de que al terminar se lo habrá comido.

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.

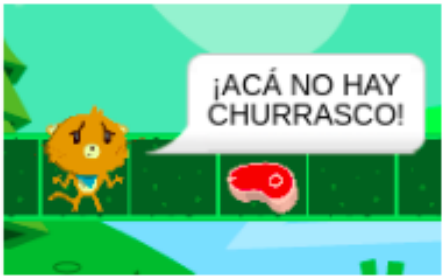


Fig. 7.4: Acá el autómata indica que se intentó comer un churrasco donde no había una. Si en este desafío en lugar de churrascos hubiesen manzanas, sería deseable que el mensaje del autómata cambie.

Para lograr esto, se podría permitir que le docente pueda configurar 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 configurable de modo que, entre otras cosas, se pueda dejar pistas o recordatorios de cómo mejorar el programa que armó le estudiante.

Por ejemplo: "¡Felicitaciones! ¡Lograste 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 definir las imágenes que le aportarán la estética al desafío.

Desde esta óptica, debería estar definido

1. Una imagen que defina 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 especificar lo siguiente

1. Cantidad de filas 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 defina lo listado a continuación.

1. Cantidad de filas 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. Definición 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("\
          [0,0,0,0,0,0],\
          [0,-,-,-,0,-],\
          [-,A,-,-,P,-],\
          [-,-,-,0,-,-],\
          [0,0,0,0,-,0],\")`,
   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 especifica el enunciado inicial que indica a le estudiante el objetivo a alcanzar en el desafío.

2. En el campo “bloques” están definidos 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 filas y seis columnas. La cantidad de filas está definida por la cantidad de arrays, y cada fila será un arreglo de caracteres.

Estos caracteres definirán cómo será esa fila. 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 flexibilidad para generar nuevos problemas para les estudiantes.

Ahora bien, al fin y al cabo, la que se encargará de hacer ese trabajo final será una clase en Typescript, que hoy por hoy no es configurable en muchos aspectos. Entonces si bien tenemos flexibilidad en definir 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 significa 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 predefinidos 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 configurables, y que pueda indicar el objeto con el que interactúa.

En conclusión, podemos decir que hoy por hoy no nos alcanza solamente con modificar valores de este archivo .json para poder definir un desafío completamente nuevo. Hay aspectos que queremos configurar 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.

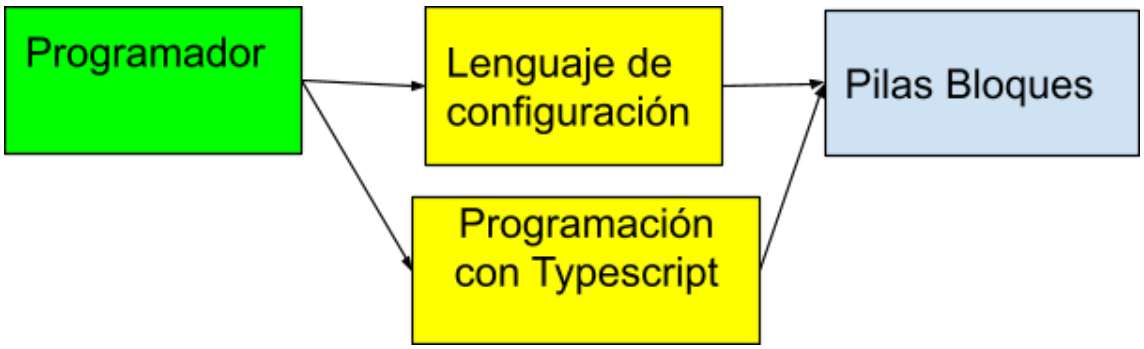


Fig. 9.1: Cómo se genera un ejercicio hoy en Pilas Bloques.

Se resume la idea de que para que Pilas Bloques tenga un nuevo desafío hay que proveer un input definido en el lenguaje de configuración (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 modificaciones para que la aplicación tenga la flexibilidad 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.

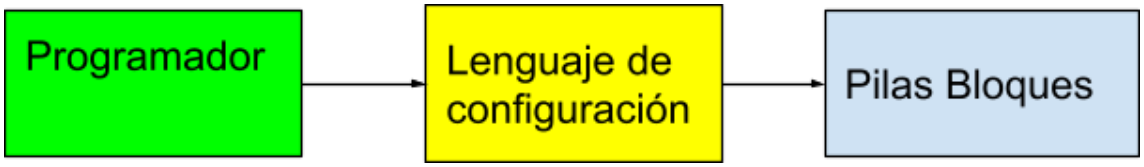


Fig. 9.2: Cómo se propone en esta tesis generar ejercicios.

Pensamos entonces que tenemos las siguientes tareas para abordar.

1. Extender este lenguaje de configuración 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. ¿Qué aspectos de este lenguaje de configuración 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 suficiente con poder agregar la especificación de los siguientes elementos.

1. Objetos interactuables: Los objetos con los que el autómata puede interactuar mediante una acción definida. 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 configurar 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 definir 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 definición 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 especificará 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 configuración. 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.

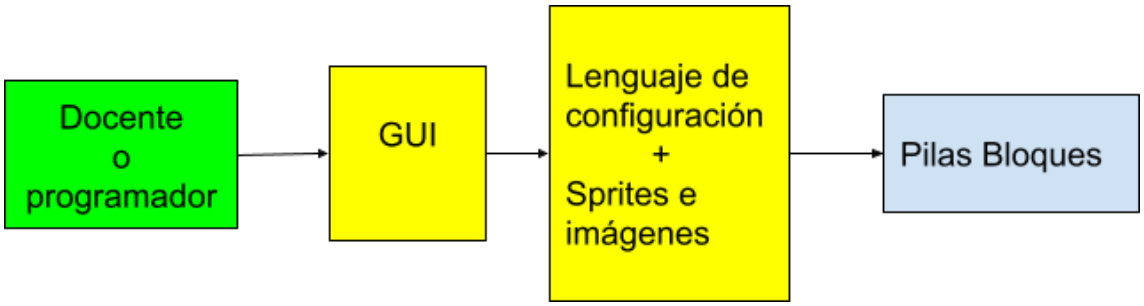


Fig. 9.3: Cómo debería ser el esquema ideal de creación de ejercicios (hoy factible gracias al trabajo de esta tesis)

Esta GUI podría ofrecer una forma interactiva para que les docentes (o inclusive desarrolladores de la aplicación) definan las características del desafío y que luego esta sea la que genera un .zip en donde esté el diccionario que contenga la definición 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 final del camino de la mejora, pero será un trabajo pendiente que no se abordará en esta tesis.

9.3. Extensión del lenguaje de configuración

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 modificados. 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 definir los valores. Explicaremos luego qué representa cada campo y sus respectivos valores.

```
{
  title: 'Desafio 1',
  scene: `"\
    [0,0,0,0,0,0],\
    [0,A,*,*,-,*],\
    [0,0,0,0,0,0],\
  "`,

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

  blocks: [
     { name: 'MoverACasillaAbajo' },
     { name: 'MoverACasillaArriba' },
     { name: 'MoverACasillaIzquierda' },
     { name: 'MoverACasillaDerecha' },
     { name: 'RecogerMedia', description: 'Recoger media', type: 'action
         ', interactsWith: 'media', errorMessage:'Aca no hay media!' },
     { name: 'RecogerPapel', description: 'Recoger papel', type: 'action
         ', interactsWith: 'papel', errorMessage:'Aca no hay papel!' },
     { name: 'hayMedia',  description: 'Hay media', type: 'sensor',
         object: 'media' }
  ],

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

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 clasificados 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 simplificar 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 fin 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 especifique información sobre el mismo. Los sprites de su animación estarán en un archivo con un nombre fijo que no será configurable, 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 especificación 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 gráfica. Podemos decir a su vez que el fondo no cambiará, de modo que con aportar una única imagen para el mismo será suficiente.

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

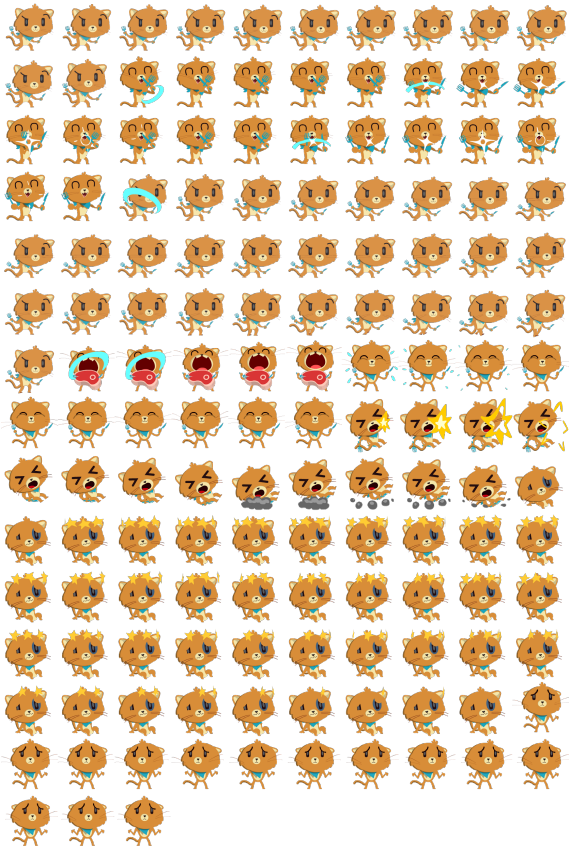


Fig. 9.4: En este caso, vemos un archivo que contiene cada sprite representando un estado de la animación de Duba. Tenemos las animaciones de cuando Duba se mueve, come un churrasco, o se choca con un obstáculo, entre otros.

Pilas Bloques define, 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 definir la animación de "comer churrasco" se llama al método de la siguiente forma

```
this.definirAnimacion("comerChurrasco",[0,0,61,62,63,64,65,65,65,65,66,67,68,69,70,71,72,73,74,75], 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 simplificarlo.

Lo que proponemos es que cada animación tenga una cantidad definida 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 simplificarlo siguiendo la propuesta de diseño de la tesis.

