RÍO NEGRO
UNIVERSIDAD NACIONAL

Códimo: una Herramienta para el Diseño de Objetos de Aprendizaje

Trabajo Final de Carrera

Licenciatura en Sistemas

Autor: Graziani, Luciano Martín.

Director: Mag. Lugani, Carlos Fabian.

Codirector: Mag. Molinari, Enrique Pablo.

Fecha de presentación: mayo 2018

Agradecimientos

A mi familia, por apoyarme durante toda mi carrera.

A mis compañeros de estudio, porque con ellos persistí.

A mis profesores, el conocimiento y las oportunidades que me ofrecieron son invaluables.

A mi director y a mi codirector de tesis, que me apoyaron incondicionalmente con este trabajo.

A la Universidad, por ofrecer la posibilidad de estudiar esta carrera.

Resumen

Este trabajo final de carrera se enmarca en el tipo de diseño “Trabajo de producción” y tiene como propósito el desarrollo de una herramienta digital para construir ‘objetos de aprendizaje’. Ésta, denominada Códimo, permite adaptar actividades curriculares del Nivel Primario, de manera que el estudiante deba aplicar habilidades del ‘pensamiento computacional’ para resolverlas.

En primer lugar, se analizarán las aplicaciones actuales que tienen propósitos similares a Códimo y se justificará su desarrollo. En segundo lugar, se analizará el concepto ‘pensamiento computacional’ desde el punto de vista histórico con el fin de delimitar el alcance del concepto para Códimo. En tercer lugar, se analizarán distintos marcos teóricos para el uso de ‘objetos de aprendizaje’ en la escuela y se definirá cuál es el más indicado para este trabajo. Por último, se diseñará y desarrollará la herramienta digital a partir de la aplicación de la teoría del ‘framework de caja negra’, junto con el uso de patrones de diseño aprendidos durante la carrera.

En cuanto al uso de esta herramienta, se relatará la experiencia realizada con Códimo llevada a cabo con dos cursos de primaria —tercer y cuarto grado— de dos escuelas primarias de la ciudad de Viedma, Río Negro, durante la semana de la Ciencia y la Tecnología en el año 2017.

Como cierre de este trabajo se analizarán las decisiones tomadas para el desarrollo de esta herramienta digital, los resultados obtenidos, y se plantean líneas de trabajo a futuro.

Palabras claves: pensamiento computacional, objetos de aprendizaje, códimo, lodas, framework de caja negra.

Índice

Introducción

El propósito de este trabajo consiste en la creación de un framework para desarrollar 'objetos de aprendizaje'. Este framework se llamará Códimo, que es un acrónimo formado por las palabras CÓDIgo, IMaginación y MOtivación porque a través de actividades interactivas (juegos, desafíos o problemas, en otras palabras, objetos de aprendizaje), se pretende motivar al estudiante para que realice actividades cotidianas de sus materias de una forma distinta y que le permita utilizar la imaginación para resolverlas. Con Códimo se busca poder adaptar actividades curriculares de manera que el estudiante tenga que aplicar habilidades del 'pensamiento computacional' para poder resolverlas.

Diversas investigaciones relacionadas con la temática tecnología-educación dan cuenta de que el sistema educativo vigente en Argentina carece de entornos flexibles que permitan desarrollar las capacidades de autoaprendizaje, creatividad, autonomía, iniciativa y trabajo en equipo[1]. En respuesta a esto, el Consejo Federal de Educación de Argentina aprobó la Resolución 263/15, que establece que "la enseñanza y el aprendizaje de la "Programación" es de importancia estratégica en el Sistema Educativo Nacional durante la escolaridad obligatoria, para fortalecer el desarrollo económico-social de la Nación". Por eso, se considera que el 'pensamiento computacional' podría ser integrado dentro de la currícula, teniendo en cuenta que la computación es un área transversal a todas. Como dijo Jeannette Wing (2006), "tener habilidades del pensamiento computacional es un requisito indispensable para todas las personas" en la sociedad del conocimiento.

Peter Drucker (1967) acuñó el concepto de "Sociedad del Conocimiento", que a diferencia del de la "Sociedad Industrial" (basado en la producción industrial), considera al conocimiento y la tecnología como los elementos de mayor impacto para el desarrollo económico y social de las comunidades. Estos cambios en la tecnología informática y en los procesos de integración y globalización han dejado en evidencia que, para dar respuesta a las nuevas necesidades en el sistema educativo actual, es necesario que éste se adapte al nuevo contexto tecnológico y social.

[1] Sobre estas líneas de investigación ver Dussel, 2010; Piscitelli, 2009; Carneiro, 2009.

1. Objetivo principal

Desarrollar el framework de Códimo para diseñar, construir y probar los objetos de aprendizaje utilizando herramientas visuales y de programación.

2. Objetivos secundarios

- Seleccionar el marco teórico para la construcción de 'objetos de aprendizaje'.
- Realizar pruebas unitarias y de integración durante el desarrollo del framework.
- Generar actividades que cumplan con la currícula escolar de una asignatura y que requiera que el estudiante los resuelva aplicando el 'pensamiento computacional'.
- Desarrollar una aplicación web sencilla que permita ejecutar las actividades sin necesidad de instalación alguna.

3. Metodología de trabajo propuesta

Para concretar los objetivos propuestos, se estableció:

- *La determinación del lenguaje de programación*: está limitada según la restricción no funcional que se necesita cumplir y que establece que debe ser posible ejecutar la aplicación desde cualquier dispositivo que tenga un explorador de Internet instalado. Esto se debe a que una aplicación web (1) tiene mayor alcance que una aplicación móvil o de escritorio, (2) puede ser ejecutada desde casi cualquier computadora, sin importar el sistema operativo, aunque sí teniendo en cuenta que debe utilizar una versión actualizada del explorador web, (3) no requiere instalar ningún otro servicio además del explorador, (4) posee un ecosistema de librerías extremadamente vasto, y (5) existen autores de renombre que publican sus libros relacionados con esta temática con licencia de acceso abierto, por ejemplo *You Don't Know JS* (Simpson, K., 2014) y *Learning JavaScript Design Patterns* (Osmani, A., 2017). Por estos motivos se decidió utilizar JavaScript como lenguaje de programación.
- *La especificación de las herramientas a utilizar para la construcción de la aplicación*:
    - GIT: es un sistema de control de versiones distribuido, lo que significa que cada computadora que contenga un proyecto bajo este sistema tendría todas las versiones de este.
    - GitHub: plataforma para sincronizar proyectos GIT. Permite llevar seguimiento de los problemas, recibir aportes externos de manera controlada, definir objetivos a alcanzar, entre otras cosas.
    - Visual Studio Code: es un editor de texto enriquecido específicamente construido para desarrolladores. Es gratuito, de código abierto y tiene una vasta plataforma de accesorios que facilitan el trabajo del desarrollador.

- *La especificación de las librerías que serán utilizadas en el desarrollo de la aplicación*:
    - PixiJS: es una librería profesional de animación y renderización para la web.
    - Blockly: con esta librería se podrán construir bloques similares a los del rompecabezas, cada uno con una función específica. Cada bloque contendrá en su interior código que será ejecutado sin que sea visto por los estudiantes.
    - ReactJS: es una librería de renderizado de vistas web. Permitirá la integración de Blockly con PixiJS.
- *El cumplimiento de reglas de estilo que permitan tener un código limpio y legible*.
- *El cumplimiento de las buenas prácticas de desarrollo de software para las tecnologías que se utilicen, con el fin de asegurar que se escriba código reutilizable*. Para ello se utilizarán tres herramientas: ESLint, Flow y Stylelint. Las dos primeras realizan un análisis sintáctico del código Javascript, sin embargo, ESLint sólo revisa que se cumplan las reglas que se configuran, mientras que Flow verifica que los tipos de datos de las variables, llamados a funciones, instanciación de clases, entre otras operaciones con datos, sean consistentes. Stylelint cumple el mismo rol que ESLint, pero para el código del estilo visual de la aplicación (CSS, Cascade Style Sheet, Hoja de Estilo en Cascada).

4. Estructura del trabajo

La tesis se organiza en una introducción, tres capítulos y la conclusión y trabajos futuros:

La introducción presenta un panorama general, incluye los objetivos, la metodología y el estado del arte.

El capítulo 1 se focaliza en el 'pensamiento computacional'. Este concepto es clave para entender por qué es necesario integrar la computación a la educación. Además, describe un conjunto de habilidades implicadas en las actividades que contendrá Códimo.

En el capítulo 2 se ofrece un panorama actual de los marcos teóricos para el desarrollo de objetos de aprendizaje y se justifica la perspectiva adoptada desde la cual se enmarca el 'objeto de aprendizaje' de Códimo.

El capítulo 3 describe y explica el desarrollo del framework de Códimo.

El capítulo 4 describe las tres actividades que se desarrollaron: *Recta Numérica, Hola Códimo y Múltiplos y Divisores*, en ese orden. Al final se incluyen reflexiones de los resultados obtenidos.

La conclusión y trabajos futuros incluyen las reflexiones finales sobre los objetivos propuestos en el trabajo, las dificultades encontradas y se especifican los lineamientos a seguir en el futuro.

5. Estado de la cuestión en herramientas similares a Códimo

Existen diversas aplicaciones dentro del espacio en el que se pretende incluir Códimo. Algunas, como CodeCombat[2] o Pilas Engine[3]. Se trata de aplicaciones web para jugar o crear juegos y animaciones (respectivamente), en las cuales es necesario programar para poder realizar acciones como: mover un personaje a un determinado lugar, tomar algún objeto, o atacar a determinada criatura. Otras utilizan bloques similares a Códimo como Alice[4], Scratch[5] o Pilas Bloques[6] y otras ofrecen recursos educativos de forma gratuita con contenido en Ciencias de la Computación y otras disciplinas como Khan Academy[7].

Además de estas aplicaciones, también existe La Hora del Código[8], un movimiento global que brinda actividades introductorias de una hora de duración a las Ciencias de la Computación, "*diseñada para mostrar que todo el mundo puede aprender a programar y así comprender los fundamentos básicos de la disciplina*" (La Hora del Código, 2017). Este encuentro se lleva a cabo cada año durante la Semana de la Educación en Ciencias de la Computación[9], la cual celebra el nacimiento del Almirante Grace Murray Hopper (9 de diciembre de 1906) quien fue una pionera en la computación.

Según su sitio, la Hora del Código está organizada por Code.org, una organización pública sin fines de lucro dedicada a promover la participación en escuelas e institutos de las Ciencias de la Computación, con una especial atención en incrementar el número de mujeres y estudiantes de colectivos minoritarios que aprenden a programar. En Argentina, la Hora del Código es promovida por la fundación Sadosky (con el acompañamiento del Ministerio de Ciencia, Tecnología e Innovación Productiva) a través de la iniciativa Program.ar[10].

Un elemento realmente importante de la Hora del Código es el repositorio de actividades que posee. Cualquier persona y organización puede presentar una actividad en el sitio, entre las cuales se encuentran Khan Academy, CodeCombat, Alice, Scratch, y muchos más. La principal diferencia entre la Hora del Código y Códimo, aunque sutil, es que la Hora del Código tiene como objetivo dar a conocer la programación, no el pensamiento computacional, en cambio Códimo se centra en la aplicación de las habilidades del 'pensamiento computacional'. Por supuesto, al introducir a los estudiantes en la programación, también se introduce en el pensamiento computacional; sin embargo, no necesariamente es a la inversa.

[2] Página oficial: https://codecombat.com.
[3] Página oficial: http://pilas-engine.com.ar.
[4] Página oficial: https://www.alice.org.
[5] Página oficial: https://scratch.mit.edu.
[6] Página oficial: http://pilasbloques.program.ar.
[7] Página oficial: https://es.khanacademy.org/about.
[8] Página oficial: https://hourofcode.com/ar.
[9] Página oficial: https://csedweek.org.
[10] Página oficial: http://program.ar/de-que-se-trata-jyd.

5.1. CodeCombat

CodeCombat es un RPG (Role Playing Game, Juego de Rol) que consiste en manejar un personaje al que se le personalizan ciertos elementos (como las armaduras, armas o habilidades). A medida que se avanza en el juego, el personaje consigue mejores objetos y habilidades, se hace más fuerte y más complejo de usar. En el caso de CodeCombat, el personaje que se utiliza es manejado a través de código de programación (Python, JavaScript, CoffeeScript o Lua, específicamente). Por otra parte, el personaje tiene un conjunto de habilidades limitadas por los objetos equipados. De esta forma, los movimientos, los mecanismos como iteraciones o condicionales, o los métodos especiales (como saltar o lanzar un hechizo), deben seleccionarse con cuidado según el nivel que se busca solucionar, ya que, sin los objetos adecuados, resulta muy difícil o simplemente imposible.

Figura 1. Captura de pantalla del sitio de CodeCombat. Un personaje, los objetos que tiene equipados y las habilidades de uno de ellos.

CodeCombat presenta varias desventajas. Un aspecto es la traducción. No sólo no está completa, sino que, una vez terminado los niveles básicos, de a poco se va mezclando el inglés con el español, ya sea en los diálogos, en los nombres de las habilidades o de los objetos, o en las descripciones de los niveles. Este impedimento no sería de gran importancia si se estuvieran analizando juegos para estudiantes avanzados de nivel secundario. Sin embargo, el objetivo primordial es introducir el pensamiento computacional en estudiantes de nivel primario.

Otra desventaja de esta aplicación web es que se necesita programar para resolver los niveles. Esto se convierte en un impedimento para considerarlo como una herramienta que pudiera introducir a estudiantes en el pensamiento computacional.

5.2. Pilas Engine

Pilas Engine es una aplicación argentina para desarrollar animaciones y videojuegos en 2D. Su ventaja principal es que todos sus elementos están en castellano, por lo que se puede escribir código como el siguiente:

```
moneda = pilas.actores.Moneda()
```

Sin embargo, no es un pseudo lenguaje en castellano. La librería pilas está escrita en el lenguaje *Python*, por lo que la programación que uno tiene que realizar sigue siendo en inglés, e inevitablemente se tienen que escribir código como el siguiente:

```
import pilasengine

class AceitunaEnemiga(pilasengine.actores.Aceituna)
  def iniciar(self)
    self.imagen = "aceituna.png"
```

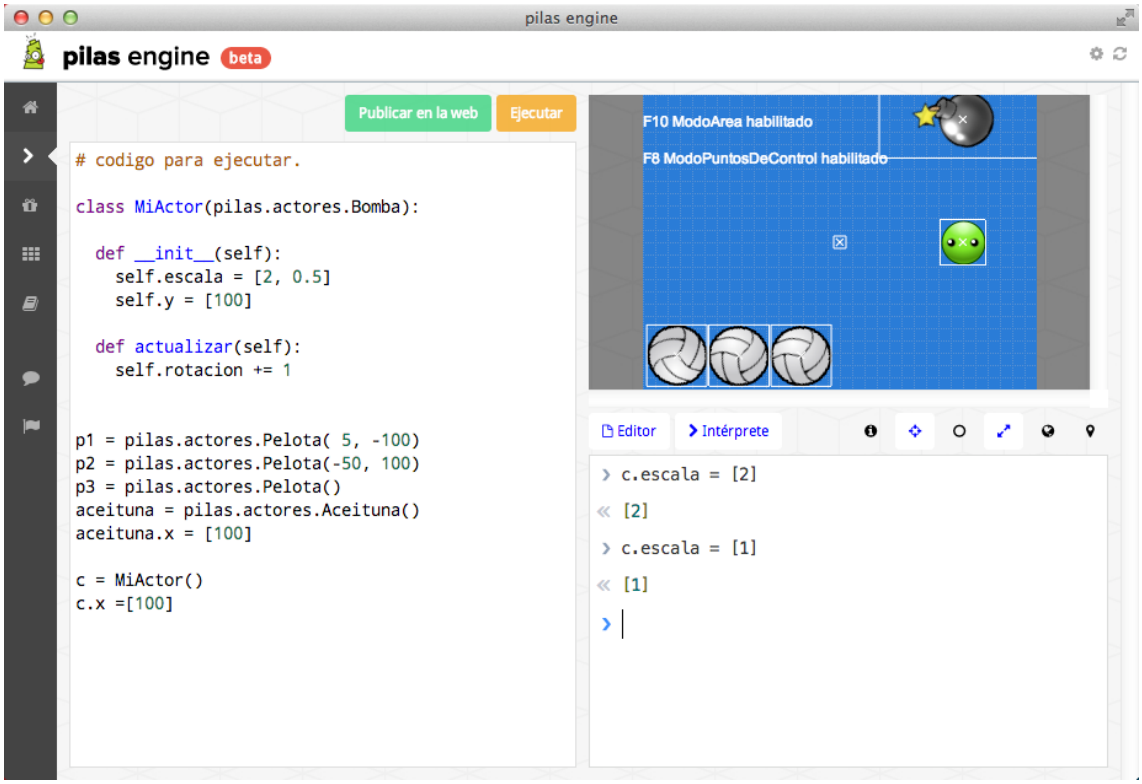

Figura 2. Captura de pantalla de la vista de desarrollo de una animación en Pilas Engine.