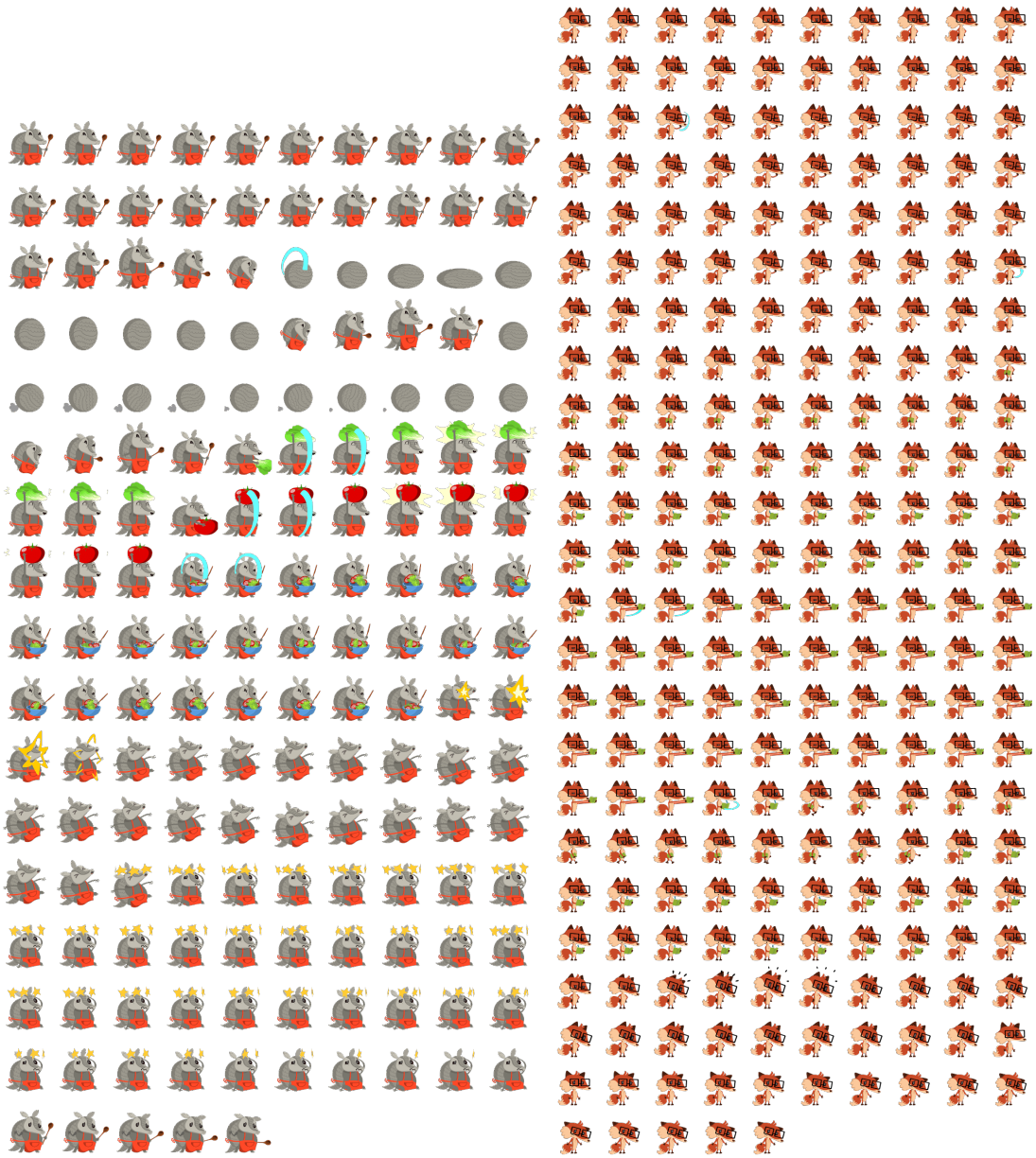


Fig. 9.6: Diseño original de Pilas Bloques (sprites distintos para distintos desafíos)

Fig. 9.7: Propuesta actual, extensible a cualquier personaje (mismos lugares para mismas animaciones)

Como podemos ver en ambos casos, los primeros diez sprites serán los del movimiento, los siguientes diez los de la interacción con un objeto, y por último los de colisión con un obstáculo.

Para no perder generalidad, los sensores no tendrán ninguna animación así que no será necesario incluir ninguna imagen para estos bloques.

En los casos de movimientos en distintas direcciones, los mismos utilizarán los mismos sprites de animación. De la misma manera, si contamos con varios tipos de objetos y cada uno deberá ser interactuado con bloques distintos, para no agregar complejidad podremos asumir que el autómata tendrá la misma animación para todos ellos.

Podemos ver en estos sprites que algunas animaciones muestran en algún momento los objetos con los que interactúa. Por eso lo ideal será utilizar imágenes sin los mismos de modo que puedan ser utilizadas de forma genérica.

En cuanto a los detalles de implementación, si contamos con dos bloques de movimiento y dos objetos A y B para interactuar con bloques distintos, se generará de forma automática el siguiente código para darle comportamiento al autómata:

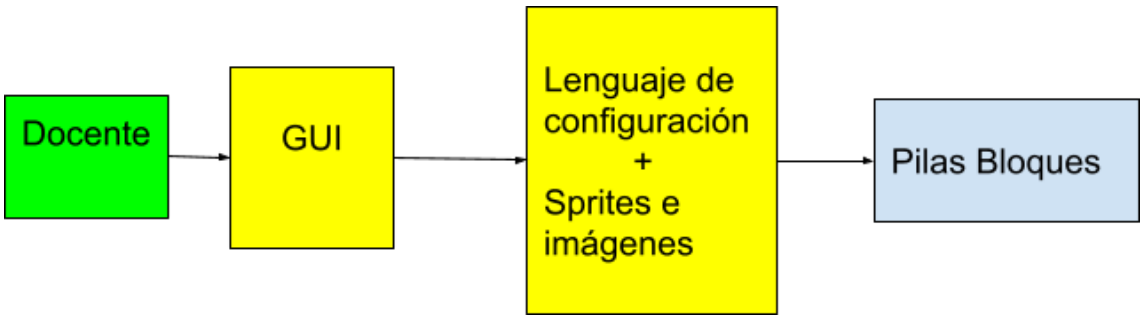
```
1 this.definirAnimacion("moverArriba",[0,1,2,3,4,5,6,7,8,9], 12);
2 this.definirAnimacion("moverAbajo",[0,1,2,3,4,5,6,7,8,9], 12);

4 this.definirAnimacion("interactuarConObjetoA",[10,11,12,13,14,15,16,17,1
      8,19], 12);
5 this.definirAnimacion("interactuarConObjetoB",[10,11,12,13,14,15,16,17,1
      8,19], 12);
```

Podemos asumir que todas las animaciones tendrán la misma velocidad en la transición entre los sprites, de ahí que el tercer parámetro es siempre el mismo número, doce en este caso.

9.5. Formato del directorio

Retomando entonces la idea del marco de trabajo al que queremos llegar.



Siguiendo con esta propuesta, el desafío estará autocontenido en un directorio.

- El lenguaje de configuración ya dijimos que estará plasmado en un archivo .json.
- Los "Sprites e imágenes" en una carpeta assets dentro de este directorio principal.

En el directorio assets estará

- automaton.png, que será un único archivo en el que estarán los sprites de las animaciones del Autómata.
- background.png, que será la imagen del fondo de la escena.
- Un directorio 'objects' y otro 'obstacles' con las imágenes para los objetos y los obstáculos respectivamente.

Como ya dijimos antes, si hay un bloque acción que indique que interactúa con una media, entonces en el campo "interactsWith" tendremos el valor "media". A raíz de eso la aplicación sabrá que hay un ícono media.jpg y que el mismo estará en el directorio 'objetos'.

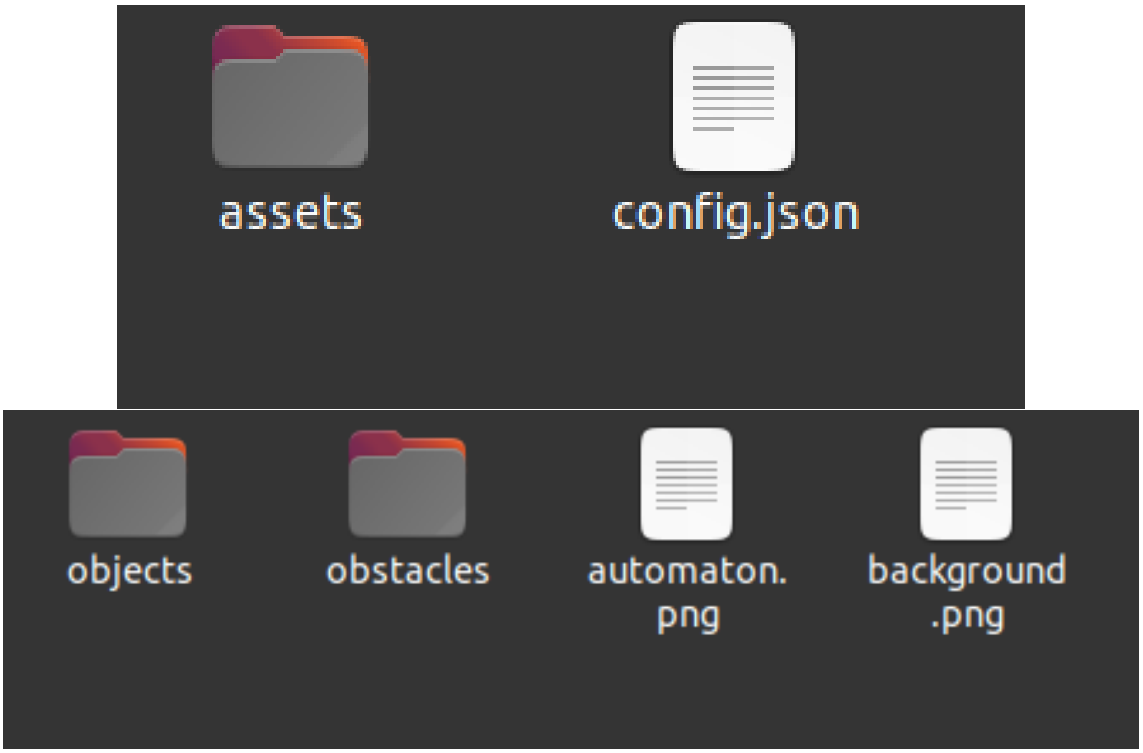


Fig. 9.8: Contenido de la carpeta del desafío. Tendremos una carpeta principal con el archivo .json de configuración y en la segunda imagen lo que irá dentro de Assets para las representaciones gráficas del desafío.