Este problema genera una barrera de acceso a su uso, ya que un estudiante que no está familiarizado con el inglés y tampoco con la programación, se sentirá desorientado y perderá rápidamente el interés. Y al igual que CodeCombat, el hecho de que se deba programar de forma explícita no la hace factible para ser utilizada como herramienta para la introducción al pensamiento computacional.

5.3. Alice

Por otro lado, existen aplicaciones más amigables para quienes recién entran en contacto con la programación, como lo es Alice, una plataforma para implementar juegos y animaciones en 3D. Trae un conjunto de librerías con modelos de objetos, personajes y escenarios; se puede manipular la luz y las cámaras y escribir un guion. Éste se escribe a través de bloques que imitan a los del rompecabezas, por lo que no es posible insertarlos de cualquier manera. De esta forma, al incrustar los bloques propuestos por la herramienta y sin darse cuenta, generan un algoritmo que se traduce en una animación.

La principal ventaja de Alice con respecto a CodeCombat es el uso de bloques como mecanismo para escribir algoritmos, el cual tiene un fundamento y ventaja pedagógica por sobre la utilización de sentencias escritas de un lenguaje de programación. Los bloques siempre generan programas sintácticamente correctos, lo que permite a los estudiantes enfocarse únicamente en comprender la semántica, es decir, el efecto que producen los bloques sobre los elementos. Además, al abstraer el uso de la sintaxis de un lenguaje, permite la traducción absoluta de las instrucciones a un idioma diferente al del lenguaje de programación.

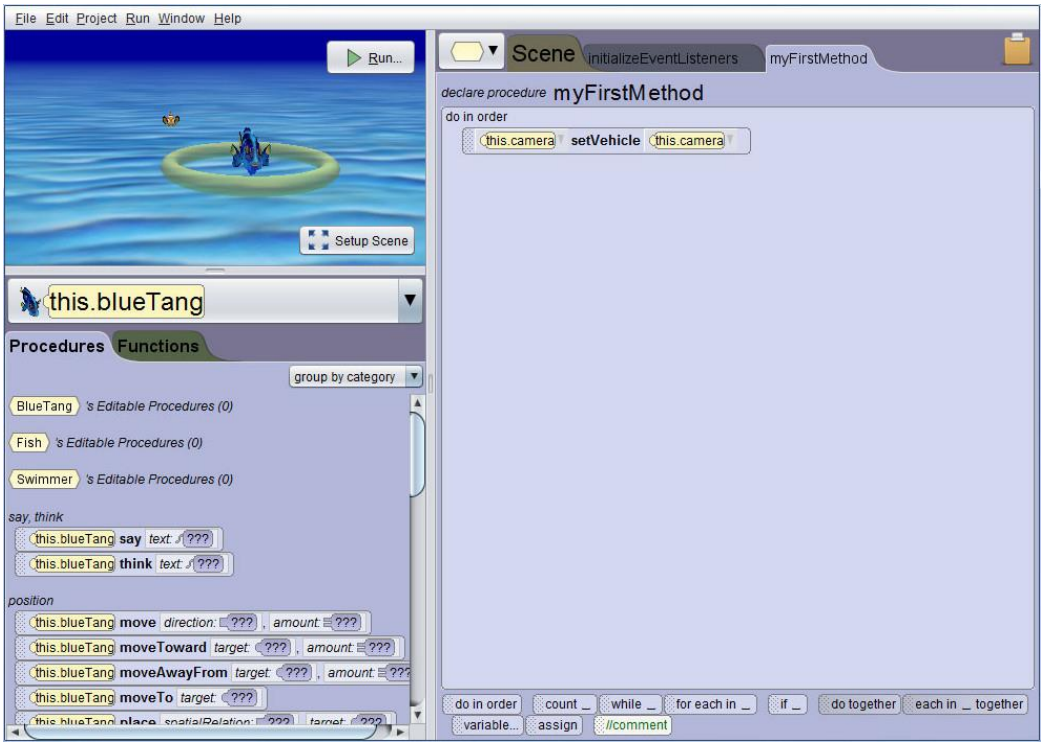

Figura 3. Captura de pantalla de la vista de desarrollo en Alice 3.3.

Sin embargo, al igual que todas las aplicaciones anteriores, Alice carece de un elemento primordial: no puede ser utilizada fuera de la clase de computación. Cuando se presentó el concepto el pensamiento computacional, se hizo referencia a la posibilidad de poder aplicarlo en distintos ámbitos, no solo en la clase de computación; en este caso, no hay forma de uso que permita adaptar actividades de otras materias.

5.4. Pilas Bloques

De la misma manera que Alice, Pilas Bloques utiliza los bloques como mecanismo para escribir algoritmos. Emplea Pilas Engine como motor gráfico, es web, y a diferencia de Alice, que brinda un escenario vacío en el cual llevar a cabo la animación o el juego, Pilas Bloques presenta un conjunto de desafíos a resolver.

Esto es lo más cercano a lo que se pretende desarrollar en este trabajo, ya que un desafío representa una actividad sencilla con un fin en particular, por ejemplo: enseñar iteración, abstracción, modularización, o condicionales. De todas maneras, carece de desafíos que adapten actividades que se den en otras materias.

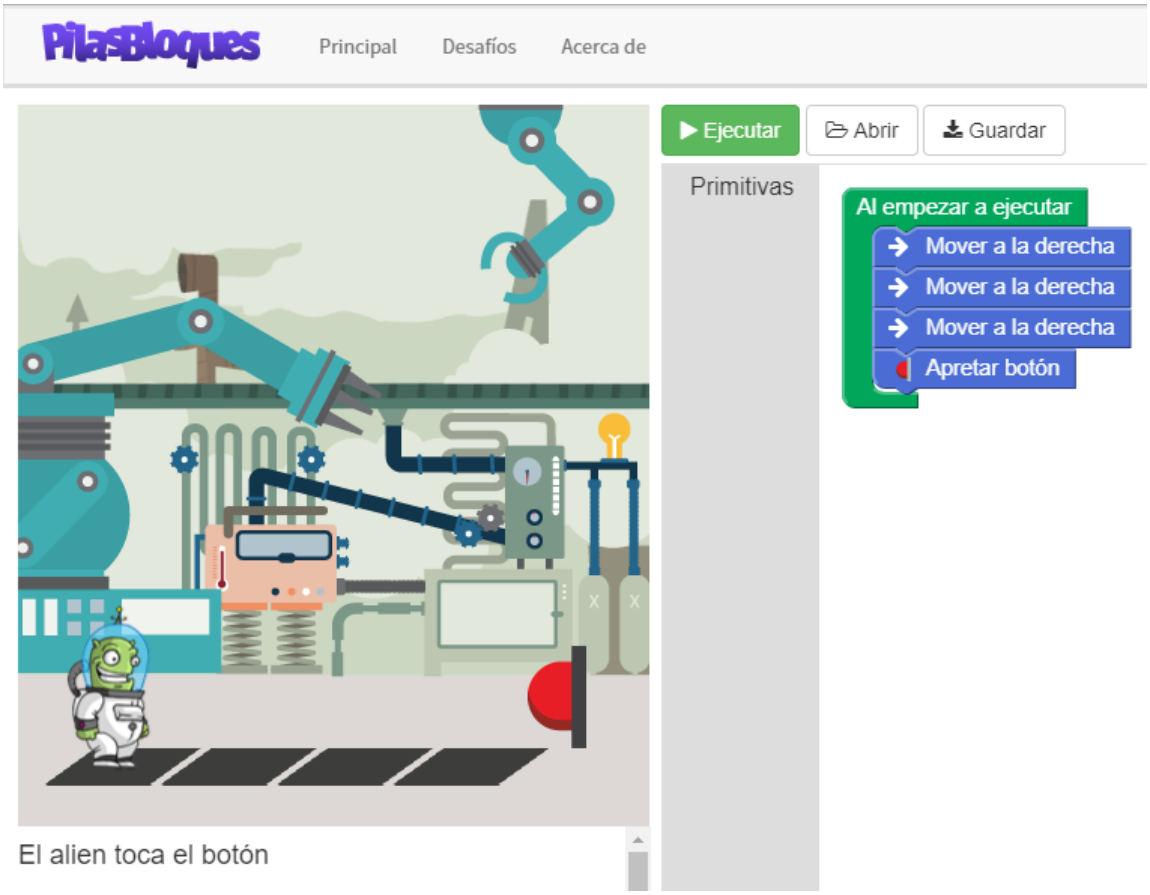

Figura 4. Captura de pantalla del desafío "El alien toca el botón" de Pilas Bloques.

5.5. Khan Academy

También existen aplicaciones web como Khan Academy que se preocupan por desarrollar actividades que cumplan con la currícula escolar. A través de éstas es posible aprender matemática, lengua, ciencias, computación, artes y humanidades. Lo importante de Khan Academy es que, como reza en su sitio, es "*gratuita, para todo el mundo y para siempre*", es decir, conocimiento abierto.

Khan Academy permite acceder como docente, padre o estudiante. Con ello se asegura que sea posible llevar a cabo un proceso de aprendizaje donde un adulto puede analizar el progreso del estudiante dentro y fuera del aula. Y, como las actividades que ofrece están orientadas a distintas asignaturas, consigue darles un uso práctico a los dispositivos digitales, dentro del aula para la mayoría de los docentes.

5.6. Códimo

Con Códimo se pretende tomar conceptos de estos programas, como los bloques de Alice, la experiencia de usuario que brinda Khan Academy y lo entretenido de CodeCombat para desarrollar una aplicación que se diferencie del resto en un punto en particular: **los objetos de aprendizaje que se desarrollen presentarán actividades que cumplan con los conceptos de una materia y que deberán ser resueltos aplicando una habilidad o un conjunto pequeño de habilidades en particular del pensamiento computacional**. De esta forma se podría romper la estructura común de un tipo de actividad, lo que forzaría al estudiante a pensar de una manera diferente la solución de una actividad que ya conoce. En este sentido, esta aplicación no implica programar, sino el uso del pensamiento computacional para la resolución de problemas.

Capítulo 1: Pensamiento computacional

La computación hizo posible profundos saltos de innovación e imaginación gracias a que facilitó los esfuerzos para resolver problemas y expandir el conocimiento del ser humano como sistema biológico y la relación que existe entre el humano y el mundo (Carneiro, 2009; Piscitelli, 2009; Dussel, 2010). Estos avances, a su vez, exigen individuos competentes que puedan llevar la capacidad de resolución de problemas a través de la computación a un espacio de posibilidades que continúa creciendo.

La ubicuidad que le brindó el desarrollo de Internet y el papel que desempeña en la innovación, la computación se convirtió en una herramienta fundamental para el desarrollo de una persona inmersa en la sociedad del conocimiento. Así, surge el interrogante acerca de qué enseñar y cómo integrar la computación en la educación inicial.

A partir de la década del ochenta del siglo pasado, empezó a plantearse el 'pensamiento computacional' como el conjunto de habilidades que permitirían mejorar los procesos de solución de problemas y el pensamiento crítico de las personas, aprovechando todo el potencial que ofrece la computación; ya que está relacionado fuertemente con el poder y la limitación de los procesos computacionales, sean ejecutados por una computadora o una persona. Los métodos y los modelos de estos procesos permiten resolver problemas y diseñar sistemas que nadie podría resolver a solas. Wing (2006), en su trabajo "Computational Thinking", plantea que el pensamiento computacional debería ser "*una habilidad fundamental para cualquiera, no sólo para los científicos de la computación*" (p. 33, traducción propia). Es decir que debería considerarse como una habilidad fundamental al mismo nivel que la lectura, la escritura y la matemática, y que debería incluirse en la currícula escolar.

El significado del concepto 'pensamiento computacional' es muy amplio y continúa evolucionando al ritmo de los avances tecnológicos. Seymour Papert es uno de los primeros teóricos en utilizar el término 'pensamiento computacional' aplicado a la educación. Este autor hizo que sus estudiantes trabajen con el lenguaje LOGO para resolver actividades que él les planteaba (Tedre y Denning, 2016). Si bien esta concepción data de la década del ochenta, a partir de los noventa y con el desarrollo de sistemas operativos visuales e intuitivos, así como con la proliferación de aplicaciones informáticas "empaquetadas", las actividades planteadas por

Papert para la clase de computación fueron dejándose de lado en favor de la enseñanza de aplicaciones vinculadas principalmente con las tareas de oficina.

Wing (2006) revive el concepto de 'pensamiento computacional' y se refiere a él como desarrollo cognitivo de las personas, y aporta ejemplos prácticos donde este tipo de pensamiento se aplica en diferentes áreas:

> El pensamiento computacional implica resolver problemas, diseñar sistemas y entender el comportamiento humano en función de los conceptos fundamentales de la Ciencia de la Computación; es reformular un problema que pareciera difícil en uno que se sabe cómo resolver, tal vez mediante reducción, transformación o simulación.
>
> El pensamiento computacional es pensamiento recursivo y es procesamiento paralelo; es interpretar código como datos y datos como código. También significa juzgar un programa no sólo por su correctitud y eficiencia sino también por su estética y su diseño de software por su simplicidad y elegancia.
>
> El pensamiento computacional es el uso de la abstracción y la descomposición para resolver tareas o para diseñar sistemas complejos. Es separación de conceptos, como también es seleccionar una representación apropiada de un problema o modelar los aspectos relevantes de un problema para hacerlo más manejable.
>
> El pensamiento computacional es modularizar algo en anticipación de múltiples usuarios, o *prefetching & caching* con anticipación. Es pensar en términos de prevención, protección, y recuperación del peor escenario a través de redundancia, contención del daño y corrección de errores.
>
> El pensamiento computacional es planificar y aprender en presencia de incertidumbre.

Wing profundiza el concepto al describir sus aspectos cognitivos:

- "*Conceptualización, no programación*". Pensar como un científico de la computación significa mucho más que ser capaz de programar. Requiere pensar en muchos niveles de abstracción.
- "*Habilidad fundamental, no de memorización*". Una habilidad fundamental es algo que cada ser humano debe saber para poder funcionar en sociedad. De memorización significa una rutina mecánica.
- "*Una manera en que los humanos, no las computadoras, piensan*". El pensamiento computacional es una forma en que los humanos resuelven problemas; no es intentar que piensen como una computadora.
- "*Complementa y combina el pensamiento matemático y el ingenieril*". El pensamiento computacional esencialmente se basa en el pensamiento matemático, al igual que el resto de las ciencias. También esencialmente se basa en el pensamiento ingenieril, dado que se construyen sistemas que interactúan con el mundo real. Las restricciones de los subyacentes dispositivos computacionales fuerzan a los científicos de la computación a

pensar computacionalmente, no solo matemáticamente. La libertad para construir mundos virtuales permite aplicar la ingeniería en sistemas más allá del mundo físico.

- "*Ideas, no artefactos*". Los artefactos de software y hardware no es lo único que se produce y que están físicamente presentes en todos lados y afectan la vida diaria, también están los conceptos computacionales que se usen para resolver problemas, administrar las tareas diarias y comunicarse e interactuar con otras personas.
- "*Para todos y en cualquier lugar*". El pensamiento computacional será una realidad cuando sea tan integral a las actividades humanas que pasará a ser una filosofía implícita.

Esta reintroducción del concepto no pudo ser más certera. La autora, desde el cognitivismo, refuerza el uso de la Ciencia de la Computación y valora el pensamiento computacional aplicado al desarrollo cognitivo de los seres humanos. En este sentido, Tedre y Denning sostienen que la investigación en este campo se ha madurado y emerge a un entendimiento multidisciplinario de la pedagogía de la programación. Así, las políticas educativas se orientan en la educación STEM (*Science, Technology, Engineering, Mathematics*; Ciencia, Tecnología, Ingeniería, Matemáticas) y están dispuestas a incluir la Ciencia de la Computación en la definición de STEM.

Por otra parte, Wing (2006) da ejemplos prácticos y valorables en el que la computación es aplicada en otras áreas:

> *Machine Learning* transformó la estadística. El aprendizaje estadístico está siendo utilizado para resolver problemas en una escala, en términos de tamaño de datos y dimensiones, inimaginables hace algunos años atrás.
>
> Las contribuciones de la Ciencia de la Computación en Biología van más allá de la habilidad de buscar en una vasta cantidad de datos con el fin de identificar patrones. El gran salto se dio porque las estructuras de datos y los algoritmos —las abstracciones de la Ciencia de la Computación— pueden representar las estructuras de proteínas de una manera que permiten elucidar su función. El pensamiento computacional está cambiando la manera en que los biólogos piensan.
>
> Similarmente, la *nano-computación* está cambiando la manera de pensar de los químicos, como la Computación Cuántica la forma de pensar de los físicos.

Wing también presenta algunos ejemplos cotidianos para darle un énfasis en cómo esas situaciones están rodeadas de habilidades que se aplican en la Ciencia de la Computación:

> Cuando un estudiante pone en su mochila las cosas que necesita para el día está aplicando *prefetching & caching*.
>
> Cuando se agregan etiquetas con nombres a un cuaderno para poder acceder más rápido a una página por un contenido en particular, se está *indexando*.
>
> Cuando alguien pierde las llaves y trata de recordar dónde las dejó al tratar de rehacer paso por paso qué hizo hasta ese momento, está aplicando *backtracking*.

> ¿En qué línea de caja se pararía en un supermercado? Eso es *performance modeling* para sistemas multi servidor.
>
> ¿Por qué el teléfono fijo sigue funcionando durante un corte de luz? Eso es *independencia de fallo y redundancia a nivel de diseño*.

"*Esta manera de pensar será parte del conjunto de habilidades de todas las personas. La computación ubicua es en la actualidad lo que el pensamiento computacional es en el futuro. La computación ubicua fue el sueño de ayer que se convirtió en la realidad de hoy; el pensamiento computacional es la realidad del mañana*" (Wing, 2006, traducción propia).

Estos ejemplos dejan en claro que muchas de las habilidades que la Ciencia de la Computación explota en profundidad son, en mayor o menor medida, utilizados por todos para realizar actividades cotidianas en todos los ámbitos de la vida. Lo poderoso de utilizar el pensamiento computacional para describir cada una de estas habilidades es la posibilidad de darles un nombre, es decir, identificarlas, describirlas, analizarlas y perfeccionarlas.

> "*Ya no es más suficiente tener que esperar hasta que los estudiantes estén en la universidad para introducir estos conceptos. Todos los estudiantes de la actualidad tendrán una vida fuertemente influenciada por la computación*" (Barr, 2011, traducción propia).

Sin embargo, incluir el pensamiento computacional en el aula no es una tarea sencilla. Según Barr y Stephenson (2011), para introducir con éxito los conceptos del pensamiento computacional en la currícula de K-12 (desde el jardín de infantes a los doce años en el Sistema de Educación de Estados Unidos de América) se requiere de un esfuerzo en dos direcciones: (1) debe haber cambios en la política educativa para poder superar problemas infraestructurales importantes; y (2) los docentes K-12 necesitan recursos, empezando por una definición del pensamiento computacional convincente y con ejemplos relevantes y apropiados para cada grado.

En este sentido, Códimo propondría soluciones focalizadas en la segunda parte de ese esfuerzo necesario, teniendo en cuenta que esta herramienta ofrecería un compendio de actividades desarrolladas según la visión del docente, en relación con cómo debería implementarse una actividad en particular a través de un ejercicio digital. Es decir, Códimo sería una herramienta que daría soporte al docente en el dictado de su materia, y no una plataforma que fuerce la estructura didáctica en una dirección determinada.

Con este enfoque abierto, Códimo no obstruiría la planificación del docente, sino que se adaptaría mediante la presentación de módulos didácticos autocontenidos para actividades específicas de una asignatura. Se hace referencia a un módulo didáctico autocontenido a una actividad que generalmente contiene un aspecto teórico, otro práctico y uno evaluativo. En el capítulo 2 se abordará este aspecto con mayor profundidad.

Para dar énfasis en el deseo de integrar el pensamiento computacional a la escuela, la caja de herramientas —escrita por la Sociedad Internacional para las Tecnologías en Educación (ISTE por su sigla en inglés) y la Asociación de Maestros de Ciencia de la Computación (CSTA por su sigla en

inglés) en el 2011— presenta una meta: "*debe prepararse a los jóvenes estudiantes para que se conviertan en pensadores computacionales que comprendan cómo las herramientas digitales de hoy pueden ayudar a resolver los problemas del mañana*". Y para ello, la tarea —de cada actor dentro del sistema educativo— debe ser la de encontrar mejores maneras de imaginar el potencial del pensamiento computacional de manera transversal a todas las disciplinas y encontrar formas de facilitar el aprendizaje dentro y a través de éstas, de manera que se pueda desarrollar en los estudiantes las habilidades que necesitan para resolver los retos presentes y futuros.

La definición de pensamiento computacional para los autores de esta caja de herramientas es la de un "*enfoque para resolver un determinado problema que empodera la integración de tecnologías digitales con ideas humanas. No reemplaza el énfasis en creatividad, razonamiento o pensamiento crítico, pero refuerza esas habilidades al tiempo que realza formas de organizar el problema de manera que la computadora pueda ayudar*".

Por último, es importante tener en cuenta que enseñar el pensamiento computacional no significa que los estudiantes deban trabajar en el campo de la Ciencia de la Computación, pero si lo hiciesen, estarían mucho mejor preparados.

1. El pensamiento computacional para K-12

Según Barr y Stephenson (2011), la actualidad de la educación K-12 es muy compleja: posee un entorno altamente politizado donde muchas prioridades, ideologías, pedagogías y ontologías compiten por la dominancia. Cuando los autores hablan de un "entorno complejo y altamente politizado" se refieren a que la educación varía según el docente, el colegio, el ministerio o la secretaría de educación de un estado y el país en el que está inserto. A su vez afirman que el contexto educativo está simultáneamente sujeto a diversas expectativas, intenso escrutinio y escasos recursos. Por eso "cualquier esfuerzo para alcanzar un cambio sistemático en este ámbito requiere de un entendimiento profundo de las realidades de este sistema" (Barr, 2011, traducción propia).

Como consecuencia, los autores afirman que incluir el pensamiento computacional en K-12 requiere de un enfoque práctico basado en una definición operacional, la cual inicialmente debe responder una serie de preguntas enfocadas específicamente para K-12:

1. ¿Qué forma tendrá el pensamiento computacional en el aula?
2. ¿Qué habilidades podrán desarrollar los estudiantes?
3. ¿Qué necesitará el docente para poder poner el pensamiento computacional en práctica?
4. ¿Qué podrá modificarse y extenderse de lo que están haciendo actualmente los docentes?

Por último, y para que la definición sea de utilidad, debe estar acompañada de ejemplos que demuestren cómo puede ser incorporado al aula.

Barr y Stephenson (2011) sintetizan un trabajo realizado por la Asociación de Profesores de la Ciencia de la Computación (CSTA) y la Sociedad Internacional por la Tecnología en la Educación (ISTE) en el que llevaron a cabo un proyecto con el fin de desarrollar una definición del pensamiento computacional para K-12. Para dicho proyecto se seleccionaron 26 líderes de pensamiento y se les encomendó desarrollar una visión compartida y un conjunto de estrategias para integrar el pensamiento computacional a través de todo el currículum de K-12, específicamente en áreas como matemática, ciencia, ingeniería y tecnología. Una primera observación planteada por este grupo es que el cambio educacional necesario es "considerablemente más complejo de lo que suponían" y que trabajar con docentes de distintas disciplinas significa "desconectar al pensamiento computacional de la Ciencia de la Computación".

Los participantes del proyecto mencionado llegaron al consenso de que "*el pensamiento computacional es un enfoque para solucionar problemas de una manera que puede implementarse con una computadora. Los estudiantes dejan de ser meros consumidores de herramientas y pasan a ser constructores de herramientas. Utilizan un conjunto de conceptos, como la abstracción, la recursión e iteración para procesar y analizar datos y para crear artefactos reales y virtuales. El pensamiento computacional es una metodología para solucionar problemas que puede ser automatizada, transferida y aplicada en las distintas materias*" (Barr y Stephenson, 2011, traducción propia).

Con el fin de que esta cultura pueda establecerse dentro del aula, los participantes del proyecto mencionado por Barr y Stephenson establecieron estrategias o características benéficas para cualquier experiencia de aprendizaje que utilice pensamiento computacional:

- Aceptación, por parte del docente y los estudiantes, de los intentos fallidos, y reconocer que un error temprano puede llevar fácilmente al camino correcto.
- Trabajo en equipo con uso explícito de:
    - *Descomposición*: dividir un problema en pequeñas partes que pueden ser resueltas con mayor facilidad.
    - *Abstracción*: simplificar un problema específico a uno más general a partir de identificar los conceptos claves.
    - *Negociación*: grupos internos de un equipo trabajando conjuntamente para unir partes de una solución.
    - *Construcción de consenso*: trabajar para construir solidaridad grupal sobre una idea o solución.

En la siguiente tabla, Barr y Stephenson muestran algunos ejemplos de habilidades del pensamiento computacional en diversas asignaturas:

Tabla 1.

Conceptos y capacidades centrales del pensamiento computacional

Concepto del pensamiento computacional	Ciencia de la Computación	Matemáticas	Ciencia	Sociales	Lengua y arte
Colección de datos	Encontrar una fuente de datos para un área de problemas.	Encontrar una fuente de datos para un problema, por ejemplo, sobre lanzamientos de monedas al aire, o de un dado.	Recolectar datos de un experimento.	Estudiar estadísticas de batallas o poblacionales.	Análisis lingüístico de oraciones.
Análisis de datos	Escribir un programa para hacer cálculos estadísticos básicos sobre un conjunto de datos.	Contar ocurrencias de lanzamientos de la moneda, o de un dado, y analizar los resultados.	Analizar los datos de un experimento.	Identificar tendencias a partir de las estadísticas.	Identificar patrones para diferentes tipos de sentencias.
Representación de datos	Utilizar estructuras de datos como el arreglo, listas enlazadas, pilas, colas, grafos, tablas hash, etc.	Utilizar histogramas, gráfico de torta o de barras para representar los datos; utilizar sets, listas, grafos, etc. para contener la información.	Sumarizar los datos de un experimento.	Sumarizar y representar tendencias.	Representar patrones de diferentes tipos de sentencias.
Descomposi-ción de problemas	Definir objetos y métodos; definir funciones.	Aplicar orden de operandos en una expresión.	Clasificar especies.		Escribir un resumen.
Abstracción	Utilizar procedimientos o funciones para encapsular un conjunto de comandos repetidos; utilizar condicionales, bucles, recursión, etc.	Utilizar variables en álgebra; identificar los elementos esenciales de un problema escrito; estudiar funciones de álgebra y compararlas con las de computación; utilizar la iteración para resolver problemas escritos.	Construir un modelo de una entidad física.	Sumarizar hechos; deducir conclusiones a partir de los hechos.	Utilizar la analogía y la metáfora; escribir historias
Algoritmos y procedimientos	Estudio de algoritmos clásicos; implementar un algoritmo para un área de problemas.	Trabajar con grandes divisiones aplicando factorización; acarreo en sumas o restas.	Hacer un procedimien-to experi-mental.		Escribir ins-trucciones.

Automatización		Utilizar herramientas como: GeoGebra, StarLogo; snippets en Python.	Utilizar *probeware*[11].	Utilizar Excel.	Utilizar un auto corrector.
Paralelización	*Threading*, *pipelining*, división de datos o tareas de tal manera que puedan ser procesadas en paralelo.	Solución de sistemas lineares; multiplicación de matrices.	Ejecución simultánea de varios experimentos.		
Simulación	Animación de algoritmos; *parameter sweeping*.	Graficar una función en el plano cartesiano y modificar los valores de las variables.	Simular el sistema solar en movimiento.	Jugar a Age of Empires (modo historia).	Hacer una representación de una historia.

Recuperado de Barr y Stephenson (2011).

A su vez, identificaron como gran factor positivo el hecho de que "*el pensamiento computacional es aplicable para cualquier otro tipo de razonamiento. Hace posible realizar todo tipo de cosas: física cuántica, biología avanzada, sistemas humano-computadora, o el desarrollo de herramientas computacionales*" (Barr y Stephenson, 2011, traducción propia).

Por último, observaron que los estudiantes (1) se involucran en el uso de herramientas para resolver problemas, (2) se sienten cómodos con el *ensayo y error*, y (3) trabajan con más entusiasmo en un entorno de trabajo en conjunto.

En la experiencia llevada a cabo con las escuelas primarias N.º 296 y N.º 355 de Viedma, Río Negro, durante la Semana de la Ciencia y la Tecnología[12], se pudo constatar los tres ítems señalados en el párrafo anterior. El grupo de la mañana estuvo formado por dos terceros grados de la escuela N.º 296, y por la limitación de computadoras se trabajó en parejas. Esto permitió que entre ambos encuentren la solución de cada ejercicio, intercambien comentarios acerca de cómo resolverlos, compartan las frustraciones, y eviten la resignación fácil ante el error. Durante la tarde participó el único cuarto grado de la escuela N.º 355, por lo que cada uno estuvo solo en una computadora. Esto causó que los coordinadores del proyecto tuvieran que asistir con mayor énfasis a cada estudiante por separado, ya que las frustraciones no se compartieron con el compañero, y por ende se resignaron con mayor facilidad. El hecho de no poder plantear el problema en voz alta con alguien más fue otro motivo que generó cierta dificultad al momento de encontrar una solución.

[11] Probeware: es el término que describe el uso de sensores y software (que puede ser utilizado en microprocesadores como el que poseen las calculadoras) para realizar mediciones científicas (Park, J. C., 2008, traducción propia).

[12] Esta experiencia se llevó a cabo en la sede Atlántica de la Universidad Nacional de Río Negro, el día 5 de septiembre del 2017 en el marco del proyecto de extensión "Imaginación y Motivación. Puntos de partida para la enseñanza de la Programación en las escuelas". Dirigido por Mg. Enrique Molinari.

En líneas generales, esta experiencia reafirma que los estudiantes se involucraron en el uso de las herramientas para resolver problemas, se sintieron cómodos con el ensayo y error, y se demostró que el trabajo en equipo es más favorecedor que trabajar solo.

2. Un lenguaje para el pensamiento computacional

Lu y Fletcher (2009) afirman que para que sea posible que se expanda la participación en Ciencia de la Computación, primero deben sentarse las bases del pensamiento computacional antes de que los estudiantes experimenten su primer lenguaje de programación. En su trabajo *Thinking About Computational Thinking*, plantean la hipótesis de que la programación es a la Ciencia de la Computación lo que la demostración por construcción es a la Matemática, y lo que el análisis literario a Lengua. Es debido a esto que, por analogía, la programación debe formar parte introductoria de los cursos avanzados de la Ciencia de la Computación, en vez de ser lo primero que se les presente los estudiantes.

Según estos autores, la enseñanza del pensamiento computacional debería enfocarse en establecer el vocabulario y la simbología que permitan ser usados para anotar y describir computación y abstracción; sugerir información, y proveer una notación para que puedan construir modelos mentales de procesos. Para ello, plantean que la forma de representar el pensamiento computacional en el aula debe estar basada en un mecanismo que actúe como "pegamento" entre los conceptos del resto de las materias y la terminología de la Ciencia de la Computación. Para ello proponen un lenguaje común —un Lenguaje del pensamiento computacional (CTL, por su nombre en inglés)—, formado por un vocabulario y símbolos que puedan ser utilizados para anotar y describir el pensamiento computacional.

2.1. Uso del Vocabulario del CTL en Matemática y Lengua

2.1.1. Ejemplo 1: Introducción a la multiplicación

Durante la introducción de la multiplicación es común el uso de frases como "la multiplicación es igual a repetir la suma del número de la izquierda una cantidad igual al del número de la derecha" y "el resultado de una multiplicación siempre es el mismo, sin importar qué número se escribe primero".

El uso de la repetición de la suma como definición es una oportunidad para introducir dos conceptos computacionales: la *iteración* y la *eficiencia*; a partir de explicar que cada signo + representa una iteración, y que mientras la operación es conmutativa, la eficiencia de las dos expresiones podría variar. Algunos ejercicios podrían ser:

1. Escribe cada multiplicación como una repetición de sumas y luego obtiene el resultado.
2. Escribe la multiplicación intercambiando los valores y compara el número de iteraciones necesarias. ¿Cuál es más eficiente?

Tabla 2.
Ejemplo:

Expresión original	Números intercambiados	¿Cuál es más eficiente?
6 x 3	3 x 6	6 x 3 necesita 3 iteraciones 3 x 6 necesita 6 iteraciones Así que 6 x 3 es más eficiente

Recuperado de Lu y Fletcher (2009, p. 262).

2.1.2. Ejemplo 2: Comprensión de lectura

Es común que existan actividades donde deben ordenarse un conjunto de sentencias de forma cronológica. Considere el siguiente ejemplo:

1. No quiero más pizza por un tiempo.
2. Comí diez porciones de pizza.
3. Más tarde en la noche, me descompuse.
4. Me sentí muy lleno.

¿Cuál de las siguientes ordenaciones es correcta?

a) 1, 3, 4, 2
b) 4, 3, 2, 1
c) 2, 3, 1, 4
d) 3, 1, 4, 2
e) 2, 4, 3, 1

En este ejercicio podría explicarse a los estudiantes que cada ordenamiento es un estado, y las cinco posibilidades, de la a) a la e), forman el *espacio de búsqueda* del problema.

Para resolverlo, podría verificarse cada estado de forma individual, pero también podría utilizarse "*dividir y conquistar*" para eliminar las respuestas incorrectas. Las siguientes preguntas son potenciales tareas:

1. ¿Cuál es el orden correcto entre el paso 2 y 3?
2. ¿Cuáles estados del espacio de búsqueda tienen al paso 2 y al 3 mal ordenados? ¿Algunas de estas respuestas podrían ser correctas?
3. ¿Qué otros posibles estados existen y no están listado?

2.2. Uso de la Notación para CTL en Matemática y Lengua

A medida que los estudiantes se enfrentan a actividades cada vez más complejas, debería introducirse las notaciones del CTL. En particular, la notación básica de tuplas para estructurar

datos y representar estados, junto a un simple sistema de reescritura para anotar los cambios de estados pueden asistir en clarificar los aspectos computacionales de muchos problemas.