10. PRIMEROS PASOS DE LA IMPLEMENTACIÓN

Con este diseño presentado, nos propusimos asentar los cimientos para esta mejora, desarrollando un análisis y diseño (que es lo que se detalló hasta ahora). Luego veremos qué avances son factibles de realizar en el corto plazo a modo de prototipado.

10.1. Modificar el tipo de archivo que importa

Actualmente la aplicación cuenta con un mecanismo que permite importar ejercicios de un archivo con extensión "pb". Esto permite generar variantes de ciertos desafíos ya existentes en Pilas Bloques. Este archivo .pb contiene un string que representa al JSON, definiendo características de este desafío que se quiere generar. Pero no puede contener las imágenes que necesitamos para definir un desafío nuevo como lo dijimos en el diseño. Es por esto que evaluamos que lo primero es modificar este mecanismo para que admita un archivo comprimido que contenga el directorio, respetando el formato presentado en el diseño.

Para hacer esto, trabajamos con una biblioteca de JavaScript llamada 'unzipit' que permite leer un archivo .zip y consumir sus archivos. Por ejemplo, el JSON que define las propiedades del ejercicio podría estar dentro de este .zip, y utilizando esta librería se debería poder leer el archivo. Luego consumiendo las propiedades del mismo, sabremos qué bloques nuevos generar y qué archivos ir a buscar para ir definiendo el desafío.

10.2. Modificar los campos que toma del JSON

Una vez resuelto esto, fue necesario que Pilas Bloques se adapte a este nuevo formato de JSON. Como primer paso, nos pusimos como meta poder lograr que, a raíz del JSON en el que estén especificados los nuevos bloques que propone le docente, la aplicación los pueda mostrar en el toolbox del desafío.

A modo de experimentación, podríamos levantar un desafío preexistente en Pilas Bloques pero modificando los bloques disponibles. Veamos el siguiente .json que pensamos para hacer las pruebas:

```
{
    'escena': 'ElMonoQueSabeContar',

      'blocks': [

        { 'name': 'MoverACasillaAbajo' },
        { 'name': 'MoverACasillaArriba' },
        { 'name': 'MoverACasillaIzquierda' },
        { 'name': 'MoverACasillaDerecha' },

        {
          'name': 'RecogerMedia',
          'description': 'Recoger media',
          'type': 'action',
          'interactsWith': 'media',
          'errorMessage': 'Aca no hay media!'
        },

        {
          'name': 'RecogerPapel',
          'description': 'Recoger papel',
          'type': 'action',
          'interactsWith': 'papel',
          'errorMessage': 'Aca no hay papel!'
        },

        {
          'name': 'hayMedia',
          'description': 'Hay media',
          'type': 'sensor',
          'object': 'media'
        }
      ]
}
```

En este momento de la tesis, la aplicación no puede consumir la totalidad de los campos mostrados en el diseño propuesta. Es por eso que el JSON, varios son reemplazados por la existencia del campo con nombre "escena". Con el mismo, la aplicación sabe que debe entonces cargar el desafío mencionado y así poder hacer algunas pruebas sin necesidad de que esté implementada la totalidad de la modificación.

Sólo que ahora modificamos el proceso de carga del ejercicio para que tome el arreglo en el campo "blocks" y que cargue esos bloques.

Los de movimiento, al estar incluidos de forma preexistente en Pilas Bloques, alcanza con sólo indicar el nombre para que los instancie. El resto de los bloques cuentan con campos antes mencionados en el diseño.

Veamos primero los detalles de la implementación y luego mostraremos los resultados con algunas imágenes que muestran los resultados obtenidos.

10.3. Detalles de implementación

Pilas Bloques cuenta con un componente de una biblioteca desarrollada por Fundación Sadosky que provee la galería de bloques disponibles. El mismo está desarrollado con Blockly [11] y, en el contexto del desarrollo de Pilas Bloques, crea un objeto llamado "blocksGallery".

Este objeto permite agregar de forma dinámica (con ciertas limitaciones todavía) nuevos bloques. Esto se puede lograr con dos métodos del objeto llamados "crearBloqueAccion" y "crearBloqueSensor". Los bloques de Acción serán necesarios para que el autómata interactúe con un objeto y los bloques Sensores serán los que utilicen los condicionales para saber si el autómata está parado sobre un determinado objeto.

Por ejemplo el bloque "¿Hay manzana?" es un Sensor, y "Comer manzana" será un bloque Acción.

Veamos un ejemplo de lo que podemos hacer entonces para crear un nuevo bloque.

```
properties = {
               descripcion: description,
               icono: interactsWith+'.png',
               comportamiento: 'Recolectar',
               argumentos:`{etiqueta: ${interactsWith}}`
             }
this.blocksGallery.crearBloqueAccion(name,properties)
```

Aquí generamos primero un diccionario con propiedades relevantes para el bloque y luego se llama al método "crearBloqueAcción" con dos parámetros.

- "name" tendrá como valor el nombre que queremos que tenga el bloque. Este nombre no será el texto que contiene el bloque en la interfaz, sino un identificador único del mismo en la galería de bloques. De ahí que con los bloques preexistentes, con sólo poner el nombre nos alcanza ya que no habrá otro bloque con el mismo nombre. Se asume además que estos identificadores serán únicos.
- "Properties" serán características adicionales que ya especificamos previamente.
  - 'descripcion' : será el texto que se mostrará en el bloque durante el desafío.
  - 'Icono': En la interfaz de la aplicación, los bloques tienen además un ícono. Este campo indica el nombre de la imagen que estará como ícono del bloque. Por ejemplo, si el bloque es "comer manzana" el ícono será la imagen correspondiente al objeto con el que se interactúa.
  - 'comportamiento' es un valor que tienen los bloques que indica ciertas características generales de la acción. Si recordamos el análisis previo, dijimos que todas las acciones sobre objetos harán que los mismos sean consumidos por el autómata. Esto corresponde a un comportamiento de 'Recoletar'. De ahí que esté hardcodeado el valor ya que todos los bloques de acción que podamos agregar en esta mejora serán de recolección.
  - 'argumentos': Admite un json pasado como un String. Permite, entre otras cosas, indicar el objeto con el que interactúa el autómata al realizar la acción. El valor de 'etiqueta' será la precondición necesaria para poder realizar la acción (en este caso, que esté parado sobre el objeto indicado por la etiqueta), y si la misma no se cumple entonces habrá un error.

La siguiente línea lo que hace es agregar a la galería de bloques disponibles este último que definimos de forma dinámica.

```
    crearBloqueAccion(name,properties)
```

Sin perder generalidad, podemos decir que con los Sensores el proceso es similar, variando algunos campos que serán particulares para las propiedades de este tipo de bloques.

Luego más adelante se hace lo siguiente:

```
const bloques = [];

desafio.blocks.forEach(
    aBlock => {
            bloques.push(aBlock.name);
      }
)

desafio.bloques= bloques;
```

Aquí creamos una lista de los bloques que estarán en el desafío y luego

```
    desafio.bloques = bloques
```

es una adaptación que hacemos para que la aplicación más adelante sepa los identificadores de los bloques que queremos mostrar. Dado que ya estaban previamente agregados a la galería de bloques entonces los reconocerá de la galería y los podrá mostrar en el toolbox.

10.4. Resultados obtenidos

Veamos primero el desafío "El mono que sabe contar" que actualmente se encuentra en la versión de release de Pilas Bloques.

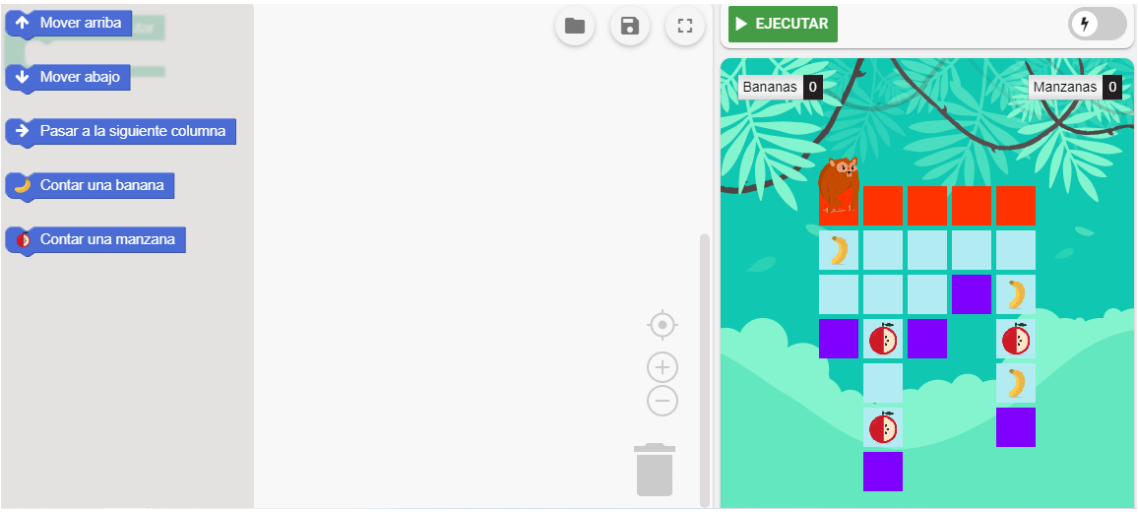


Fig. 10.1: En este caso el mono debe contar la cantidad de manzanas y bananas que hay en el escenario. Para eso disponemos de bloques de Acción para moverse y contar las frutas.

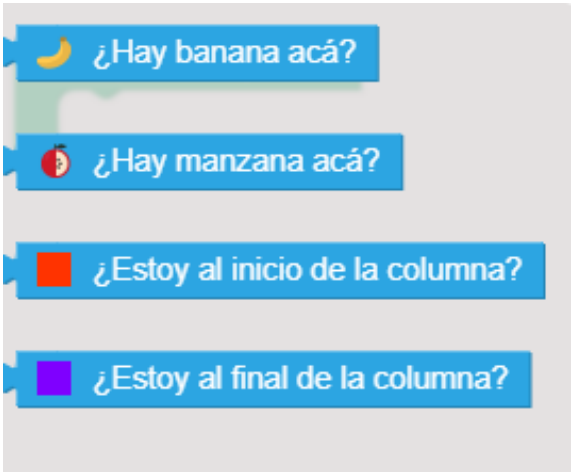


Fig. 10.2: A su vez cuenta con bloques Sensores que le permiten saber si está parado sobre una banana o sobre una manzana, entre otras cosas.

Como ya lo comentamos antes, notar que un bloque tiene un ícono en su parte izquierda y luego un texto que acompaña que indica lo que hace para que le estudiante lo entienda.

¿En relación a esto, qué avances pudimos lograr? Veámoslo en las siguientes dos imágenes.

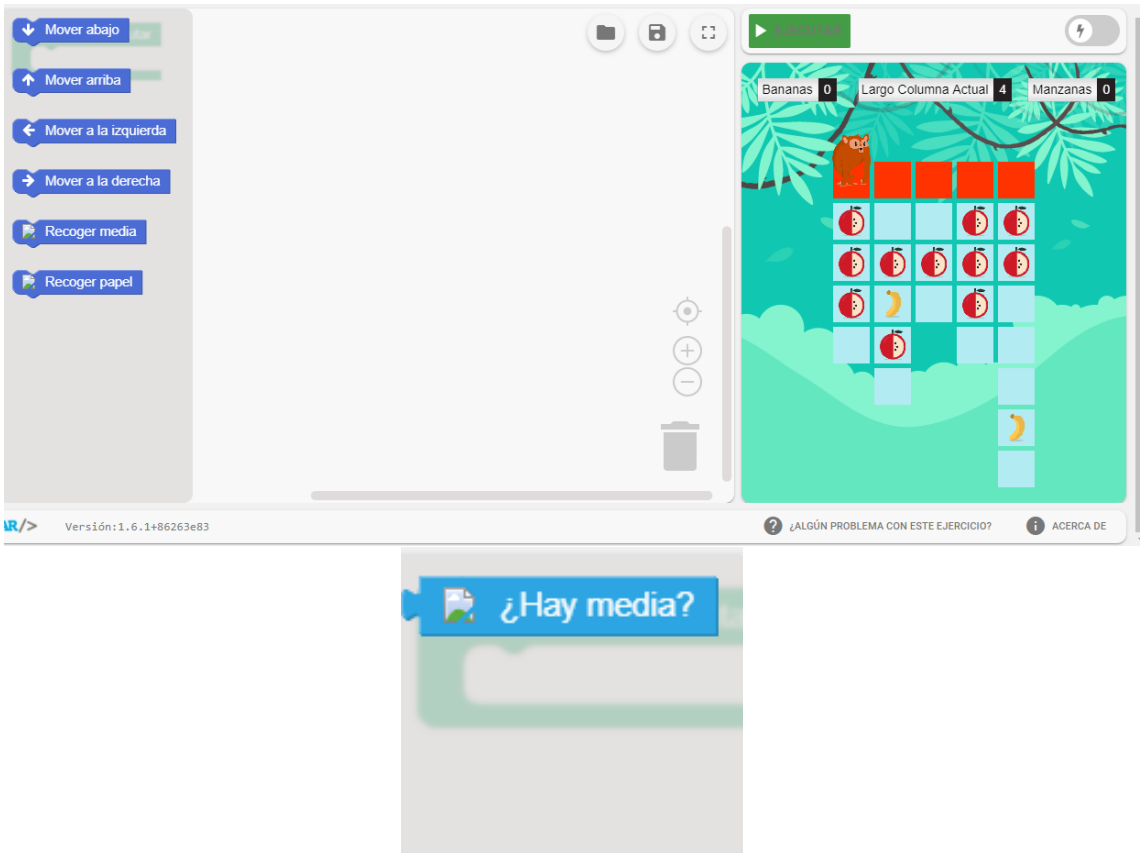


Fig. 10.3: Lo que podemos ver es que levantamos el mismo escenario pero modificando los bloques disponibles, ya sea en las Acciones como en los Sensores. En este caso generamos "Recoger Media", "Recoger papel" y "¿Hay media?" y pudimos sacar "Contar manzana" y "Contar banana".

Veamos ahora qué sucede si modificamos los campos de los bloques, quedando como el siguiente .json :

```
{
    "escena": "ElMonoCuentaDeNuevo",
    "blocks": [
    { "name": "MoverACasillaAbajo" },
    { "name": "MoverACasillaArriba" },
    { "name": "MoverACasillaIzquierda" },
    { "name": "MoverACasillaDerecha" },
    {
      "name": "EscribirTesis",
      "description": "Escribir tesis",
      "type": "action",
      "interactsWith": "media",
      "errorMessage": "Aca no hay media!"
    },
    {
      "name": "EntregarTesis",
      "description": "Entregar tesis",
      "type": "action",
      "interactsWith": "papel",
      "errorMessage": "Aca no hay papel!"
    },
    {
      "name": "estaTerminadaLaTesis",
      "description": "Tesis terminada?",
      "type": "sensor",
      "object": "media"
  }
  ]
}
```

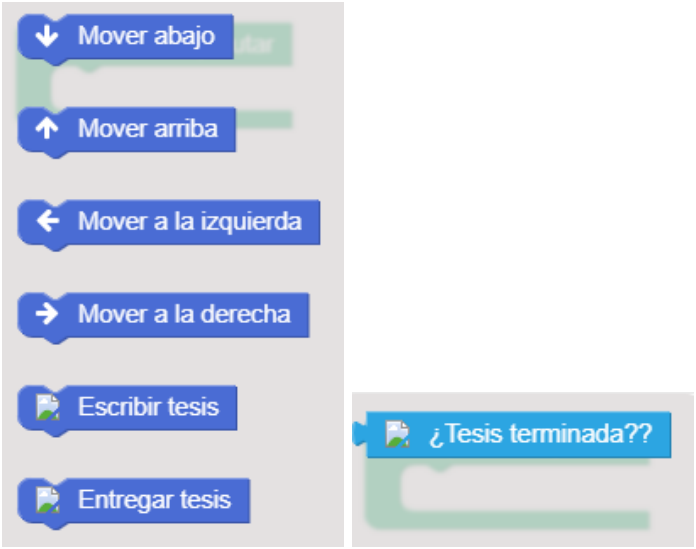


Fig. 10.4: Estos son bloques que aparecen con este nuevo JSON. Como podemos ver, cambiando los bloques definidos en el JSON, pudimos cambiar los bloques que aparecen de forma dinámica, sin necesidad de recompilar la aplicación o modificando su código.

10.5. Más avances en el desarrollo y nuevas actividades para agregar

Luego de algunos meses de trabajo, el equipo de desarrollo de Pilas Bloques tomó lo previamente implementado y desarrollaron un conjunto de mejoras adicionales que van en la línea de lo propuesto en la tesis. Hicieron modificaciones en cuanto a la implementación final por lo que ciertos detalles de implementación antes expuesto ya no están presentes en la aplicación, pero con las mismas ahora podemos implementar a modo de prototipado un conjunto de desafíos con las características analizadas en este trabajo.

A continuación, presentamos un ejemplo de actividad agregada junto a la justificación de la misma. Si bien podemos generar desafíos muy variados a modo de demostración, nos pareció una mejor idea tomar una secuencia actualmente implementada en Pilas Bloques y proponer actividades que se consideren aptas para mejorar la trayectoria de le estudiante.

Veamos la siguiente secuencia

Fig. 10.5: En esta secuencia, uno de los temas abordados es la utilización de repeticiones anidadas para resolver desafíos.

Por ejemplo, retomemos en detalle el desafío "El recolector de estrellas" para entender esto.

Fig. 10.6: En este caso, el fuego será el autómata. El mismo debe tomar todas las estrellas, las cuales serán objetos con los que podrá interactuar el autómata.

No existen obstáculos, pero sí podemos pensar que es un desafío con grillas. La grilla será una matriz de cuatro filas y cinco columnas, en donde en la primera columna no habrá estrellas.

Es importante recalcar que en la última fila, no se puede ir más arriba, como se muestra en la siguiente imagen.

Fig. 10.7: El autómata aquí le indica a le estudiante que no puede moverse una fila para arriba porque está en el límite de la grilla. La indicación es mediante el mensaje "No puedo ir para arriba"

De hecho, un error muy frecuente con el que nos solemos encontrar en el espacio de trabajo de los talleres es la siguiente situación.