2.2.1. Ejemplo 3: Encontrar la raíz cuadrada

Una manera común de encontrar la raíz cuadrada de un número sin usar la calculadora es a partir de repetir el método "*estimar-dividir-promediar*" (EDA, por su significado en inglés *estimate-divide-average*). El proceso para obtener cada estimación está dada al dividir n (el número inicial al cual se le quiere encontrar la raíz) por la estimación actual *g*, y promediar ese resultado con g. Al denotar el cálculo anterior con el símbolo ⇒, es posible demostrar el proceso computacional para encontrar una raíz cuadrada. Considere el siguiente ejemplo:

1. Encuentre la aproximación más certera de la raíz de 60 con un máximo de 3 dígitos decimales.

$$2 \quad \Rightarrow \quad 16 \quad \Rightarrow \quad 9.875$$
$$\Rightarrow \quad 7.995 \quad \Rightarrow \quad 7.749$$
$$\Rightarrow \quad 7.746$$

Con este problema es posible explicar que ⇒ es una abstracción de la función:

$$f(g) = \frac{\left(\frac{60}{g} + g\right)}{2}$$

Otra alternativa es describir la solución como una *búsqueda binaria* en el cual se define un rango que es acotado sucesivamente hasta alcanzar un término medio suficientemente cerca de la respuesta. Esta computación comparativa brinda la oportunidad para discutir *representación y decisión*. Para EDA, un único número captura por completo un estado ya que el siguiente estado es resultado de aplicar una función unaria al estado actual. En cambio, en la búsqueda binaria, es necesaria una representación de un par de números, debido a que el siguiente estado está determinado por los dos límites del rango actual.

2.2.2. Ejemplo 4: Análisis sintáctico de oraciones

Es común que cuando se quiere enseñar el análisis sintáctico y el diagrama de oraciones se utilice el sistema Reed-Kellogg o el diagrama de árbol. A su vez, es frecuente que los primeros ejercicios pidan identificar el sintagma nominal y el sintagma verbal de una oración. Luego, cada sub-oración es subsecuentemente identificada con otras categorías lingüísticas.

Independientemente del sistema de representación, este tipo de ejercicios es útil para familiarizar a los estudiantes con diferentes notaciones. Más aún, la representación de la "gramática libre de contexto" y el "proceso de derivación" —en relación con el proceso de

desfragmentación de una oración a través del árbol de gramática generativa—, son buenas oportunidades para enfatizar el poder de la *recursión* y el *no-determinismo*.

Ejemplo:

oración	→	sintagma-nominal + sintagma-verbal
sintagma-verbal	→	verbo + sintagma-nominal
sintagma-nominal	→	artículo | sustantivo | nulo
verbo	→	{ cadena de caracteres }
artículo	→	{ colección de posibles artículos }
sustantivo	→	{ cadena de caracteres }
nulo	→	{ vacío }

La tercera regla muestra dos posibles reescrituras del sintagma nominal. A su vez, junto con la segunda regla, muestra recursión. El proceso de diagramación puede ser descrito a través de la forma arbórea o —para mostrar los procesos computacionales no-determinísticos— como una derivación a partir de aplicar las reglas gramaticales de manera recursiva:

(oración)	→	(sintagma-nominal, sintagma-verbal)
	→	((artículo, sustantivo), sintagma-verbal)
	→	((el, verano), terminó)

3. El pensamiento computacional para Códimo

Códimo concibe el pensamiento computacional desde un punto de vista práctico para que pueda ser aplicado sin demasiado esfuerzo por parte de los estudiantes de nivel primario de la escuela argentina. En este trabajo, se entiende el pensamiento computacional como un conjunto de herramientas que permiten mejorar el aprendizaje de conceptos de otras asignaturas a través de la computación, de forma que la computadora pueda ser integrada al aula de manera inmediata.

Cada herramienta del pensamiento computacional puede ser aplicada a un conjunto de actividades de diferentes materias, por ejemplo, el desarrollo de algoritmos puede ser utilizado en Matemática y Lengua. De esta forma, Códimo divide su contenido en actividades donde cada una de ellas aplica una de estas herramientas para un concepto de una materia.

Capítulo 2: Objetos de aprendizaje

1. ¿Qué es un 'objeto de aprendizaje'?

Según David Wiley (2003) un 'objeto de aprendizaje' es la construcción más pequeña de un contenido educativo (relativa al tamaño del curso) que puede ser elaborada por *diseñadores instruccionales* —responsables de elaborar experiencias educativas aplicando una teoría de aprendizaje—, y que puede ser reutilizada en diferentes contextos de aprendizaje. A su vez, Wiley presenta y analiza una serie de definiciones sobre objetos de aprendizaje y llega a la conclusión de que no existe una definición abarcadora de lo que significa y que varía según el autor que lo defina.

Entre las distintas definiciones, menciona una que se adecua muy bien a las necesidades de Códimo: un objeto de aprendizaje "se define como la experiencia instruccional más pequeña e independiente que contiene un objetivo, una actividad de aprendizaje y una evaluación" (L'Allier, 1997, traducción propia).

- **Objetivo**: es el elemento que define los criterios que se deben cumplir para completar la actividad de aprendizaje.
- **Actividad de aprendizaje**: es el elemento que cumple la función de enseñar.
- **Evaluación**: es el elemento que determina si el objetivo o el proceso que realizó el estudiante se cumplieron.

A esta definición deben agregarse ciertos requerimientos funcionales que fueron definidos por la Universidad de Wisconsin (Universidad de Wisconsin, 2017), en su biblioteca digital de objetos de aprendizaje:

- Deben ser pequeñas partes de aprendizaje autocontenidas.
- Deben ser lo suficientemente pequeños para poder ser embebidos en actividades, lecciones, unidades o cursos.
- Deben ser flexibles, portables y adaptables, y capaces de poder ser utilizados en múltiples entornos de enseñanza y a través de diversas disciplinas.

El uso de objetos de aprendizaje tiene diversos beneficios para el docente. Reigeluth y Nelson (citado por Wiley, 2003) explican que es común que los docentes dividan el material didáctico en pequeñas partes constitutivas y que luego las reorganicen de manera que les permitan alcanzar los objetivos educativos definidos para un curso en particular. Por ello, el hecho de que los objetos de aprendizaje ya signifiquen la unidad didáctica más pequeña, este paso inicial estaría solucionado, lo que en teoría les permitiría a los docentes aumentar la velocidad y eficiencia con la cual diseñan y desarrollan el plan educativo.

Por otra parte, como los objetos de aprendizaje son módulos independientes e íntegramente digitales, facilitan la inclusión de este tipo de tecnología en el aula. Esta facilidad está dada primero porque el docente no tiene que adaptar por completo su plan para incluir actividades digitales, es decir, puede ser un proceso incremental donde vaya integrándolo de a poco; y segundo porque no requieren ser ejecutados exclusivamente sobre una computadora, sino que es posible que funcionen sobre una netbook, una tablet o desde el celular, lo que permite darles una utilidad educativa a estos dispositivos.

Desde el punto de vista del proceso de desarrollo, el hecho de que un objeto de aprendizaje sea la mínima unidad educativa permite que:

1. Puedan ser desarrollados de manera distribuida.
2. Puedan ser probados de forma aislada.
3. Sea posible definir una interfaz que facilite la comunicación entre ellos.
4. Sea posible construir una biblioteca de objetos de aprendizaje donde cualquiera pueda subir el desarrollo de cada uno.
5. Sea más explícita la metodología de evaluación y permita generar datos sobre los estudiantes.

2. Marco teórico para la construcción de objetos de aprendizaje

Ahora que se definió concretamente lo que es un objeto de aprendizaje, se puede definir el marco teórico que se utilizará para desarrollarlos, entre los cuales se encuentran: IMA-CID (Enfoque de Modelado Integral — Conceptual, Instruccional y Didáctico, Barbosa, 2006, traducción propia) es un marco que define un proceso de tres etapas donde cada una de ellas tiene como producto final un modelo; LODM (Método para el Desarrollo de Objetos de Aprendizaje, Graciotto et al., 2011, traducción propia) es una adaptación más práctica de IMA-CID que integra el concepto de transformaciones y considera el proceso como uno iterativo incremental; LODAS (Teoría de Diseño de Objetos de Aprendizaje y Secuenciación, Wiley, 2000, traducción propia) es una teoría de diseño de objetos de aprendizaje y cómo deben secuenciarse los contenidos, además define un marco para su uso de objetos de aprendizaje dentro del aula, cómo deben trabajarlos los docentes y qué expectativas tienen del estudiante; FLOM (Modelo Formal de Objetos de Aprendizaje, Sánchez et al., 2015, traducción propia), establece cuáles son los roles de cada actor

activo en el proceso de construcción de objetos de aprendizaje, a su vez que establece un proceso de desarrollo.

Luego de definir cada uno de ellos con un grado de profundidad suficiente para comprender cómo funcionan, a continuación, se los analizará, identificando los aspectos más relevantes y justificando el o los motivos por los cuales se decidió utilizar FLOM para la construcción de objetos de aprendizaje en Códimo.

2.1. Enfoque de Modelado Integral — Conceptual, Instruccional y Didáctico (IMA-CID)

Los módulos educacionales consisten en unidades de estudio concisas entregadas a los estudiantes a través del uso de tecnologías y recursos computacionales (Barbosa, 2006, traducción propia).

2.1.1. Módulo educacional

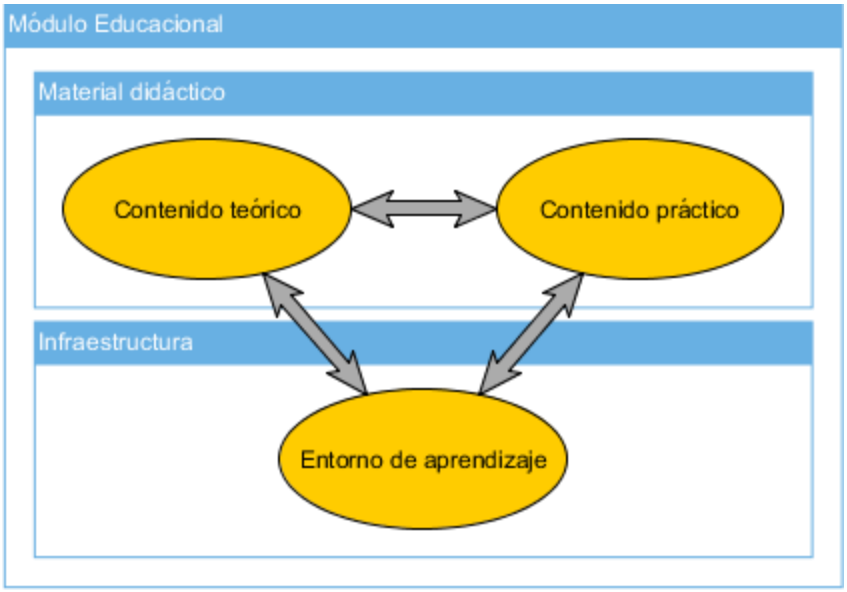

Figura 5. Principales componentes de un módulo educacional. Barbosa y Maldonado (2006a).

El material didáctico está compuesto por contenido teórico y práctico, y con el objetivo de entregar este material a los estudiantes, es necesario que haya una infraestructura adecuada para el aprendizaje.

2.1.2. Un enfoque integral para el modelado de contenido educacional

IMA-CID está compuesto por un conjunto de modelos[13] (conceptual, instruccional y didáctico), donde cada uno afronta aspectos específicos del desarrollo de contenido educacional. A pesar de ello, están intrínsecamente relacionados y deben ser aplicados de una manera integral.

2.1.2.1. Modelo conceptual

Consiste en una descripción de alto nivel del dominio del conocimiento, representando los conceptos principales y las relaciones entre ellos. Éstas pueden ser divididas en dos clases: (1) estructural; y (2) específicas del dominio. Las relaciones estructurales son útiles para definir taxonomías entre conceptos, y hacer inferencias acerca del conocimiento. Las relaciones *tipo-de* y *parte-de* son ejemplos de relaciones estructurales. Por otro lado, las relaciones específicas del dominio son definidas por el usuario, su significado está asociado con un tema en particular y conllevan su propia semántica.

A continuación, se presentará de manera general los lineamientos para la construcción del modelo conceptual:

1. Utilizar el modelo instruccional para identificar los principales conceptos a ser enseñados/aprendidos. Su uso sustancial para describir el dominio del conocimiento puede ayudar a identificar los conceptos.
2. Identificar las relaciones estructurales entre los conceptos seleccionados.
3. Identificar las relaciones específicas del dominio entre los conceptos seleccionados.

2.1.2.2. Modelo instruccional

Además de los conceptos, es necesario que otros tipos de información y elementos instruccionales sean considerados como parte del dominio del conocimiento. Este modelo es quien se encarga de definir esta información adicional, relacionándola con los conceptos previamente definidos. Debe tenerse en cuenta que el modelo instruccional no tiene interés en cómo está relacionada la información, sino en identificar los tipos de información que permitan desarrollar contenido educacional más significativo y motivante.

La construcción del modelo instruccional está formada por dos fases: (1) el refinamiento del modelo conceptual; y (2) la definición de los elementos instruccionales. Durante el refinamiento del modelo conceptual se especifica qué tipo de información adicional (denominada *ítems informativos*) puede ser incorporada a los conceptos ya definidos. IMA-CID adoptó la definición Teoría de Visualización de Componentes (CDT, Component Display Theory, propuesto por Merrill (Merrill, citado por Barbosa, 2006, p. 6)), la cual especifica como ítems informativos a los siguientes elementos:

[13] Un modelo es un conjunto coherente de elementos formales que describen algo construido para algún propósito susceptible a alguna forma de análisis. En el apartado siguiente se desarrollará el concepto teórico del diseño orientado a modelos.

- *Conceptos*: símbolos, eventos y objetos que comparten características y están identificados por un mismo nombre. Representan una gran parte del lenguaje y entenderlos es esencial para poder comunicarse.
- *Hechos*: piezas de información lógicamente asociadas.
- *Procedimientos*: un conjunto de pasos para poder solucionar un problema o alcanzar un objetivo.
- *Principios*: explican o predicen por qué algo ocurre de una manera determinada.

Durante la fase dos se definen los elementos instruccionales:

- *Elementos explicativos*: representan la información complementaria utilizada para explicar un tema en particular.
- *Elementos exploratorios*: permite al estudiante navegar a través del dominio teórico, conceptos prácticos y otra información relevante.
- *Elementos evaluativos*: permite evaluar la competencia del estudiante sobre el dominio teórico.

2.1.2.3. Modelo didáctico

Este modelo es responsable de establecer los prerrequisitos y secuencias de presentación de los elementos conceptuales e instruccionales. Es así por lo que el modelo didáctico puede ser utilizado para ilustrar la manera en que el espacio didáctico varía a medida que es navegado por el estudiante. Los modelos didácticos también son útiles para representar contextos dinámicos de aprendizaje, donde los elementos del contenido son representados según determinados parámetros, definidos en términos de la característica del curso, estudiantes y docentes.

2.2. Método para el Desarrollo de Objetos de Aprendizaje (LODM)

LODM establece que el diseño de objetos de aprendizaje afecta en gran medida las experiencias durante el aprendizaje, en el enfoque de las actividades y en el grado de confianza en relación con la efectividad del aprendizaje que brindan las actividades (Graciotto et al., 2011, traducción propia).

Por otra parte, los autores de LODM plantean la necesidad de que la información que se utiliza para desarrollar un objeto de aprendizaje debe ser procesada e incrementada hasta alcanzar el producto deseado. Cada fase en el desarrollo de un objeto de aprendizaje requiere un perfil específico de usuario/desarrollador y genera diferentes modelos que describan esa fase. Por ejemplo, durante la fase de análisis, el mapa conceptual sería un modelo factible para representar la información.

A su vez, LODM considera que es mejor emplear herramientas específicas para cada fase como así también, el hecho de proveer reglas para la transformación de cada modelo. El resultado de cada herramienta —generalmente un archivo definido en un lenguaje de un dominio específico— , es transformado en un modelo más detallado hasta que finalmente se termina desarrollando el

conjunto de objetos de aprendizaje deseado. Debido a que la totalidad del desarrollo está basado en modelos y sus transformaciones, es posible realizar estas transformaciones más fácilmente al requerir la modificación de ciertos parámetros.

LODM es una extensión de IMA-CID, por eso adopta el desarrollo orientado a modelos como marco para la construcción de objetos de aprendizaje. Para las transformaciones, los autores de LODM desarrollaron una herramienta llamada LODE (*Learning Object Development Environment*, Entorno de Desarrollo de Objetos de Aprendizaje), que aún no está disponible para su uso.

2.2.1. Desarrollo orientado a modelos

Es una técnica de desarrollo de software que busca resolver la falta de una actualización constante de los modelos de diseño durante el proceso de desarrollo, a partir de adoptar la transformación e incrementación de modelos en todo el ciclo de vida de un software (Stahl, citado por Graciotto, 2011, p. 2). De esta forma, las modificaciones no son realizadas en el último modelo —ni en la implementación—, sino más bien, en el modelo que captura el conocimiento relevante acerca del cambio. Por ejemplo, si se identifica un problema acerca de la falta de un objetivo de aprendizaje, el cambio es realizado en el primer modelo (de análisis) y se propaga al resto a través de transformaciones (semi) automáticas.