Fig. 10.8: Si bien pudieron tomar todas las estrellas, aún así el fuego intenta ir a la siguiente fila. Esto es considerado un error ya que debería tomarlas sin llegar a esta situación

Se suele llegar a esta situación con el siguiente programa

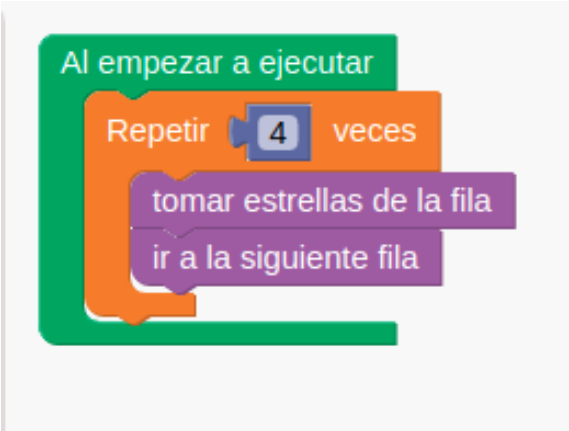
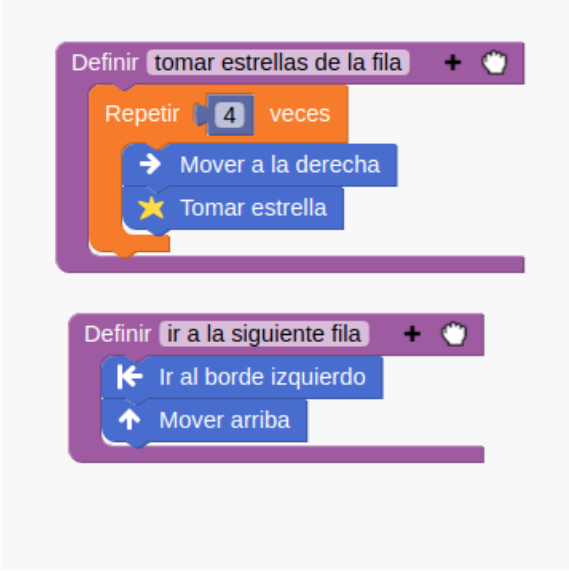


Fig. 10.9: El autómata, al estar realizando la última repetición, estará en la fila superior del escenario e intentará pasar a la siguiente, chocando con los límites de la grilla.

En el caso anterior, "tomar estrellas de la fila" e "ir a la siguiente fila" estarán definidos como



Una posible solución al problema entonces es hacer

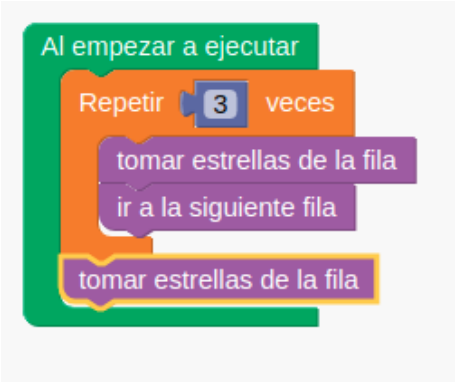


Fig. 10.10: Le indicamos al autómata hacer tres veces la repetición y, llegado a la última fila, sólo tomar las estrellas.

De alguna forma, estamos resolviendo el problema del "caso borde" que es la última fila.

Sin embargo, la solución más recurrente que suelen proponer les estudiantes en una primera instancia es la presentada más arriba. Aún más, una vez visto el problema del choque en la última fila, no es frecuente que propongan una solución correcta como la más reciente.

Se podría decir que a estas alturas del trabajo con Pilas Bloques no tienen incorporado el concepto de "caso borde" y algunas formas de abordar esta problemática.

Es por esto que se propone introducir dos desafíos que ayuden a incorporar mejor este concepto.

Fig. 10.11: Los desafíos propuestos estarán entre "El alien y las tuercas" y "El recolector de estrellas", señalados en la imagen con un rectángulo rojo.

10.6. Desafíos propuestos

El desafío "La Masacre de los Boniatos" consiste en hacer que el autómata pueda consumir todas los objetos que están en el escenario.

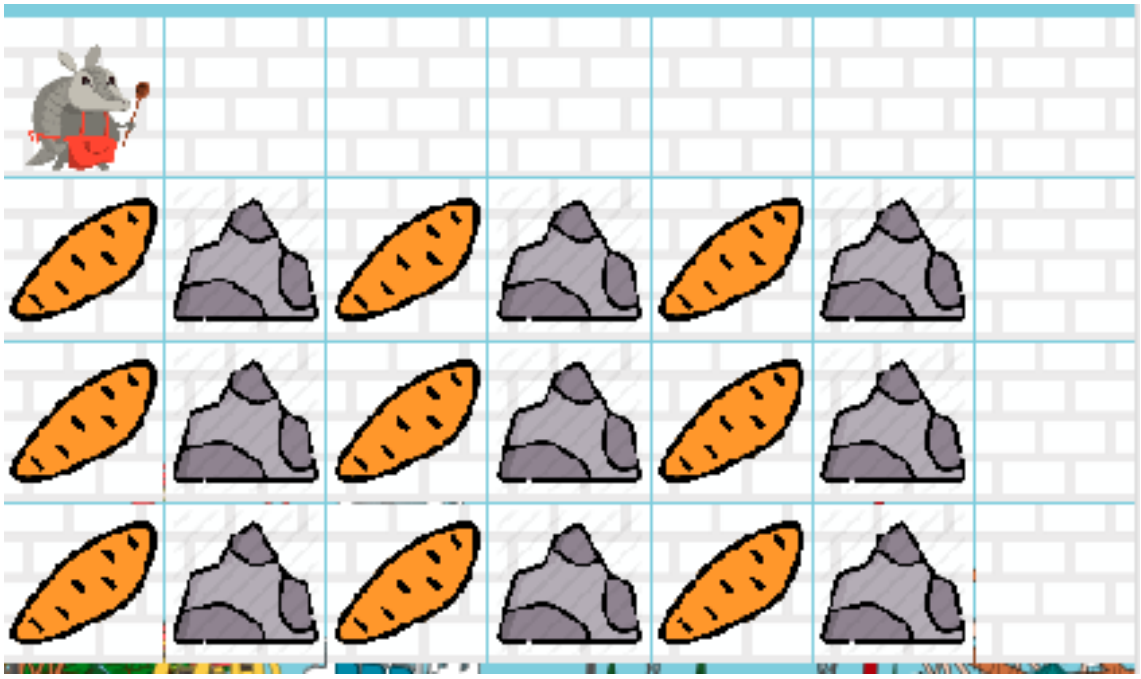
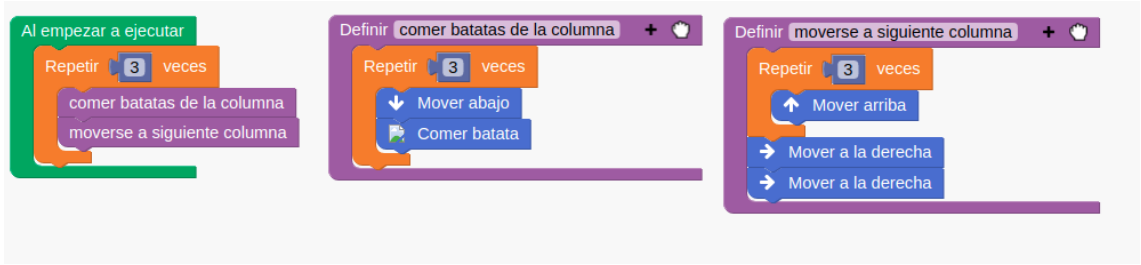


Fig. 10.12: Las piedras serán obstáculos y los objetos con los que puede interactuar el autómata son batatas (boniatos). Ante cada reinicio del desafío, el escenario será siempre el mismo.

Una posible resolución del problema podría ser



Con la solución recién propuesta, se llega a este escenario.

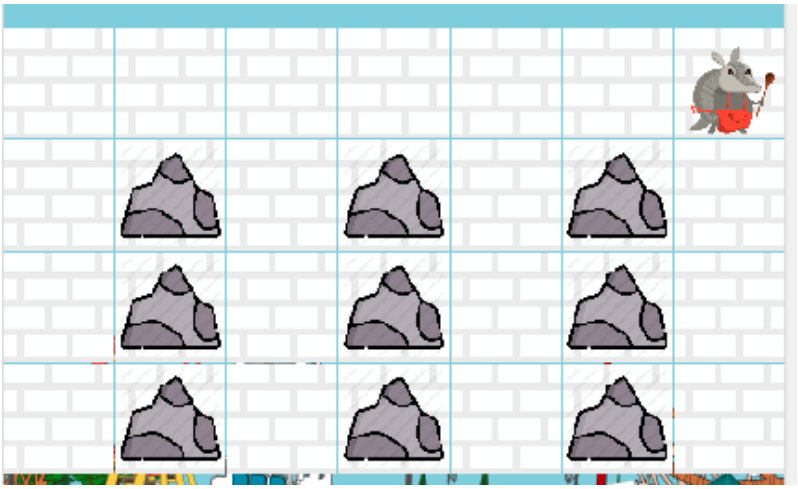


Fig. 10.13: Notar que, a diferencia del desafío "El Recolector de Estrellas", es posible esta resolución ya que contamos con dos columnas mediante las cuales podemos movernos a la derecha al realizar la última repetición y no nos chocamos con el borde de la grilla.

Veamos ahora este nuevo desafío, "Los boniatos contraatacan".

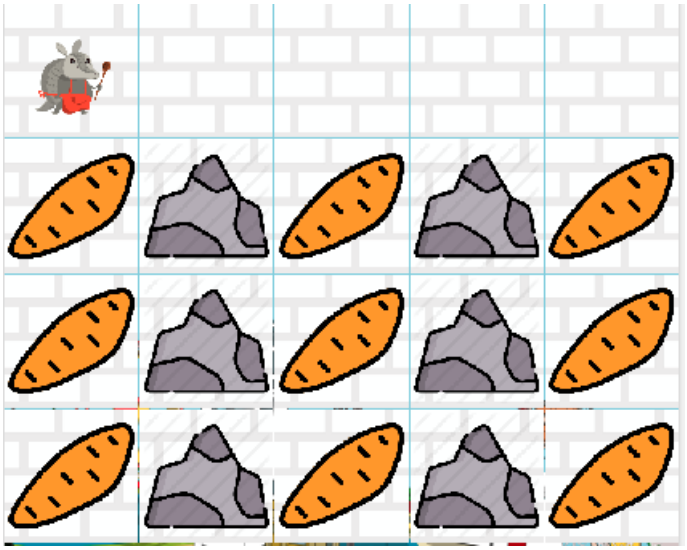


Fig. 10.14: Es similar al anterior, sólo que ahora no tendremos dos columnas adicionales para movernos a la derecha. Por eso mismo, la solución propuesta para el caso anterior no servirá, y deberán tratar la última columna como "caso borde"

La forma de encarar la solución será similar a lo antes expuesto con el desafío "El Recolector de Estrellas".