Un modelo es un conjunto coherente de elementos formales que describen algo construido para algún propósito susceptible a alguna forma de análisis (Mellor, citado por Graciotto, 2011, p. 2). Estos elementos son descritos en un lenguaje que los expertos del dominio comprenden, por ejemplo, a través de un mapa conceptual. Las transformaciones de los modelos son realizadas a partir del mapeo de funciones que representan el conocimiento recabado del experto. Estas transformaciones pueden ser automáticas o semiautomáticas. Por ejemplo, los objetivos de aprendizaje definidos por el experto deben ser implementados en el objeto de aprendizaje. En vez de depender completamente de una persona para realizar este mapeo, puede utilizarse un conjunto de funciones para inicializar el modelo de diseño del objeto de aprendizaje, donde cada función representa un objetivo de aprendizaje. Este proceso formal de transformación reduce el riesgo de perder información durante la transformación de un modelo (por ejemplo, no considerar un objetivo de aprendizaje).

2.3. Teoría de Diseño de Objetos de Aprendizaje y Secuenciación (LODAS)

LODAS significa *Learning Objects Design And Sequence theory*, y representa una teoría de diseño instruccional creado específicamente para dar soporte al diseño de objetos de aprendizaje. LODAS sigue un conjunto de principios contenidos en otras teorías de diseño como base fundacional (Wiley, 2000, traducción propia):

2.3.1. Alcance

- *Los objetos de aprendizaje no deben ser todos del mismo tamaño*; y mientras algunos pueden ser pequeños, deben ser combinados en objetos de aprendizaje suficientemente grandes para enseñar el epítome o la actual elaboración de un concepto. El epítome de un concepto significa la definición del concepto en su completitud de la manera más sencilla posible. Por otro lado, una elaboración de un concepto hace referencia a una versión un poco más compleja. De esta forma es posible identificar cuál sería el paso inicial de enseñanza y su alcance.
- *Los eventos instruccionales y las actividades son resultado de la instancia de un modelo de trabajo*, y deben ser lo suficientemente grandes como para enseñar conceptos significativos del mundo real. Uno o más eventos instruccionales pueden formar un objeto de aprendizaje (Síntesis de Modelo de Trabajo y Teoría del Dominio). Un modelo de trabajo es una especificación de cómo varios eventos instruccionales pueden ser creados, facilitando la creación de objetos de aprendizaje con elementos similares.
- *Los objetos de aprendizaje pueden tener dos formas: como clúster de habilidades* (nivel macro) y como problemas específicos (nivel micro). Los *clústeres de habilidades* deben tener un alcance limitado, de forma que un clúster en particular no requiera de más de doscientas horas para aprenderlo. El primer clúster debe ser lo más pequeño posible como para que los estudiantes que se inicien practiquen con una versión simplificada pero auténtica del problema real. El clúster final debe ser lo suficientemente grande como para apoyarse en todas las habilidades constitutivas identificadas en el análisis preliminar.

2.3.2. Secuencia

- Los objetos de aprendizaje deben ser presentados de menor a mayor complejidad, empezando con el caso *epítome*, o de menor complejidad.
- Los 'objetos de aprendizaje' deben ser ordenados de forma que simulen una situación del mundo real de menor a mayor fidelidad. A su vez, debe tenerse en cuenta el grado de dificultad de cada uno de ellos, de manera tal que los más sencillos estén al principio de la secuencia, y los más difíciles, al final.
- Los objetos de aprendizaje deben ser ordenados según su tipo y nivel. Los *clústeres de habilidades* de nivel macro deben ser actividades separadas, de forma que las habilidades sean enseñadas una a la vez y combinadas con gradualidad. Los clústeres de nivel intermedio deben ser ordenados como una única tarea, donde cada habilidad sea enseñada simultáneamente. Los *micros clústeres* pueden ser secuenciados de menor a mayor complejidad o, de ser posible, de manera aleatoria con el fin de promover la transferencia del conocimiento.

2.3.3. Objetivos, condiciones y métodos

2.3.3.1. Objetivos

LODAS fue diseñado para alcanzar cinco objetivos principales:

1. Catalizar el diálogo sobre el uso de objetos de aprendizaje en el contexto del diseño instruccional.
2. Proveer soporte explícito para el diseño de objetos de aprendizaje.
3. Proveer soporte explícito para la secuenciación de objetos de aprendizaje.
4. Proveer soporte de reusabilidad de objetos de aprendizaje.
5. Proveer compatibilidad con experiencia basada en el dominio e investigación de modelado de aprendizaje.

2.3.3.2. Condiciones

LODAS plantea ciertas condiciones que deben alcanzarse para que su implementación sea efectiva; en caso de no cumplir alguna de éstas creará, casi con certeza, una gran variación en las actividades de evaluación del diseño instruccional, así como también en las evaluaciones de las actividades (etapa fundamental para la mejora continua del diseño instruccional).

2.3.3.2.1. Tipos de contenido

El contenido utilizado por los estudiantes es comúnmente dividido en tres tipos, según lo que esperan recibir de los docentes: conocimiento, habilidades y aptitudes. El modelo 4C-ID (*Four Component Instructional Design*, Diseño Instruccional de Cuatro Componentes), del cual LODAS toma ciertos elementos, se enfoca específicamente en el *dominio de habilidades cognitivas complejas*. Van Merriënboer (citado por Wiley, 2000, p. 42), las describe como:

- *complejas*, en el sentido de que (1) abarcan un conjunto de habilidades constitutivas, y (2) al menos una de esas habilidades requiere de procesamiento consciente; y
- *cognitivas*, indica que la mayoría de estas habilidades constitutivas están en el dominio cognitivo —opuesto al dominio afectivo o motor.

LODAS también exhibe esta restricción de contenido, y aunque los métodos descritos a continuación posiblemente permitirán ofrecer un aprendizaje en cualquier dominio de habilidades, el rigor del enfoque podrá ser sobre exigente para habilidades más simples que puedan ser enseñadas de una manera menos exhaustiva.

2.3.3.2.2. Entorno de aprendizaje

LODAS funcionará mejor en un entorno de aprendizaje donde el docente y el estudiante posean una responsabilidad conjunta en el aprendizaje, y la evaluación es apreciada como una herramienta para facilitar el avance. Los estudiantes deben tener un interés profundo en su progreso a través del dominio del conocimiento, participando por voluntad propia en las

evaluaciones formativas y utilizando los resultados para monitorear su propio aprendizaje. Los docentes deberían utilizar las evaluaciones de manera formativa, asegurándose de apoyar el esfuerzo de los estudiantes.

2.3.3.2.3. Características del estudiante

Los estudiantes que puedan y quieran monitorear y regular su propio aprendizaje tienen mayores chances de éxito en entornos de aprendizaje desarrollados a través de LODAS que aquellos que no quieran o no puedan, debido a que LODAS ofrece acceso en tiempo real al progreso del estudiante. Por consiguiente, es necesario un cambio en la manera de pensar de un estudiante acerca de los resultados de una evaluación. No sólo deben participar con entusiasmo en las evaluaciones sin temor al "fracaso", también necesitan aprender que notas bajas no siempre tienen consecuencias negativas.

Aunque la capacitación es necesaria para orientar a los estudiantes en esta nueva manera de pensar acerca de las evaluaciones y el entorno de aprendizaje, aquellos que estén dispuestos a aceptar este nuevo paradigma de evaluaciones y auto control tendrán grandes chances de éxito en este entorno. Por otro lado, aquellos estudiantes que no estén predispuestos a aceptar este paradigma, es muy probable que tengan problemas para tener éxito en un entorno LODAS.

2.3.3.2.4. Características del docente

El docente debe estar dispuesto a empoderar al estudiante en su propio proceso de aprendizaje, es decir, deben estar dispuestos a delegar parte del control. Además, deben aceptar el uso de herramientas de diagnóstico de evaluaciones al contrario que las herramientas de clasificación. También deben estar dispuestos a explicar los resultados de evaluaciones formativas a los estudiantes sin ser prejuiciosos. Por último, tienen que tomar un rol "pasivo", es decir, guiar al estudiante sin demasiada intervención.

También necesitarán competencias con la computadora para tener éxito en el uso de LODAS. Aún si, emplean objetos de aprendizaje que fueron diseñados y secuenciados por terceros; los docentes deben ser capaces de utilizar y entender el entorno como si fueran un estudiante.

2.3.3.2.5. Implementación de LODAS

LODAS puede ser dividido en dos grandes secciones: prescripciones del diseño instruccional y prescripciones del diseño de objetos de aprendizaje. El desarrollador tiene la posibilidad de elegir cuál sección implementar, pero debe asegurarse que aquello que elija debe implementarse como un modelo intacto; en otras palabras, una vez que se implementa, no debe modificarse.

LODAS funcionará más eficientemente cuando se utiliza en conjunto con una biblioteca digital de objetos de aprendizaje. Mientras que LODAS permite diseñar cada uno de los objetos, uno de sus objetivos es promover la reutilización.

2.3.3.3. Métodos de LODAS

LODAS describe un proceso para transformar contenido en especificaciones para (1) el alcance y diseño de objetos de aprendizaje, y (2) la secuenciación y combinación de objetos de aprendizaje. Los métodos del proceso son divididos en seis categorías: Actividades preliminares; Análisis y síntesis de contenido; Diseño de prácticas y presentación de información; Selección y/o diseño de objetos de aprendizaje; Secuenciación de objetos de aprendizaje, y Recursión para mejorar la calidad.

2.3.3.3.1. Actividades preliminares

Antes de aplicar LODAS, los docentes y diseñadores instruccionales deben comparar los objetivos, valores y condiciones identificadas. Como se describió anteriormente, LODAS no siempre será el modelo instruccional óptimo.

2.3.3.3.2. Análisis y síntesis de contenido

Aquí, las *habilidades cognitivas complejas* a enseñar son divididas en un conjunto de *habilidades constitutivas*. Luego se combinan y recombinan estas habilidades para formar modelos de trabajo, que representan actividades que las personas realizan en la realidad. Además, estos modelos de trabajo se deben ordenar en una escala unidimensional de pericia según el grado de complejidad.

2.3.3.3.3. Diseño de prácticas y presentación de información

Está compuesto por dos pasos, (1) clasificación de los modelos de trabajo y las habilidades constitutivas, y (2) diseño de prácticas y presentación de información. El primero es un prerrequisito y se encarga de categorizar los modelos de trabajo y las *habilidades constitutivas* según sean *recurrentes* o *no-recurrentes*. Las primeras hacen referencia a aquellas habilidades que siempre se llevan a cabo de la misma manera (métodos algorítmicos) y requieren de conocimientos previos para asistir en su desarrollo; las segundas hacen referencia a las habilidades que varían sustancialmente dependiendo el contexto en el cual son empleadas (habilidades heurísticas) y requieren de conocimiento de apoyo para poder completarse.

El segundo paso provee métodos específicos para estos cuatro tipos de contenido: habilidades recurrentes, información pre requerida, habilidades no-recurrentes e información de apoyo:

- **Práctica de tareas principales**: también denominadas *clúster de habilidades*, son aplicables para tareas no-recurrentes y proveen un entorno de trabajo para el resto del diseño instruccional. Son similares a los *modelos de trabajo* en el sentido de que representan una tarea entera.
- **Práctica de tareas secundarias**: ocasionalmente, las tareas principales pueden ser suplementadas por tareas secundarias específicas con el fin de alcanzar un nivel deseado de pericia en la performance de la tarea principal.

- **Presentación justo-a-tiempo de información**: tiene como fin (1) presentar de manera directa la información de la actividad en combinación con el problema asociado, y (2) facilitar el acceso a la información en forma de ayudas durante el desarrollo de la práctica.
- **Promocionar la elaboración y la comprensión**: la presentación de la información debe emplear significativamente las relaciones y vínculos al conocimiento que es familiar. El fin de elaborar información de presentación es el de proveer a los estudiantes los modelos mentales y heurísticos que necesitan para participar exitosamente en el desarrollo de habilidades no-recurrentes.

2.3.3.4. Selección y/o diseño de objetos de aprendizaje

- **Según la taxonomía**: todos los objetos de aprendizaje tienen ciertas cualidades, y son las diferencias en el grado en el cual exhiben estas cualidades que hace que un objeto de aprendizaje sea de un tipo o de otro.
- **Según las características**: según el número de elementos combinados, el tipo de esos elementos, el grado de reusabilidad, misma funcionalidad, entre otros.

2.3.3.5. Secuenciación de objetos de aprendizaje

En este punto del proceso de diseño, el dominio del contenido ha sido analizado y sintetizado, el diseño instruccional fue completado, y los objetos de aprendizaje diseñados en base al diseño instruccional. El paso final de este proceso es el diseño de la secuenciación de los objetos de aprendizaje. Esta secuenciación debe ocurrir en tres niveles: secuenciación de modelos de trabajo, secuenciación de tipos de caso, y secuenciación de problemas específicos.

2.3.3.6. Recursión para mejorar la calidad

Esta recursión debería verse como un proceso continuo. La mejoría de la calidad es parte de cada aspecto de la teoría de diseño. Las métricas de calidad se especifican al inicio del proceso, y guían el diseño como la evaluación continua de calidad.

2.4. Modelo Formal de Objetos de Aprendizaje (FLOM)

El modelo desarrollado por los autores de FLOM (Sánchez et al., 2015, traducción propia) está orientado al trabajo colaborativo para el desarrollo de objetos de aprendizaje en el cual el *facilitador* (el docente) define los objetos de aprendizaje que deberían ser generados para un objetivo educativo en particular, mientras que los *estudiantes* crean los objetos de aprendizaje específicos tomando en cuenta las especificaciones del facilitador. FLOM es el acrónimo de *Formal Learning Objects Model* (Modelo Formal de objetos de aprendizaje) y está formado por un *modelo de composición* y un *modelo de grupo*.

El *modelo de composición* describe los elementos que hacen a un objeto de aprendizaje y la manera en que ellos deben ser integrados de manera que tengan un valor para el estudiante. El *modelo de grupo* describe el comportamiento de los participantes en el desarrollo de objetos de

aprendizaje, y está basado en diagramas que identifican los roles y las actividades involucradas en la construcción de objetos de aprendizaje.

> "*FLOM puede servir como guía para el desarrollo de sistemas de construcción de objetos de aprendizaje debido a que define unívocamente sus componentes y establece formalmente cómo debe llevarse a cabo la participación durante la generación de objetos de aprendizaje*" (Sánchez, 2015, traducción propia).

2.4.1. Estructura de un objeto de aprendizaje

Para FLOM, un objeto de aprendizaje representa el conocimiento adquirido luego de comprender, aplicar, sintetizar y evaluar un tema específico. De esta manera, un objeto de aprendizaje debe consistir en seis elementos:

1. Los objetivos educativos que alcanzar.
2. Las habilidades o competencias que adquirirán los estudiantes.
3. Las competencias necesarias para su uso.
4. El contenido educativo propiamente dicho.
5. Las actividades que el estudiante debe realizar para poder adquirir el conocimiento o desarrollar las competencias definidas antes.
6. El medio de evaluación para determinar si el conocimiento fue adquirido.

2.4.2. Modelo de composición

La arquitectura de este modelo consiste en cuatro grandes capas, las cuales describen los elementos de objetos de aprendizaje y cómo deben ser integrados:

1. **Objetos Digitales (OD)**: son objetos multimedia libres de contexto, como por ejemplo imágenes, vídeos, texto o audio.
2. **Objetos de Información (OI)**: está constituido por OD integrados.
3. **Objetos de Aprendizaje (OA)**: están conformados por al menos cuatro OI; uno que representa un objetivo de aprendizaje, uno para el contenido teórico, otro para la práctica y un último para el proceso de evaluación.
4. **Colecciones de Aprendizaje (CA)**: son grupos de objetos de aprendizaje bajo un mismo contexto y con un mismo conjunto de objetivos de aprendizaje.
   (ver página siguiente).

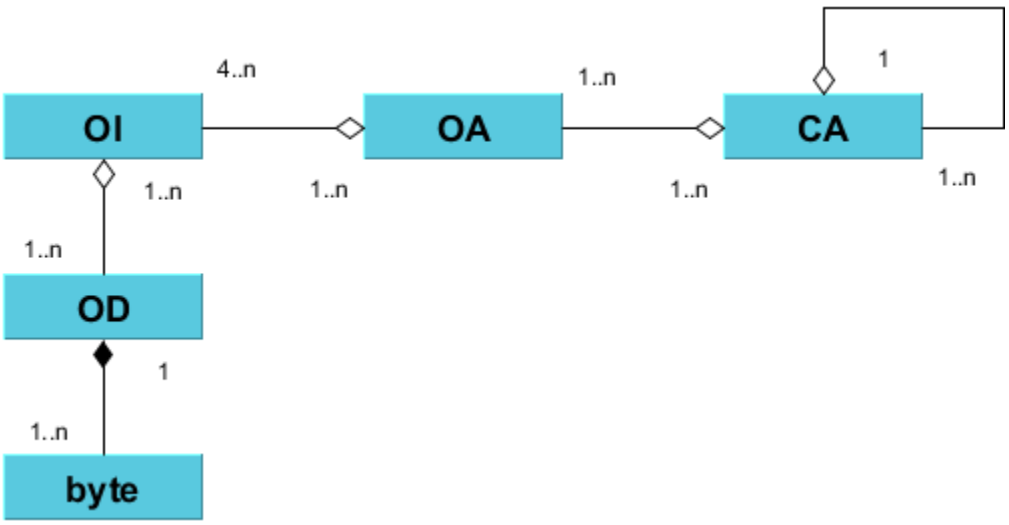

Figura 6. Estructura de un objeto de aprendizaje. Pérez-Lezama, Sánchez, y Starostenko. (2012).