10.7. Justificación de las actividades propuestas

Lo que se quiere lograr con esto es que le estudiante pueda ver con claridad el problema del caso borde y elabore formas de resolverlo, y habiendo resuelto estos dos desafíos propuestos, "el recolector de estrellas" presentará menos dificultades.

Las actividades propuestas se han validado completando el Anexo "La programación y su didáctica - Plantilla de diseño de actividad" obtenida del curso "La Programación y su didáctica", dictada por Fundación Sadosky. La misma pretende sistematizar y formalizar la tarea de elaboración de nuevos desafíos y actividades para la enseñanza de la programación.

10.8. Armado de los desafíos con esta nueva mejora

Como dijimos antes, estos desafíos se pudieron desarrollar gracias a un conjunto de mejoras que se implementaron en Pilas Bloques. Las mismas ya se encuentran en la versión en producción de Pilas Bloques 1.8, pero están desactivadas por un feature flag hasta su estabilización y testing más profundo. Se espera que esté en producción a fin de año.

En lo que sigue, veremos algunas diferencias menores en relación a lo previamente diseñado y propuesto. Estas modificaciones fueron introducidas por el equipo de Pilas Bloques de acuerdo a necesidades suyas y otras decisiones de diseño e implementación

- En lugar de "objetos interactuables", se hablará de Prizes.
- Veremos además que algunos campos fueron traducidos al inglés, porque el equipo de desarrollo está privilegiando el código en este idioma.
- Veremos que algunos archivos no están explícitamente mencionados y que algunos se introdujeron en la implementación. A su vez, la disposición de los mismos en el directorio del desafío tendrá leves modificaciones.

Por otro lado, como esta es una primera aproximación para alcanzar el desarrollo final de la mejora, algunas cuestiones no fueron implementadas todavía.

Al momento de cargar la aplicación, contamos con una opción con la siguiente imagen.

Fig. 10.15: La aplicación provee la posibilidad de que en la opción "Importar desafíos" podamos ingresar un .zip, y con el mismo la aplicación puede generar los desafíos antes mostrados.

Veamos la composición de estos .zip para mostrar que siguen los lineamientos de diseño desarrollados en la tesis.

10.8.1. Archivo desafio.json

```
{
    "id": "unIdCualquiera",
    "grid": [
        "[aL,-,-,-,-,-,-],
        [pbatata,opiedra,pbatata,opiedra,pbatata,opiedra,-],
        [pbatata,opiedra,pbatata,opiedra,pbatata,opiedra,-],
        [pbatata,opiedra,pbatata,opiedra,pbatata,opiedra,-]"
    ],
    "blocks":[
        { "name": "MoverACasillaAbajo" },
        { "name": "MoverACasillaArriba" },
        { "name": "MoverACasillaIzquierda" },
        { "name": "MoverACasillaDerecha" },
        { "name": "Repetir" },
        { "name": "Procedimiento" },
        {
            "type": "action",
            "name": "Recoger por etiqueta",
            "description": "Comer batata",
            "interactsWith": "Prize"
        }
    ]
}
```

En el mismo tenemos los siguientes campos

- Id : Este campo responde a necesidades de la aplicación. Debemos completarlo con algún valor, pero no aporta elementos significativos al desafío.
- Grid: Podemos ver que es una lista con un string. El string es el que define la forma de la grilla.
  - Cada fila se define como una lista de caracteres, delimitados por corchetes como en una declaración de una lista, y cada casilla estará delimitada por una coma.
  - El string '-' define una casilla vacía
  - El string 'aL' indica dónde inicia el autómata
  - El string 'opiedra' está compuesto por dos partes. La 'o' indica que será un obstáculo, y lo que sigue será el nombre del mismo (luego se verá para qué se utiliza este nombre)
  - El string 'pbatata' sigue la misma regla que los obstáculos. La 'p' hace referencia a 'prize'.
- Blocks: Aquí se especificarán los bloques que se quieran tener disponibles
  - Como ya dijimos, en algunos casos esos bloques ya estaban previamente implementados en Pilas Bloques. Los mismos, con sólo indicar su nombre ya estarán incorporados en el toolbox del desafío.

- El caso del bloque “Procedimiento”, permite que se incorpore el armado de procedimientos propios, como el caso de “comer batatas de la columna” o “moverse a siguiente columna”
- En el caso de nuevos bloques, como el caso de “Comer batata”, se pueden incorporar especificando como se muestra en la anterior imagen. Notar que el campo campo ‘interactsWith’ no habla del objeto en cuestión sino que menciona ‘Prize’. Eso es porque por ahora no se puede diferenciar el objeto con el cual se interactúa, lo cual implica que, de tener varios objetos distintos, podrán ser todos interactuados por el mismo bloque de acción sin tener ningún tipo de mensaje de error por parte del autómata.

Veamos el contenido de los directorios en la implementación

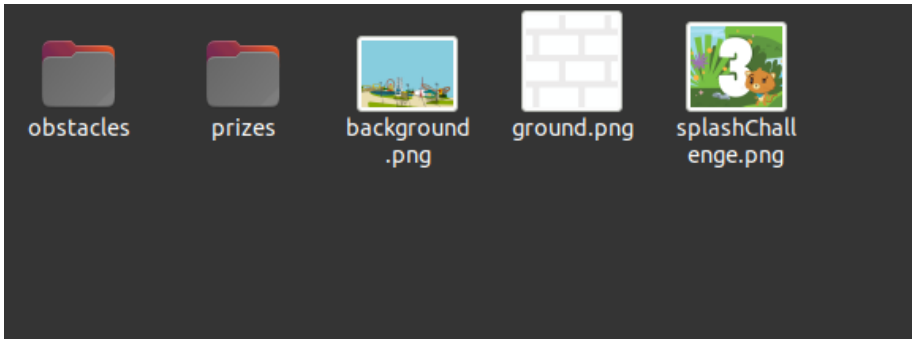


Fig. 10.16: Contenido de la nueva carpeta del desafío de acuerdo a los cambios de implementación.

Tendremos que

- La imagen de fondo estará definida en “background.png”
- La imagen de cada casilla de la grilla en “ground.png”
- El splasher del desafio definido en “splashChallenge.png”

Dentro de la carpeta prizes tendremos las imágenes que representan a los prizes.

Fig. 10.17: En este caso, tendremos dos prizes (objetos interactuables). Recordemos que en la grilla, los prizes se representaban como “p” concatenando el nombre del prize. Con ese nombre que concatenamos, Pilas Bloques sabrá qué ícono ir a buscar para mostrar el prize.

En el caso previamente mostrado, las papas no se utilizaron, pero se incluyeron aquí para exponer la posibilidad de contar con más de un tipo de objeto.

En el caso de los obstáculos, tendremos la imagen de la piedra para ubicarla como un obstáculo visible para le estudiante. Sin embargo, y a diferencia de lo especificado en el diseño de la tesis, también estarán las imágenes que representan los límites de la grilla. La aplicación buscará de forma predefinida un conjunto de imágenes y las ubicará de acuerdo a los nombres de cada una.

De la misma manera que con los prizes, cuando en la grilla encuentre un elemento cuyo nombre sea una 'o' y otro nombre concatenado, sabrá que es un obstáculo y el ícono del mismo lo buscará a partir de este nombre en la carpeta correspondiente.

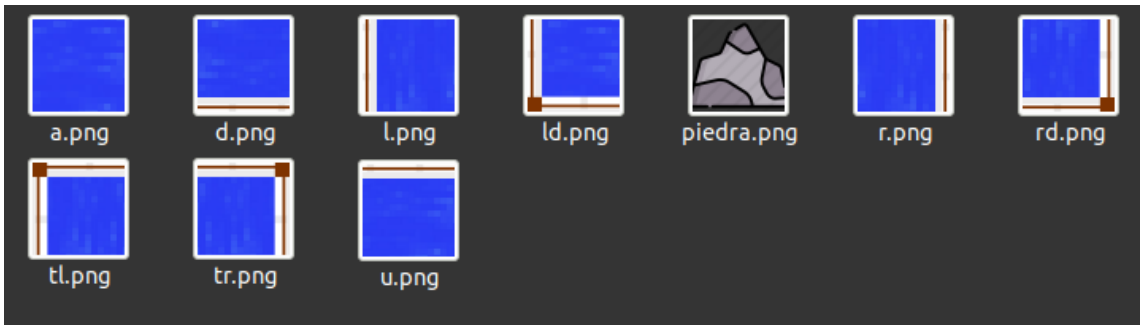


Fig. 10.18: En este caso tendremos como obstáculos a la piedra. Los íconos que no son la piedra, son los que se utilizan para definir los límites de la grilla. Por cuestiones de implementación, no podemos modificar el nombre de estos archivos.

Con todo esto, ya hemos explicado todos los aspectos que se deben incorporar en el directorio para definir un nuevo desafío para la actual implementación en Pilas Bloques.

11. CONCLUSIÓN Y PRÓXIMOS PASOS

Con lo desarrollado hasta este momento consideramos que se dieron grandes pasos hacia la mejora propuesta de la tesis. El nuevo diseño de las configuraciones de desafío fue validado mediante su implementación en la propia herramienta. Además, aunque la implementación fue parcial, se demuestra que es un mínimo suficiente para producir actividades que permiten saldar problemas en la secuencia didáctica existente.

Como dijimos previamente, las mejoras ya se encuentran en la versión en producción de Pilas Bloques 1.8, pero están desactivadas por un feature flag hasta su estabilización y testing más profundo. Se espera que esté disponible para el uso por parte de docentes a fin de año.

Intentando hacer un resumen de lo producido, se empezó el trabajo sin mucha claridad y con expectativas muy altas, apuntando a que esta mejora abarque todos los tipos de desafíos existentes en Pilas Bloques.