2.4.3. Modelo de Grupo

Este modelo describe los roles, las tareas y las interacciones de cada actor. Existen tres tipos de actores: (1) el *administrador*, quien está encargado de coordinar los cursos y los participantes de cada uno de ellos; (2) el *facilitador*, el cual es el responsable de ofrecer el curso; y (3) el *estudiante*, que toma el curso para adquirir nuevas competencias.

(ver página siguiente).

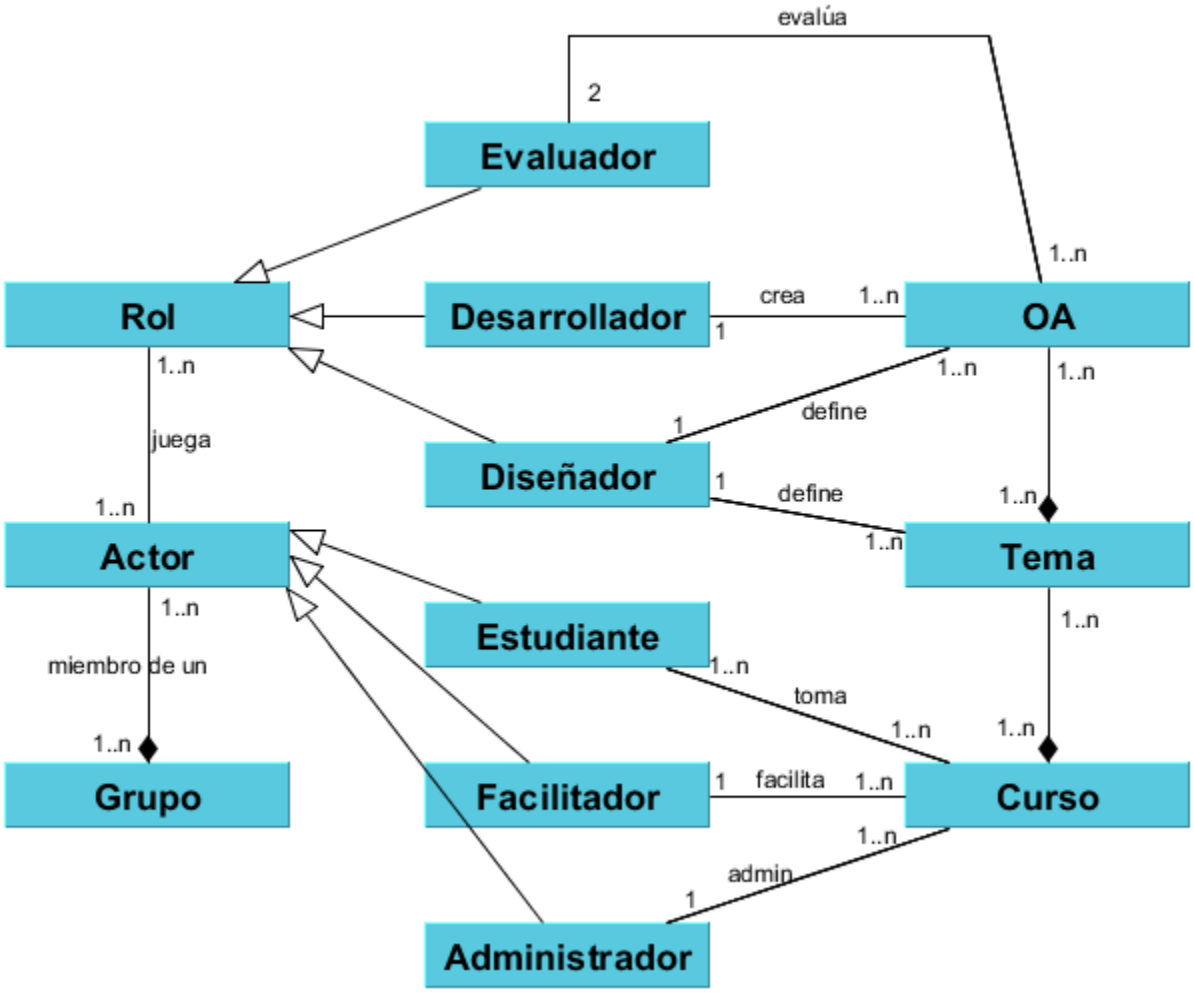

Figura 7. Modelo de Grupo de FLOM. Pérez-Lezama, Sánchez, y Starostenko. (2015).

2.4.4. FLOM como metodología para la construcción de objetos de aprendizaje

El principal objetivo detrás de FLOM es el de desarrollar objetos de aprendizaje en un ciclo continuo de diseño y evaluación. La metodología que plantea está compuesta por seis fases:

1. *Análisis*: los facilitadores definen los temas a ser incluidos en el curso y los objetos de aprendizaje que abarcan cada tema sobre la base de los objetivos de aprendizaje definidos.
2. *Diseño*: en él, los estudiantes deben delinear el contenido de cada objeto de aprendizaje y buscar recursos multimediales para incluir en ellos.
3. *Desarrollo*: proceso de construcción del objeto de aprendizaje.
4. *Evaluación*: se analiza la efectividad del objeto de aprendizaje.
5. *Feedback*: el instructor determina el grado de calidad del objeto de aprendizaje.
6. *Despliegue*: los objetos de aprendizaje son distribuidos a los estudiantes.

Este diseño instruccional es una representación en espiral del ciclo de vida de un objeto de aprendizaje, debido a que FLOM lo considera como un proceso continuo de refinamiento y mejora de los objetos de aprendizaje resultantes. Según los autores, las principales características que se desprenden del uso de este diseño son:

- Puede ser aplicado en la creación de cursos y material instruccional sin importar la asignatura.
- Está estructurado en fases secuenciales e interrelacionadas, lo que significa que el resultado de una es la entrada de la siguiente.
- Es cíclico e iterativo, ya que luego de que el instructor provee el feedback acerca de la calidad del objeto de aprendizaje desarrollado, el estudiante debe rediseñarlo. Así, cíclicamente hasta que el instructor considere que el objeto de aprendizaje está completado.

2.4.5. Validación del uso de FLOM

Los autores de FLOM definieron un prototipo para validar que la implementación de FLOM que se está llevando a cabo cumpla con una serie de elementos. Las principales funciones de este prototipo llamado FLOM-Tool son las siguientes:

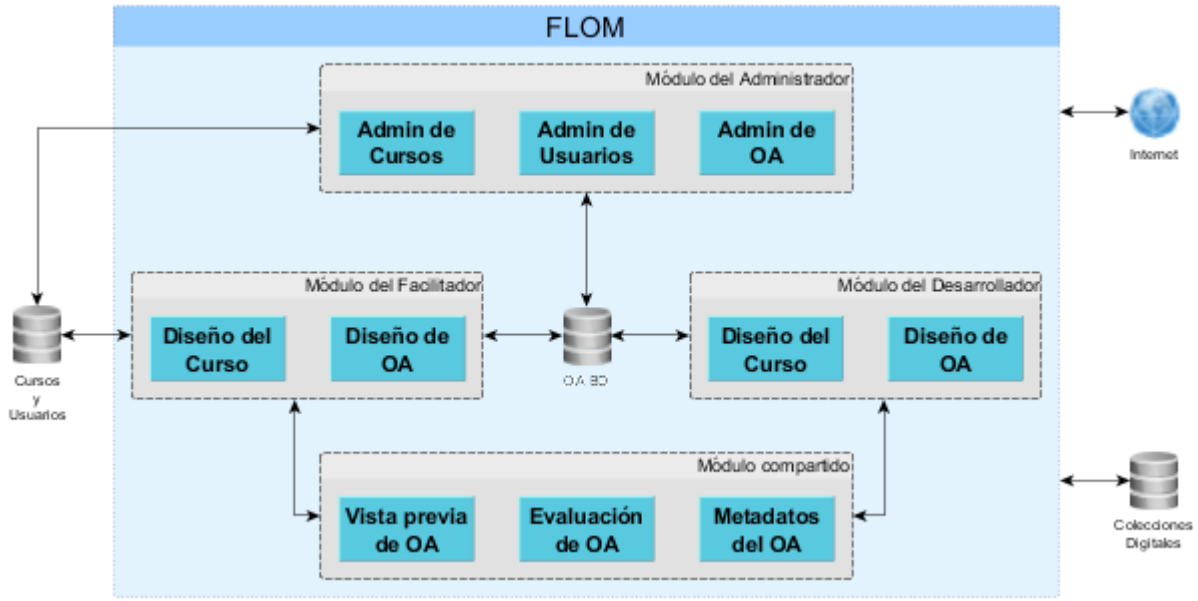

Figura 8. Funcionalidades de FLOM-Tool. Sánchez, A. J., Pérez-Lezama, C. y Starostenko O. (2015).

1. *Diseño del Curso*: el diseñador puede agregar, visualizar y eliminar temas.
2. *Diseño de Objetos de Aprendizaje*: le permite al facilitador definir cada objeto de aprendizaje, incluyendo el código, técnicas didácticas así también como la práctica y evaluación sugerida.
3. *Metadatos del objeto de aprendizaje*: el diseñador también es responsable de definir los metadatos de cada objeto de aprendizaje teniendo en cuenta las principales categorías del estándar LOM (Learning Object Metadata, Metadatos de Objetos de Aprendizaje).

4. *Asignación de objetos de aprendizaje*: el diseñador debe asignar al desarrollador, teniendo en cuenta las habilidades, intereses y conocimiento previo de cada uno de los estudiantes.
5. *Búsqueda de recursos*: los desarrolladores deben buscar recursos multimedia para los objetos digitales que quieran incorporar al objeto de aprendizaje.
6. *Desarrollo de objetos de aprendizaje*: en esta actividad se incorporan los objetos digitales en plantillas predefinidas. Los objetos de información generados se secuencian de una manera lógica con el fin de generar un objeto de aprendizaje.
7. *Previsualización de objetos de aprendizaje*: tanto el diseñador como el desarrollador pueden visualizar y ejecutar los objetos de aprendizaje desarrollados con el fin de verificar que haya una integración correcta.
8. *Evaluación de objetos de aprendizaje*: el diseñador y el desarrollador evaluarán la calidad del objeto de aprendizaje terminado. Esta tarea incluye una instancia de feedback en la cual el diseñador puede darle comentarios al desarrollador acerca de la calidad del objeto de aprendizaje con el fin de mejorarlo.

2.5. Comparación y justificación del marco elegido

Antes de empezar con el análisis de cada marco, es importante remarcar que la elección de un marco teórico concreto para Códimo está fuertemente definido por los siguientes cuatro requisitos no funcionales de la herramienta:

- Es una plataforma para contener actividades de asignaturas de la escuela primaria y secundaria.
- Las actividades tienen como requisito obligatorio que cumplan con el plan de estudios.
- El diseño de éstas debe considerar la aplicación de conceptos del 'pensamiento computacional' como parte troncal del desarrollo de la actividad.
- El desarrollo de las actividades no debe ser llevada a cabo ni por el docente ni por los estudiantes, sino por un desarrollador.

Estos requisitos no funcionales requieren que el diseño sea conciso y detallado, pero que además sea flexible para poder aplicarlo gradualmente y sin afectar demasiado a los docentes que participen en el desarrollo de las actividades.

2.5.1. IMA-CID (2006)

La cualidad más llamativa de este marco es la descripción del módulo educacional como definición de lo que es un objeto de aprendizaje, más específicamente de lo que significa un entorno de aprendizaje: la infraestructura necesaria para que el estudiante pueda realizar los objetos de aprendizaje. Un ejemplo de infraestructura es la plataforma *moodle*, otro sería la plataforma diseñada para Códimo.

Sin embargo, tiene como principal desventaja la curva de aprendizaje que posee. En el trabajo original (Barbosa, 2006), la teoría está acompañada de un ejemplo de aplicación de esta y se puede observar cómo se detallan, con un alto nivel teórico, los elementos, sus relaciones, y los propios modelos. Este esfuerzo sobrepasa la necesidad principal de Códimo: poder ofrecer un framework para el desarrollo y ejecución de objetos de aprendizaje.

Otro motivo que se identifica como impedimento para su aplicación es el hecho de que el docente no aparece como un actor representativo durante el proceso de desarrollo de los objetos de aprendizaje, mientras que para que Códimo sea considerado correcto o útil, las actividades que contenga deben estar fuertemente validadas por ellos.

De todas maneras, es factible afirmar que este modelo sería adecuado para el desarrollo de un objeto de aprendizaje en particular que requiera de una validación teórica profunda.

2.5.2. LODM (2011)

Este marco simplifica en parte la teoría propuesta por IMA-CID en cuanto a la formalidad que requiere cada modelo; mientras que propone un proceso de desarrollo más contemporáneo: desarrollo iterativo-incremental. Además de esto, identifica y define los cambios entre modelos como *transformaciones*, y asegura que pueden automatizarse o semi automatizarse.

Estos dos aspectos hacen de este marco más plausible que el anterior. Sin embargo, hereda un problema propio de IMA-CID: la falta de consideración del docente y el estudiante como actores activos durante la construcción de objetos de aprendizaje, aun cuando menciona que existen roles específicos para cada modelo, ya que solo hace referencia a un desarrollador de objetos de aprendizaje.

2.5.3. LODAS (2000)

Un elemento clave de esta teoría es que al significado de un objeto de aprendizaje le introduce el proceso de elaboración de un concepto, que empieza por el epítome —la definición más completa pero más sencilla, es decir, el caso base—, para luego avanzar a una definición más elaborada. Esta descomposición de la enseñanza está asociada con la otra característica que plantea esta teoría: la secuenciación, que significa la manera en que los objetos de aprendizaje son integrados para construir un proceso de enseñanza más envolvente.

La teoría de secuenciación que plantea está muy bien constituida, ya que describe cómo debe presentarse el concepto a través de los objetos de aprendizaje, de qué manera deben ordenarse para conseguir el mayor provecho, y cómo construir actividades con distintos grados de dificultad que motiven la autosuperación.

Otra característica particular de esta teoría son las restricciones para su uso. Detalla muy bien cuáles son los requisitos del entorno de aprendizaje, del estudiante y del docente; y afirma que de no poder alcanzar alguno de ellos, la implementación de LODAS está condenada al fracaso (describiendo cuáles podrían ser algunas de las posibles situaciones en las que se vería envuelto

el estudiante). Este aspecto puede considerarse como positivo, ya que permite identificar inmediatamente si es posible o no implementarlo en un determinado entorno.

Actualmente, en el sistema educativo argentino no es posible asegurar alguno de esos requisitos, como por ejemplo el hecho de que el docente esté preparado desde un principio para el uso adecuado de las herramientas TIC; o que el entorno de aprendizaje ofrezca un medio en donde el estudiante pueda llevar un registro de su progreso en el aprendizaje y así poder monitorear dicho progreso.

El cumplimiento de estas condiciones *sine qua non* por parte de Códimo generaría un rechazo en vez de aceptación, debido a que el docente, como así también el sistema educativo en general, se verían amenazados por el hecho de exigir de manera autoritaria (y sin un proceso avalado por el sistema) que se tengan que cambiar las costumbres.

Los requisitos planteados por LODAS constituyen buenas metas a largo plazo. Sin embargo, lo necesario para Códimo —en este momento— es poder ofrecer a los docentes, sobre todo a quienes no están acostumbrados en el uso de las TIC, una herramienta no disruptiva que se pueda integrar a la currícula.

2.5.4. FLOM (2015)

El proceso de desarrollo de objetos de aprendizaje que plantea es el que mejor se adecúa a los requisitos de Códimo, en cuanto a considerar al docente como actor central. Otro aspecto favorable de este marco es el hecho de que pueda servir como guía para el desarrollo de sistemas de construcción de objetos de aprendizaje, y no exclusivamente para definir un proceso formal de construcción aislada.

Sin embargo, y esto no es un aspecto negativo sino una restricción (actual) de Códimo, no será posible considerar al estudiante como desarrollador de objetos de aprendizaje por el público al que va dirigido: estudiantes de nivel primario. Justamente Códimo fue concebido como una herramienta para introducir al estudiante en el 'pensamiento computacional', no a la programación. De todas maneras, sí sería factible analizar esto como un objetivo a futuro para estudiantes de secundaria o de los primeros dos años del nivel universitario, ya que ellos sí podrían estar más preparados para llevar a cabo un desarrollo.

A su vez, la sencillez del proceso de construcción permite que su implementación no requiera demasiado esfuerzo. Esto se debe a que cada fase es muy similar a las etapas de desarrollo de cualquier otro tipo de proyecto de software que se estudia en una carrera relacionada a sistemas.

Por todo lo ante dicho, se consideró que el marco más adecuado es FLOM. La implementación de este modelo formal para Códimo será una adaptación, ya que no se considerará al estudiante como actor principal para el desarrollo inicial del objeto de aprendizaje, aunque sí se mantendrá la participación del docente como actor principal para la definición de los requisitos educativos de los ejercicios, con el fin de definir el proceso de la actividad.

Tanto la descripción de la estructura como la categorización de los objetos de aprendizaje es un punto fuerte de este modelo, porque permite describir en forma desagregada los componentes al momento del diseño y desarrollo y para poder realizar una buena base de datos de objetos de aprendizaje.

2.6. FLOM para Códimo

2.6.1. Estructura de un objeto de aprendizaje

A la definición de la estructura de un objeto de aprendizaje propuesta por FLOM es necesario detallar cada uno de los aspectos:

- *Los objetivos educativos que alcanzar*: éstos deben ser definidos principalmente por el docente. La actividad debe centrarse en reforzar uno o varios conceptos dictados en clase.
- *Las habilidades o competencias que adquirirán los estudiantes*: además de que la actividad deba reforzar lo aprendido en clase, será más enriquecedor si se puede introducir o profundizar alguna habilidad del 'pensamiento computacional'.
- *Las competencias necesarias para su uso*: esto no requerirá de demasiado esfuerzo ya que el docente las habrá definido en la currícula de su materia. Vale aclarar que estos requisitos serán únicamente informativos, tanto para el estudiante que intente solucionarlo, como para el desarrollador que quisiera realizar una actividad más compleja a partir de otros objetos de aprendizaje. Otro aspecto importante para aclarar es que esta primera versión de Códimo no dará soporte nativo para la secuenciación de objetos de aprendizaje, por lo que las competencias necesarias no actuarán como restricciones.
- *El contenido educativo propiamente dicho*: está formado por las imágenes, los componentes visuales, las diferentes instrucciones que dispone la actividad, como también por el código escrito para animar cada elemento de la actividad.
- *Las actividades que el estudiante debe realizar para poder adquirir el conocimiento o desarrollar las competencias definidas antes*: en Códimo, una actividad está constituida por un conjunto de niveles separados en tres dificultades: fácil, normal y difícil, donde es posible que cada dificultad posea bloques específicos, por ejemplo, alguno de iteración, de repetición, o de control.
- *El medio de evaluación para determinar si el conocimiento fue adquirido: una primera etapa para resolver este punto ya fue alcanzada*: cada actividad debe tener programada la solución correcta y cuáles representar un error. Esta evaluación sólo se preocupa del resultado final y sirve para darle un cierre al ejercicio.
  Resta implementar un mecanismo que le permita al docente evaluar el proceso del estudiante, ya sea a través de algún cuestionario o al evaluar cómo fue el desarrollo hasta encontrar la solución correcta (es decir, qué bloques utilizó, de qué manera los ordenó, cuáles sacó o qué tipo de soluciones fallidas tuvo hasta alcanzar la correcta).

2.6.2. Modelo de composición

1. Objetos Digitales (OD).
    a. Imágenes.
    b. Metadatos.
    c. Texto.
    d. Vídeos.
    e. Sonidos.
2. Objetos de Información (OI).
    a. Un número.
    b. Un laberinto.
    c. Una recta numérica.
    d. Los bloques.
    e. Los botones.
    f. Los carteles informativos.
    g. El diseño de una actividad.
3. Objetos de Aprendizaje (OA). Formado por:
    a. *Un OI que represente el objetivo de aprendizaje*: es el diseño central del objeto de aprendizaje. Debe ser elaborado principalmente por el docente y con apoyo del desarrollador. No es visible de manera directa en la actividad, sin embargo, es el componente central que le da sentido al objeto de aprendizaje.
    b. *Un OI que represente el contenido teórico*: deberá ser tomado en cuenta, pero con cierta precaución, ya que —como se confirmó en una reunión con los docentes[14]— al ser una herramienta de apoyo, Códimo no debe contener teoría al respecto. Esto tiene sentido ya que la manera de dar un concepto varía según el docente y el grupo de un año en particular, por lo que Códimo debe evitar presuponer lo más que pueda en relación con este aspecto.
    c. *Un OI para la práctica*: definitivamente, una actividad de Códimo requerirá más de uno. Esto puede observarse en el punto dos de esta numeración, donde se detallan algunos de los OI de la actividad "*Recta Numérica*" de Códimo, descrita al final del capítulo 4.
    d. *Un OI para el proceso de evaluación*: será una reflexión sobre lo que se aprendió con la actividad. Estará constituido como un múltiple choice, con oraciones sencillas y, posiblemente, con alguna opción que describa un concepto que no se trabajó, a modo de trampilla. No será implementado en este trabajo, porque está fuera de su alcance, como, por ejemplo, tener que desarrollar un sistema que permita llevar un registro de los estudiantes que accedieron, realizaron la actividad y su evaluación, y cuál fue su resultado.

[14] Ver Anexo 2017-08-17 Meeting N. º 03.

4. Colecciones de Aprendizaje (CA): con respecto a este objeto, no será implementado en este trabajo ya que requiere de secuenciar un conjunto variado de objetos de aprendizaje, lo que supera el alcance de esta tesis.

2.6.3. Modelo de grupo

En Códimo, el rol del Evaluador, Diseñador y Facilitador será llevados a cabo por el docente. Evaluador porque es quien determina si el objeto de aprendizaje construido cumple con el objetivo inicial; si será posible que los estudiantes lo utilicen sin sentirse abrumados, o si posee un grado de dificultad adecuado. Diseñador porque es quien lleva a cabo la planificación de la asignatura, y, por ende, es quien sabe cómo será dado un concepto. Facilitador porque es quien interactúa con los estudiantes.

El rol de Desarrollador y Diseñador estarán representados por una persona con conocimientos en programación. Este grado de amplitud para este rol es adrede ya que Códimo es un proyecto de código abierto, por lo que cualquiera puede desarrollar su propio objeto de aprendizaje e integrarlo en una plataforma. También se considera a esta persona en el rol de Diseñador debido a que indiscutiblemente, quien desarrolle, deberá ayudar al docente en el proceso de diseño, mientras que al mismo tiempo tendrá que llevar a cabo un diseño a nivel de componentes, más cercano al desarrollo.

Por último, el rol de Administrador no será contemplado en este trabajo, pero de todas formas se considera adecuado presuponer que el referente TIC de la escuela sería un buen candidato para desempeñarlo.

2.6.4. Metodología de construcción de Objetos de Aprendizaje

1. *Análisis*: el docente y el desarrollador del objeto de aprendizaje deben considerar qué actividades son las más adecuadas para ser construidas.
2. *Diseño*: una vez elegida la actividad a desarrollar, se debe tomar la currícula elaborada por el docente para la materia que tiene a cargo, analizar lo planteado para el objetivo de aprendizaje que se decidió representar en un objeto de aprendizaje y diseñar de qué manera será construido el mismo: ¿como un juego de tipo rompecabezas, como un trabajo de investigación, o como una actividad para armar oraciones?
   Quien se encargue del desarrollo también debe tener en cuenta qué conceptos del 'pensamiento computacional' querrá integrar en la actividad.
3. *Desarrollo*: en el proceso de construcción del objeto de aprendizaje se deben seguir los estándares propuestos en el capítulo 4, Diseño de Software, para permitir que cualquier otra persona que quiera aplicarle un cambio no deba esforzarse para comprender la forma de programar de otro. Estos estándares también aseguran un cierto grado de calidad ya que, entre otros aspectos, obligan la implementación de tests visuales y de ejecución automática que aseguran que todo funciona como se espera que lo haga.

4. *Evaluación y Feedback*: el docente debe probar el objeto de aprendizaje para validar que cumple con los objetivos de aprendizaje propuestos y, sobre la base de ello, ofrecer feedback que le permita al desarrollador resolver las deficiencias planteadas.
5. *Despliegue*: el objeto de aprendizaje será integrado a la plataforma de actividades de Códimo para que pueda ser utilizado por los estudiantes.

2.6.5. Validación del uso de FLOM

Este proceso no será llevado a cabo en este trabajo porque Códimo se encuentra en una etapa muy temprana de su desarrollo y no posee un conjunto suficientemente grande de objetos de aprendizaje como para analizar el grado de éxito de la implementación de este marco teórico.

Sin embargo, se puede afirmar que las dos actividades desarrolladas, "*Recta Numérica*" y "*Múltiplos y Divisores*", descritas en la última sección del capítulo 4, pudieron desarrollarse siguiendo todos los pasos que fueron planteados en la sección anterior, lo que representa un paso muy positivo para poder afirmar que la decisión de utilizar este marco fue la más adecuada.

Capítulo 3: Diseño del framework de Códimo

La reutilización de un artefacto de software es la característica que tiene para poder ser utilizado nuevamente, variando su forma inicial o el contexto para el cual fue construido originalmente (Deutsch, citado por Campo, 1998). Esto se diferencia de la mera utilización en el sentido de que esta última no afecta ni la forma ni el contexto original.

Según Campo (1998), en su libro "Desarrollo de Frameworks Orientados a Objetos", existen dos formas básicas de reutilización: (1) reutilización de porciones de código y (2) reutilización de unidades funcionales completas, las cuales pueden tomar la forma de procedimientos, paquetes o módulos. Mientras que en el primero se copia y se modifica el código en un nuevo contexto, en el segundo mecanismo de reutilización es necesario definir interfaces que permitan que el código a reutilizar pueda comunicarse con el código cliente.

En el caso de Códimo, el mecanismo que se implementó para reutilizar la funcionalidad central de la aplicación fue el framework. Éste es un paradigma de desarrollo en el cual la reutilización es parte integral (Campo, 1998). Además, permite implementar el comportamiento genérico de un dominio de aplicación. En el caso de Códimo, este dominio está constituido por las actividades que refuerzan los conceptos de las materias a través del 'pensamiento computacional'.

1. Códimo como framework de caja negra

Un framework de caja negra es un diseño de software que permite extender un sistema generalmente a través de la implementación de una interfaz. Durante la ejecución, el framework se encarga de ejecutar las clases internas de él, como las implementadas por el código cliente, sin necesidad de que el usuario del framework defina nada más. Códimo hace uso de esta técnica para facilitarle al usuario el desarrollo de las actividades. El objetivo principal del framework de Códimo es abstraer al usuario de:

1. Qué librería elegir como *Engine* (motor gráfico).

2. Definir cómo deben procesarse las instrucciones.
3. De qué forma debe programarse una funcionalidad de un componente del *Engine*.
4. De qué forma debe programarse el procesador de una instrucción.
5. Qué librería elegir para el *Workspace* (espacio de trabajo).
6. De qué forma se diseñan las instrucciones.
7. Qué librería utilizar para el sitio web.
8. De qué forma instanciar el *Engine* y el *Workspace* y realizar la comunicación interna entre ellos.
9. Cómo informar al usuario de la aplicación cada evento de la actividad.

De esta manera, el usuario debe enfocarse en:

1. Desarrollar los componentes del *Engine* siguiendo la interfaz del generador de componentes.
2. Desarrollar procesadores de instrucciones y funcionalidades de componentes según lo establece la interfaz de cada uno.
3. Desarrollar la parte visual de una instrucción según lo define el Workspace a utilizar.
4. Instanciar la actividad.
5. Definir los niveles.

Debe tenerse en cuenta que en el momento en que se escribió esta tesis, sólo había un único *Engine* y un único *Workspace*, limitando así el tipo de actividades que pueden realizarse. Para poder agregar nuevos motores gráficos o espacios de trabajo, es necesaria una refactorización mayor que no es el objetivo de este trabajo.

2. Principales componentes de Códimo

Antes de poder analizar los distintos puntos de extensión que ofrece el framework, es importante describir brevemente cuáles son los componentes centrales del framework y qué función cumplen.

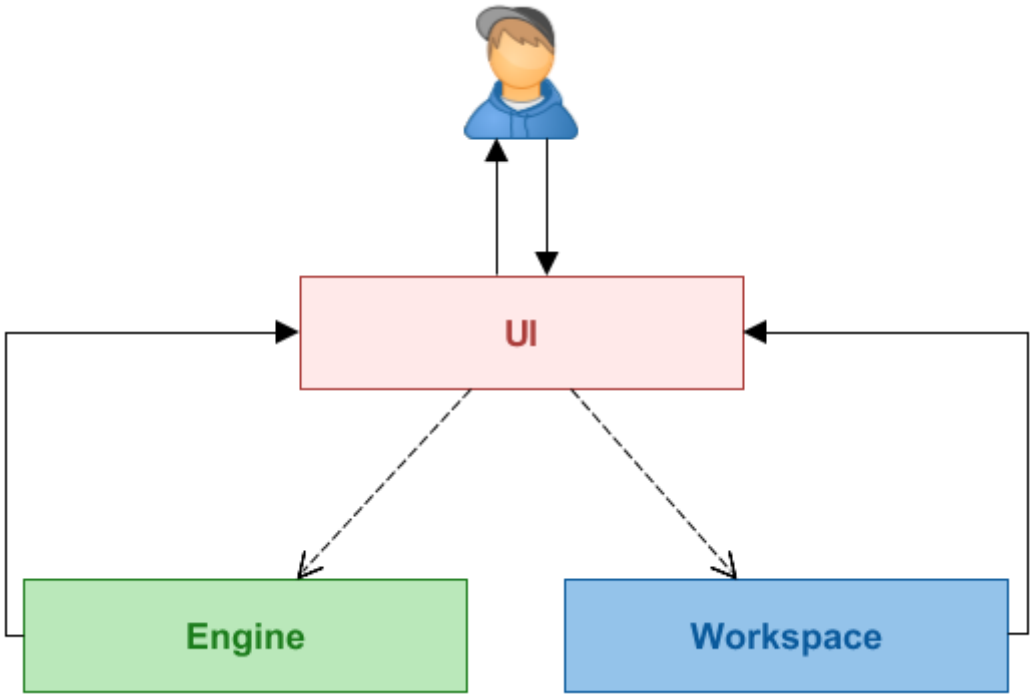

Figura 9. Principales componentes del framework de Códimo.

La **UI** (Interfaz de Usuario) es el componente que interactúa con el usuario (estudiante). A partir de las acciones del usuario, la **UI** las interpreta y delega las responsabilidades de procesar cada acción al **Engine** o al **Workspace** dependiendo de quién tenga la lógica. Una vez que el responsable de procesarlas terminó, le envía la respuesta a la **UI**, la cual se encarga de informarle al usuario los resultados de una manera intuitiva.

El **Engine** (motor gráfico) es el responsable de instanciar todos los componentes visuales estáticos e interactivos, mantener el estado inicial y actual de cada uno de ellos, y ofrecer dos métodos para (1) reiniciar todo al estado inicial y para (2) ejecutar un conjunto de instrucciones.

Por último, el **Workspace** (espacio de trabajo) se encarga de ofrecerle al usuario un conjunto de instrucciones a utilizar, y un espacio donde poder incrustarlas hasta formar una secuencia. Luego procesa esa secuencia de instrucciones en crudo y se las envía a la **UI** para que a su vez se las entregue al **Engine**.

3. Puntos de extensión de Códimo para las actividades

En esta sección se explicarán los puntos de extensión que se utilizan en las actividades. Vale aclarar que a falta de un mecanismo nativo de JavaScript de abstracciones como lo son las interfaces, lo más cercano que se pudo llegar fue a la definición de tipos de objetos a través del uso de *Flow*[15].

[15] Sitio oficial: https://flow.org.

3.1. Engine

3.1.1. Componentes

Un componente es un objeto que posee como mínimo una propiedad: `view`. A este componente, a través del constructor de componentes, es posible inyectarle funcionalidades (descritas a continuación).

El generador de componentes es un objeto que cumple con el siguiente tipo:

```
type Generator = {|
  addFunctionality(
    key: string,
    functionality: FunctionalityBuilder,
  ): Generator,
  build(): CodimoComponent,
|};

type ComponentBuilder = (
  view: Container,
  size: number,
  margin: number,
) => Generator;
```

La función `ComponentBuilder` recibe como primer parámetro una instancia de un contenedor de un elemento visual, que puede ser un número, una pared, un espacio en la recta numérica, una flecha; es decir, los elementos que serán renderizados con el *Engine*. El segundo y tercer parámetros se utilizan como puntos de referencia para el tamaño de cada elemento.

El resultado de esta función es el generador del componente, el cual posee dos métodos. El primero sirve para inyectar una funcionalidad al componente y el segundo, para hacer definitiva la instancia del objeto con sus funcionalidades.

El mecanismo utilizado para la inyección de una funcionalidad se denomina *mixin*. Un mixin es un objeto que contiene métodos que son utilizados en otras clases. La diferencia principal entre mixins y herencia es que los primeros poseen un conjunto fijo y limitados de métodos; mientras que, con la herencia, y sobre todo con la herencia múltiple, la cantidad de métodos de la clase final no se sabe.