Luego, a partir de la experiencia del uso de la aplicación en la práctica real con escuelas, se pudo seleccionar un conjunto de desafíos relevantes en la enseñanza de la programación y así acotar el espectro de desafíos que podremos definir en esta primera instancia.

Por último, se elaboró el análisis y diseño de la propuesta, con una primera implementación que puede dar cuenta del trabajo realizado y con gran potencial a futuro para el crecimiento de la aplicación.

Habiendo llegado a este punto, creemos que los siguientes pasos serán:

1. Construir documentación sobre el formato del archivo de desafío para que cada característica esté al alcance de un docente sin necesidad de recurrir a esta tesis.

2. Que se puedan generar bloques de acciones diferenciados para cada objeto. Esto significa que, si bien actualmente podemos hacer distintos bloques cada uno con un nombre y una descripción propia, cualquiera sirve para interactuar con cualquier objeto. Los bloques de acciones deberán poder ejecutarse de forma exitosa sólo cuando se cumpla la precondición de que el autómata esté parado sobre el objeto que corresponda. Esto se necesita para justificar el uso de alternativas condicionales con sensores.

3. Lo mismo se deberá hacer con los bloques sensores. Es decir, deberán devolver el valor de verdad que corresponda según el objeto sobre el que está situado el autómata.

4. Que los nuevos bloques puedan obtener su ícono de un directorio en el .zip. Esto significa que aparezcan los nuevos bloques como ya mostramos anteriormente pero además que tengan el ícono correspondiente en su parte izquierda.

5. Que el autómata y sus animaciones puedan también ser especificadas en el directorio. Estas deberán estar definidas por un archivo con sprites de acuerdo a lo que se especificó previamente en la tesis.

6. Que aparezca el mensaje de felicitación a le estudiante, el cual deberá ser el indicado por le docente en el .json. Como dijimos anteriormente, el chequeo de que se resolvió el desafío por defecto será que no queden objetos por interactuar.

Por otro lado, se abre la puerta para pensar otros desarrollos que potenciarán aún más los alcances de Pilas Bloques. Se podría avanzar en:

1. El diseño e implementación de una UI para que le docente sin formación específica no necesite escribir un JSON a mano para generarlo. Esta UI deberá permitir parametrizar todos estos aspectos configurables del desafío que vimos en la tesis con el objetivo de que luego pueda generarlo. Deberá incluir cosas como:

   Editor de estructura de la escena donde se puedan indicar las dimensiones de la grilla y la posición de los objetos, obstáculos y autómata cuando empiece el desafío.

   Editor de sprites de personajes donde se puedan, por ejemplo, poner los sprites de movimiento, interacción con objetos y colisión con obstáculos. Estos se podrán poner de forma independiente y luego se exportará un archivo que incluya todos estos en un mismo archivo, respetando las especificaciones antes descritas. Se deberá también validar que la cantidad de sprites de cada animación es la correcta. Editor de bloques: En el mismo le docente podrá crear nuevos bloques, indicar su nombre, íconos, objetos con los que interactuará, etc. Importador de íconos e imágenes necesarias para terminar de completar cuestiones del desafío que no fueron incluidas anteriormente.

2. Dado que se quiere poder fomentar el desarrollo colaborativo de nuevos desafíos, será ideal pensar un repositorio al que docentes puedan subir los nuevos desafíos que arman y que eso sea accesible por otres.

Sumado a lo anterior, luego se podrá modificar la API de Pilas Bloques para que los desafíos puedan ser también obtenidos de un servicio Web sin necesidad de contar de forma local con el directorio.

Bibliografía

[1] Federico Aloi, Franco Bulgarelli, and Lucas Spigariol. Mumuki, una plataforma libre para aprender a programar. In *Memorias del 3er Congreso Nacional de Ingeniería Informática/Sistemas de Información, ser. CONAIISI. 2015.*, 2015.

[2] Arduino. `https://www.arduino.cc`, 2005.

[3] Computing at School Working Group. Computer science: A curriculum for schools. Technical report, Computing at School, 2012.

[4] Anne Beug. Teaching introductory programming concepts: A comparison of scratch and arduino. Master's thesis, Faculty of California Polytechnic State University, San Luis Obispo, 06 2012.

[5] Plan Ceibal. `https://www.ceibal.edu.uy/es`, 2019.

[6] Ching Chang, Yu-Ling Lin, and Chih-Kai Chang. Using visual programming language for remedial instruction: Comparison of alice and scratch. In Jhing-Fa Wang and Rynson Lau, editors, *Advances in Web-Based Learning – ICWL 2013*, pages 224–233, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[7] Stephen Cooper, Wanda Dann, and Randy Pausch. Teaching objects-first in introductory computer science. *SIGCSE Bull.*, 35(1):191–195, jan 2003.

[8] Hernán Czemerinski, Julián Dabbah, Claudia Rosana Floris, Franco Frizzo, Maria Carmen Leonardi, José Antonio Marone, María Virginia Mauco, Nerina Menchón, Martín Felipe Mezzanotte, and Ángeles Schang. *Ciencias de la computación para el aula : 1er. ciclo de primaria : libro para docentes.* Fundación Dr. Manuel Sadosky, Jul 2018. ISBN: 978-987-27416-6-2. `http://program.ar/manual-primer-ciclo-primaria/`.

[9] Tebring Daly. *Influence of Alice 3: Reducing the Hurdles to Success in a Cs1 Programming Course.* PhD thesis, University of North Texas, Denton, Texas, May 2013. University of North Texas Libraries, UNT Digital Library, https://digital.library.unt.edu.

[10] Pablo M. Factorovich and Federico A. Sawady O'Connor. *Actividades para aprender a Program.AR.* Fundación Sadosky, 2015. In Spanish. ISBN: 978-987-27-4161-7. `http://program.ar/descargas/manual-docente-descarga-web.pdf`.

[11] N Fraser et al. Blockly: A visual programming editor. *URL: https://code.google.com/p/blockly*, 2013.

[12] Marcos J Gómez and Pablo E. "Fidel" Martínez López. ¿scratch, python, o qué? In *Actas de las Primeras Jornadas Argentinas de Didáctica de Ciencias de la Computación: JADiCC 2021 / Sandra Boari ... [et al.]. - 1a ed. - Ciudad Autónoma de Buenos Aires*, pages 101–114. Fundación Sadosky, 2022.

[13] Filiz Kalelioğlu. A new way of teaching programming skills to k-12 students. *Comput. Hum. Behav.*, 52(C):200–210, nov 2015.

[14] Michael Kölling. Lessons from the design of three educational programming environments: Blue, bluej and greenfoot. *International Journal of People-Oriented Programming (IJPOP)*, 4:5–32, 07 2016.

[15] Pablo E. Martínez López. *Las Bases Conceptuales de la Programación. Una nueva forma de aprender a programar.* Author, E-Book, december 2013. In Spanish. ISBN: 978-987-33-4081-9. Ebook URL: `http://www.gobstones.org/bibliografia/Libros/BasesConceptualesProg.pdf`.

[16] Program.ar. Formación docente. `https://program.ar/formacion-docente/`.

[17] Program.AR. `https://program.ar`, 2013.

[18] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[19] Fundación Sadosky. `https://www.fundacionsadosky.org.ar/`, 2011.

[20] Alfredo Sanzo, Fernando Schapachnik, Pablo Factorovich, and Federico Sawady O'Connor. Pilas bloques: A scenario-based children learning platform. In *Learning Technologies (LACLO), 2017 Twelfth Latin American Conference on*, pages 1–6. IEEE, 2017.

11. ANEXO - 'LA PROGRAMACIÓN Y SU DIDÁCTICA - PLANTILLA DE DISEÑO DE ACTIVIDAD'

En lo que sigue presentaremos las fichas para el diseño de ambas actividades antes presentadas. Las mismas fueron diseñadas por Fundación Sadosky como forma de sistematizar, formalizar y validar el armado de nuevas actividades. La propuesta está orientada a cualquier entorno o práctica que se asocien con la enseñanza de la programación, y en este caso se hará en particular para Pilas Bloques con los dos desafíos antes mostrados.

A modo de resumen, la primera actividad ("La masacre de los boniatos") tiene el fin de presentar y familiarizar a les estudiantes con el dominio del problema y un esquema de resolución del desafío. El siguiente desafío ("Los boniatos contraatacan") pone como requerimiento que hayan hecho este primer desafío para estar familiarizados con la situación problemática, y fue pensado exclusivamente para introducir el problema del "caso borde" y el abordaje para su resolución.

11.1. Descripción de Ficha para Diseño de Actividades - La Programación y su didáctica - Plantilla de Diseño de Actividad

Se requiere armar una actividad para desarrollar en clase, que tenga un problema asociado, y para el cual se deberá determinar su solución, y evaluar soluciones alternativas (y soluciones incorrectas).

La actividad tiene que basarse en didáctica por indagación y sobre alguno de los temas ya vistos. Puede ser totalmente nueva, o una variante significativa sobre algún ejercicio de los que ya hicieron. La actividad puede requerir o no el uso de computadoras, y puede realizarse en cualquiera de las plataformas vistas. En el caso de requerir PC, alcanza con proponer el diseño+solución de la actividad (por más que no la puedan diseñar en el software asociado; pueden hacer de cuenta como que alguien la programará en el futuro). Es opcional implementar dicha actividad en el entorno.

Deberán completar los siguientes puntos.

- **Nombre de la actividad**
- **Objetivos de la actividad** : Indicar al menos un objetivo primario (e.g., "Que los estudiantes aprendan la Repetición Condicional"), y dos objetivos secundarios (e.g. "Que los estudiantes reconozcan la idea de repetición condicional en ejemplos de la vida real", "Que los estudiantes demuestren conocimiento sobre repetición condicional en ejercicios con y sin computadoras", "Que los estudiantes reconozcan la necesidad de la definición de una estrategia y división en subtareas").
- **Conceptos en juego** : Seleccionar los conceptos principales que se ponen en juego en la actividad.
- **Nivel educativo de la actividad**: Nivel educativo (jardín, primario, secundario, etc.) y grado-año-edad de la población objetivo de esta actividad.

- **Prerrequisitos para realizar la actividad** : Qué temas/conceptos deben haber sido vistos antes del desarrollo de la actividad (e.g., Se deberá conocer el uso de repeticiones simples).
- **Material necesario / Entorno**: Si la actividad requiriera computadora, indicar para qué entorno está pensada. (Pilas Bloques, Gobstones, App Inventor, etc.). Si no, indicar material necesario para realizar la actividad unplugged.
- **Duración de la actividad en minutos**
- **Descripción de la actividad (x ej. Escenario/s inicial/es, Primitivas)**: Incluir detalles de la dinámica de la actividad, de qué partes consta la misma, y cómo será la participación de los estudiantes y las intervenciones de los docentes, entre otras cosas. Debe quedar claro qué parte de la actividad se trabaja en forma individual, grupal y/o colectiva. Recordar dejar espacio para la exploración por parte de los estudiantes, así como los diferentes niveles de trabajo mencionados anteriormente.
- **Enunciado del problema**: Descripción de la situación problemática a intentar resolver por medio de la actividad. Incluir todos los detalles posibles de contexto. Es importante que el ejercicio a resolver esté debidamente problematizado (i.e., no alcanza con que se pida resolver algo, sino que debe estar planteado de forma de que se presente en un contexto adecuado, y que motive a la resolución).
- **Más información. Links a archivos adicionales / Observaciones**: Pueden incluir aquí hipervínculos a archivos con información extra sobre la actividad propuesta, las soluciones a discutir, o cualquier otra información que crean relevante. Sugerencia: hacer links a archivos en Google Drive, Dropbox u otros servicios de acceso público. Pueden acortarse con cor.to ó bit.ly .
- **Criterio de evaluación de las soluciones**: Qué cosas evaluarían en las soluciones al problema presentadas por los alumnos (e.g., el uso adecuado de la división en subtareas para alguna parte concreta del ejercicio). Qué les interesaría remarcar acerca de las posibles soluciones, tanto en los aspectos necesarios/requeridos para una solución correcta, como en las falencias típicas/posibles de aquellas que son erróneas (e.g., usar una repetición simple para resolver un problema que requeriría repetición condicional).
- **Conceptos en juego**:Seleccionar los conceptos principales que se ponen en juego en la actividad (Ver cuadro presentado a continuación de este listado)
- **Solución propuesta**

Conceptos en juego

	Comandos primitivos		Parametrización
	Estrategia de solución		Funciones
	Legibiliad y elección de nombres		Expresiones aritméticas (sumar, multiplicar, etc.)
	Subtareas y procedimientos		Expresiones lógicas (y, o, mayor, menor, etc.)
	Repetición simple		Eventos
	Alternativas condicionales		Variables
	Repetición Condicional, Recorridos		Otros:
	Sensores numéricos		

11.2. Ficha para Diseño de Actividad 1

Nombre de la actividad: La Masacre de los Boniatos

Objetivos de la actividad: Que les estudiantes resuelvan un desafío que requiere repeticiones anidadas. Dar pie a la actividad siguiente, para tratar el problema de los casos bordes.

Nivel educativo de la actividad: 1er año de secundaria

Prerrequisitos para realizar la actividad: Se deberá conocer el uso de las repeticiones simples y la secuencialidad entre acciones.

Material necesario / Entorno: Pilas Bloques

Descripción de la actividad (x ej. Escenario/s inicial/es, Primitivas):En una primera instancia el docente presentará el escenario e indicará que el mismo no varía, es decir que es siempre la misma disposición de boniatos.

Siguiente paso, analizará en términos de solución al problema las opciones con las que cuentan les estudiantes para la resolución. En este momento se espera que le docente induzca la idea de utilizar procedimientos y repeticiones para armar una solución.

Habiendo separado en dos partes esa repetición "comer los boniatos de la columna" y "pasar a la siguiente columna", les dejará a les estudiantes el trabajo de implementar cada uno de ellos.

Una vez visto que pudieron elaborar los mismos, se presentará la solución utilizando estos procedimientos en la repetición.

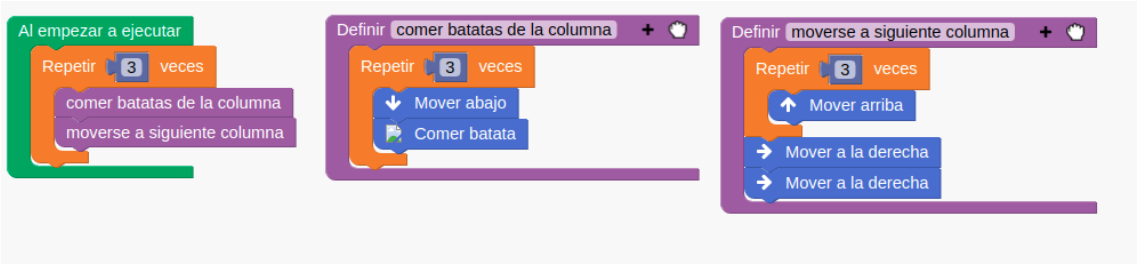
Enunciado del problema: Lita quiere comerse todos los boniatos. Ayudala a que pueda hacerlo sin chocar con las piedras.

Criterio de evaluación de las soluciones:

Se tendrán en cuenta dos aspectos.

- La capacidad de desarrollar los dos procedimientos propuestos.
- La capacidad de utilizar los mismos en una repetición como solución principal.

Solución propuesta:



Conceptos en juego

x	Comandos primitivos		Parametrización
x	Estrategia de solución		Funciones
x	Legibiliad y elección de nombres		Expresiones aritméticas (sumar, multiplicar, etc.)
x	Subtareas y procedimientos		Expresiones lógicas (y, o, mayor, menor, etc.)
x	Repetición simple		Eventos
	Alternativas condicionales		Variables
	Repetición Condicional, Recorridos		Otros:
	Sensores numéricos		

11.3. Ficha para Diseño de Actividad 2

Nombre de la actividad: La boniatos contraatacan

Objetivos de la actividad: Que les estudiantes puedan incorporar la problemática de "caso borde" y estrategias para resolverlo.

Nivel educativo de la actividad: 1er año de secundaria

Prerrequisitos para realizar la actividad: Debe haber resuelto el desafío "La masacre de los boniatos" utilizando la resolución propuesta para el mismo.

Material necesario / Entorno: Pilas Bloques

Descripción de la actividad (x ej. Escenario/s inicial/es, Primitivas):Se espera que al ver el escenario inicial puedan reconocerlo y que intenten resolverlo de la misma forma que el desafío anterior. Esto será sin intervención del docente.

Al haber avanzado en las propuestas de solución se deberá presentar la solución utilizada en el desafío anterior y analizar con el curso de forma conjunta cuál es el nuevo problema que se presenta.

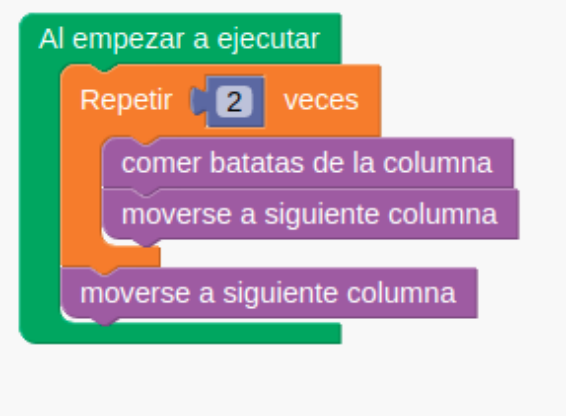
En este momento se deberá hacer explícito el inconveniente de que Lita no puede moverse a una columna de la derecha luego de comer los boniatos de la tercera columna. Es decir, debe quedar claro que la última columna se debe tratar como un caso particular distinto al resto.

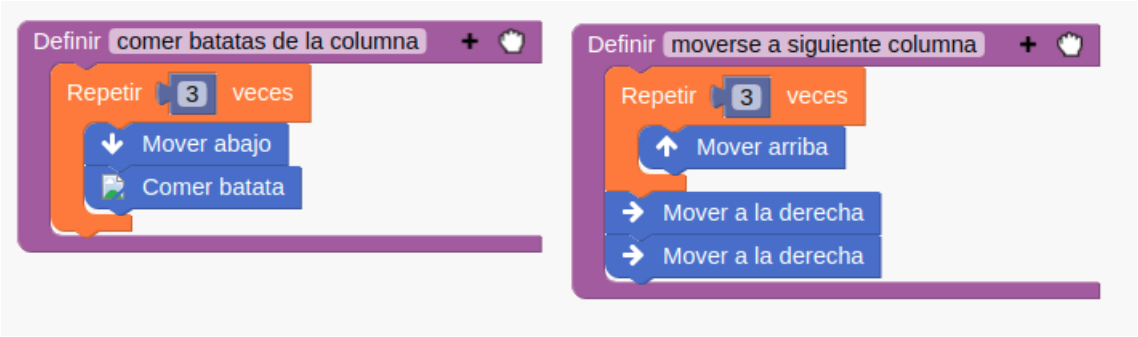
Será importante recalcar que no será necesario escribir nuevos procedimientos para resolver el desafío sino que deben poder resolverlo con lo ya elaborado, haciendo algunas modificaciones al programa principal.

Enunciado del problema: Lita quiere seguir comiendo los boniatos, pero los mismos volvieron mejor organizados y no le harán la tarea tan fácil. Podés ver las diferencias con el desafío anterior?

Criterio de evaluación de las soluciones: Se espera que pueda resolver la problemática del caso borde utilizando los procedimientos.

Solución propuesta:





Conceptos en juego

x	Comandos primitivos		Parametrización
x	Estrategia de solución		Funciones
x	Legibiliad y elección de nombres		Expresiones aritméticas (sumar, multiplicar, etc.)
x	Subtareas y procedimientos		Expresiones lógicas (y, o, mayor, menor, etc.)
x	Repetición simple		Eventos
	Alternativas condicionales		Variables
	Repetición Condicional, Recorridos		Otros:
	Sensores numéricos		