Además de métodos, los mixins en Códimo pueden poseer propiedades. Por ejemplo, en la funcionalidad de posicionamiento, se le incluyen tres atributos de sólo lectura al componente. Uno es para saber la posición actual, otro para saber cuál es la posición final y el último para conocer cuál es la posición inicial, necesaria para cuando se requiera reiniciar la actividad.

3.1.2. Funcionalidades

Una funcionalidad es un conjunto de propiedades y métodos que se le inyectan a un componente con el objetivo de que éste pueda realizar ciertas operaciones como saltar, moverse, entre otras. Para poder desarrollar una funcionalidad es necesario que su constructor cumpla con el siguiente tipo de dato:

```
type FunctionalityBuilder = (
  size: number,
  margin: number,
  component: CodimoComponent,
) => Functionality;
```

Como puede observarse, `FunctionalityBuilder` es una función que recibe tres parámetros. Los dos primeros son dos variables bases que se utilizan para definir el tamaño de todos los elementos. El último es el componente de Códimo que está en proceso de construcción. Este objeto, para que cumpla con el tipo `CodimoComponent`, debe tener obligatoriamente la propiedad `view`. Esta propiedad hace referencia al elemento que renderiza la librería del *Engine*.

La funcionalidad realiza operaciones sobre esa vista, como, por ejemplo, moverla en los ejes de coordenadas cartesianas, cambiarla de color, de tamaño, rotarla o reemplazarle el *sprite* que posee, es decir, la imagen, o conjunto de imágenes, asociadas a la vista, y que representan un único elemento.

3.1.3. Procesador de instrucciones

Un procesador es una función que recibe una instrucción y devuelve una promesa. Una promesa en JavaScript es un mecanismo para ejecutar código asincrónico. En otras palabras, es una operación que no sabe cuánto tardará, pero en cuanto está lista, anuncia que terminó. Si en el proceso falló, termina con errores.

Como el proceso de una instrucción requiere mover varios elementos, al ser asincrónico, permite que el procesador de cada elemento se ejecute en paralelo, y una vez que terminaron todos, se continúa con la siguiente instrucción. A su vez, como cada procesador que posee el *Engine* está asociado a un componente en particular, la ejecución de todos los procesadores en paralelo no genera condiciones de carrera.

Existen tres tipos de procesadores: (1) aquellos que se invocan antes de la ejecución de la actividad, denominados `PreExecutionChecker`, y son utilizados para hacer verificaciones previas a la ejecución; (2) otros que, durante la ejecución de la actividad, procesan una instrucción, denominados `ExecutionProcessor`; y (3) aquellos que se invocan al momento de resetear la actividad, los cuales no reciben ninguna instrucción para procesar, y se denominan `ResetProcessor`.

Un procesador del tipo `PreExecutionChecker` debe cumplir con el siguiente tipo:

```
type PreExecutionChecker = (
  instructions: Instructions,
  handleHighlightBlock: HighlightBlockHandler,
) => void;
```

Para que un procesador pueda ser del tipo `ExecutionProcessor` debe cumplir con la siguiente estructura:

```
type ExecutionProcessor = (instruction: Instruction) => Promise<void>;
```

Por último, para que pueda ser del tipo `ResetProcessor` debe cumplir con esta otra estructura:

```
type ResetProcessor = () => Promise<void>;
```

Una ventaja que tiene este enfoque de construcción de procesadores es que es posible realizar chequeos durante la construcción del motor. Ya que un procesador se encarga de realizar operaciones con un componente, es común que esté fuertemente relacionado a una funcionalidad. Por ejemplo, el procesador de instrucciones de movimiento depende de que el componente posea los métodos de movimiento, como así también el estado asociado. Como requiere de ello, durante la instanciación del motor, el constructor del procesador de instrucciones de movimiento se asegura de que el objeto posea los métodos y el estado que necesita. Si no es así, lanza un error que interrumpe la ejecución normal.

Esto último es más una buena práctica que una propiedad de Códimo. Sin embargo, es importante mencionarlo ya que deja en evidencia que es posible interrumpir la etapa de construcción de la actividad, dando un mensaje descriptivo al desarrollador para que sepa qué le faltó, antes de encontrarse con un error poco descriptivo como el famoso "*undefined is not a function*" en medio de la ejecución de la actividad.

Para poder construir un procesador, es necesario cumplir con el siguiente tipo:

```
type ProcessorBuilder = (
  actor: CodimoComponent,
  beforeUpdateStateCheckers: Map<string, Checker>,
  engineData: EngineData,
) => Processors;
```

El segundo parámetro es la colección de verificadores que utiliza el procesador (se explicarán a continuación). El tercer parámetro son los metadatos definidos en la actividad, como, por ejemplo, `size` y `margin`.

3.1.4. Verificadores (checkers)

Un verificador es una función que recibe el estado actual y futuro de un componente, antes de ejecutar la siguiente instrucción, y devuelve una promesa. En su ejecución, verifica que la actividad no caiga en un estado erróneo. Llegado el caso de que así sea, es posible animar el componente acorde al error e interrumpir el proceso de la actividad.

Un verificador debe cumplir con el siguiente tipo:

```
type Checker = (state: any) => Promise<void | Error>;
```

Mientras que el constructor de un verificador debe cumplir con este otro:

```
type CheckerBuilder = (
  actor: CodimoComponent,
  metadata: EngineData,
) => Checker;
```

Por ejemplo, uno de los verificadores del procesador de movimiento analiza si el actor de la actividad se chocó con una pared. En este caso, invoca la animación del "choque contra la pared" del actor e interrumpe la ejecución de la actividad informando al estudiante sobre ese error.

3.1.5. Engine

El *Engine* es un objeto que posee (1) una vista que contiene a todas las instancias de las vistas de los componentes de una actividad, y (2) dos métodos: uno para ejecutar un conjunto de instrucciones y otro para reiniciar la actividad.

Un objeto de este tipo de cumplir con la siguiente definición:

```
type Engine = {|
  view: Container,
  excecuteSetOfInstructions(
    instructions: Instructions,
    handleHighlightBlock: (id: string) => void,
  ): Promise<void>,
  handleResetGame(): Promise<void>,
|};
```

Por otra parte, el constructor de un *Engine* sigue un enfoque similar al del componente:

```
type EngineGenerator = {|
  addPreExecutionChecker(
    key: string,
    checker: PreExecutionChecker,
  ): EngineGenerator,
  addExecutionProcessor(
    key: string,
    processor: ExecutionProcessor,
  ): EngineGenerator,
  addResetProcessor(
    key: string,
    processor: ResetProcessor,
  ): EngineGenerator,
  addWillStopExecutionChecker(
    key: string,
    checker: WillStopExecutionChecker,
  ): EngineGenerator,
  build(): Engine,
|};
```

- El método `addPreExecutionChecker` permite agregar procesadores de pre-ejecución.
- El método `addExecutionProcessor` permite agregar procesadores de ejecución.
- El método `addResetProcessor` permite agregar procesadores de reinicio.
- El método `addWillStopExecutionChecker` permite agregar verificadores a nivel de *Engine*. Estos cumplen con la misma definición que cualquier tipo de verificador, con la diferencia de que se invocan al final de la ejecución de todas las instrucciones. Por ejemplo, es posible que el actor nunca llegue a destino: aunque el conjunto de instrucciones nunca produjo un estado erróneo, el estado final de la ejecución lo es.
- El método `build` invoca la construcción de todos los procesadores de la colección, y si no hay ningún error, devuelve un objeto de tipo `Engine`.

3.2. Interfaz de Usuario

3.2.1. Activity

El único punto de acceso a la Interfaz de Usuario que Códimo ofrece a una actividad se llama `<Activity />`. Es un componente de React que posee cuatro parámetros, y que se encarga de instanciar el *Engine* y el *Workspace*. Para esta tesis, este componente no requiere definir la clase del *Engine* ni del *Workspace* porque instancia el único existente de cada uno. El *Engine* está implementado con PixiJS y el *Workspace* con Blockly.

Las propiedades que requiere un componente de tipo `<Activity />` son las siguientes:

- `@param Array<string> [backgroundImages]`: este atributo opcional es un arreglo con más de un elemento de tipo `string`. Cada elemento debe ser una URL que apunte a una imagen. Estas imágenes se utilizan en el fondo de la actividad y sirven para que sea más ameno.
- `@param Engine engine`: este objeto debe ser una instancia de `Engine`. Internamente, este objeto se lo construye con un componente que se encarga de instanciar PixiJS, y configurar todo para que inicie el *Engine*.
- `@param Metadata metadata`: es un objeto plano que contiene valores de configuración para un ejercicio específico de la actividad, como el tamaño base de los componentes, el camino del número, el rango válido de números, la dificultad, entre otros.
- `@param boolean [hasNoEnd]`: cuando una actividad posee un único ejercicio que no tiene fin, esta propiedad opcional se define con el valor `true`.

Por ejemplo, la instancia de `<Activity />` del Hola Mundo de Códimo es la siguiente:

```
<Activity
  engine={helloCodimoEngine(metadata)}
  metadata={metadata}
  hasNoEnd={true}
/>
```

La función `helloCodimoEngine` instancia el *Engine* de la actividad a partir del objeto metadata que se pasa.

3.2.2. CodimoRouter

Para poder cargar una actividad en un sitio es necesario cumplir con una estructura obligatoria. Un proyecto que quiere integrar Códimo, debe definir en su configuración un alias para las actividades llamado `activities`. Este alias debe apuntar a una carpeta que contenga todas las actividades. Específicamente, Códimo utiliza Webpack[16] como herramienta de compilación, y permite definir alias. Por defecto, éste apunta a la carpeta "`./activities`".

Este requerimiento obliga al usuario de Códimo a que guarde las actividades en un lugar específico y definido por el framework; pero, por otro lado, elimina el esfuerzo que tiene que hacer para configurar y desarrollar ese aspecto. Esto es conocido como una propiedad de diseño de frameworks: *convención por sobre configuración*.

Otra convención que establece este componente es el archivo índice de una actividad. Para poder cargarla, además de que la carpeta raíz de la actividad esté dentro de `activities`, el archivo índice debe llamarse `index.jsx`. Llegado el caso de que ese archivo no exista, Códimo lanzará un error. Este archivo debe exportar por defecto un componente React de tipo *High order*.

[16] Página oficial: https://webpack.js.org.

Un HoC (High order Component) es un componente abstracto que recibe propiedades e instancia otro componente. En este sentido, HoC es al desarrollo web orientado a componentes, lo que las interfaces son a un lenguaje orientado a objetos. En este caso, el objeto que manipula el HoC no sabe qué instancia de ese HoC está manipulando, sólo sabe qué propiedades recibe y qué tipo de componente devuelve. De esta manera, Códimo no sabe qué actividad está cargando, sólo sabe que el HoC de las actividades puede recibir una única propiedad llamada `metadata` y devuelve un componente de tipo `<Activity />`.

La actividad "NumericLine" define el siguiente HoC dentro del `index.jsx`:

```
type Props = {|
  metadata: Metadata,
|};
export default function NumericLine({ metadata }: Props) {
  return (
    <Activity
      backgroundImages={[wp1, wp2, wp3, wp4, wp5]}
      engine={engine(metadata)}
      metadata={metadata}
    />
  );
}
```

Otra convención que define `<CodimoRouter />`, son los ejercicios de una actividad, es decir, de dónde recuperar los metadatos de cada ejercicio. Actualmente existen dos tipos de metadatos: los del *Engine* y los del *Workspace*. Cada uno de ellos debe estar definido en un lugar específico.

Códimo intentará buscar los metadatos del *Engine* en la siguiente ruta:

```
activities/${activityName}/levels/${difficulty}/${level}.json
```

Donde `activityName` es el nombre de la actividad, `difficulty` es la dificultad del ejercicio (`easy | normal | hard`) y `level` es el nivel (puede ser un número o una palabra). Por otro lado, intentará buscar los metadatos del *Workspace* en la siguiente ruta:

```
activities/${activityName}/levels/${difficulty}/blocklyData.json
```

La última convención que define `<CodimoRouter />` está fuertemente relacionada con la anterior, y es la manera de recuperar los tres valores anteriores: la URI. La URL final de un sitio donde debe abrirse una *actividad* de Códimo en un *nivel* de una *dificultad* en particular debe seguir el siguiente formato:

```
/:activity/:difficulty/:level
```

Por ejemplo: https://codimo.netlify.com/NumericLine/easy/001 carga el ejercicio `001` (asociado a `:level`) de dificultad `easy` (asociado a :`difficulty`) de la actividad `NumericLine` (asociado a `:activity`).

4. Requerimientos no funcionales que limitaron la elección de tecnologías

Una característica necesaria para que Códimo pudiera ser fácilmente accesible por los estudiantes de las distintas escuelas —en particular las de Viedma—, es que estuviera en Internet. Asociados con este requerimiento se desprenden otros tres:

1. El único lenguaje que puede ejecutarse en un explorador, actualmente, es Javascript.
2. Debe ser una aplicación liviana a la que se pueda acceder a través de conexiones de media o baja velocidad en un tiempo relativamente rápido, sin que tarde más de un minuto cargando todo.
3. Debe ser de código abierto: el hecho de que las dependencias de Códimo sean de código abierto reduce la probabilidad de que dejen de mantenerse y queden obsoletas.

Con respecto al ítem uno, es posible utilizar una herramienta como Unity3D[17], que permite exportar el desarrollo a diferentes plataformas, entre las cuales están los exploradores web. Sin embargo, este tipo de herramientas tienen ciertas desventajas que no favorecen su utilización para Códimo:

- Es un *Engine* complejo para desarrollo de juegos y animaciones en 2D, 2.5D y 3D. Permite manipular sonidos, modelos gráficos, escenarios, luz, cámaras, físicas y scripts (entre los elementos principales). Esto supera ampliamente lo que se necesita: **un render gráfico lo más sencillo posible**.
- No es un proyecto de código abierto.
- Al fomentar la venta de librerías asociadas al motor, se dificulta encontrar desarrollos que permitan construir Códimo en base a otras herramientas, y así facilitar el desarrollo de los elementos principales del framework de Códimo.

Entre las librerías de código abierto existentes, PixiJS fue la candidata que más interesó:

1. Es mantenida por una empresa que desarrolla animaciones y videojuegos a CartoonNetwork, Disney, Google y Uber, entre otros.
2. Tiene un mantenimiento constante de los desarrolladores de la propia empresa como también de contribuyentes externos.
3. Es liviana: es un render 2D que no manipula cámaras, ni sonido, ni luz, ni físicas, lo que hace que su tamaño sea muchísimo menor.

[17] Sitio oficial: https://unity3d.com.

4. Por último, su rica documentación, como los ejemplos con código que tienen en su sitio, reafirmaron su elección.

Por otro lado, para desarrollar la actividad de matemática que se eligió, fue necesario integrar una librería que construya bloques que representen acciones. BlocklyJS es la librería *de facto*. Fue construida y es mantenida por Google y actualmente existen diversas adaptaciones de la librería original. Sin embargo, no se encontró ninguna que ofreciera una documentación tan rica para una librería tan compleja. Además, BlocklyJS ofrece una herramienta[18] de código abierto[19] para la construcción de bloques.

Por último, se decidió utilizar React como librería para la Interfaz de Usuario. Entre las más similares e importantes se encuentran VueJS, Angular y EmberJS. Los motivos de la elección de React frente al resto de las librerías son los siguientes:

- El tesista hizo prácticas personales con ella, por lo que está más familiarizado.
- El desarrollo en componentes presentado por React facilita la construcción de módulos aislados para el *Engine* y el *Workspace* de Códimo.
- Existen múltiples librerías de terceros de código abierto que permiten reducir las porciones de código Javascript que se envían al explorador web en la primera consulta, lo que permite reducir el tiempo de espera.
- Para el momento en que se analizaron estas tecnologías, React era la única librería que tenía un desarrollo de terceros llamado Storybook, que permitió desarrollar cada componente y cada estado de este de forma separada. Esto facilitó en gran medida la construcción no solo de la interfaz de usuario sino también de los componentes del *Engine*.

[18] Sitio oficial: https://blockly-demo.appspot.com/static/demos/blockfactory/index.html.
[19] Sitio oficial: https://github.com/google/blockly-devtools.

Capítulo 4: Actividades de Códimo

En una primera etapa, se implementó un prototipo de Códimo que consistió en el desarrollo de la actividad "*Recta Numérica*". Este proceso se llevó a cabo dentro del marco del proyecto de extensión "Imaginación y Motivación. Puntos de partida para la enseñanza de la Programación en las escuelas" mencionado en el primer capítulo, en el cual se trabajó con la escuela N.º 296 de la ciudad de Viedma, Río Negro. El trabajo realizado en conjunto consistió en reuniones con docentes asignados por el director, con el fin de poder recibir una crítica del funcionamiento de la actividad. En los anexos se pueden encontrar las dos últimas minutas de reunión que poseen información de qué cosas se requirieron y cuáles se llevaron a cabo.

Una vez completado el primer prototipo[20], se prosiguió con su refactorización y construcción de la primera versión alfa del framework. Con ella, se desarrolló el "*Hola Mundo*" de Códimo: "*Hola Códimo*"[21]. Por último, y utilizando el framework de Códimo desde el inicio, se construyó la última actividad de esta tesis: "*Múltiplos y Divisores*"[22], que reutiliza ciertos elementos de la "*Recta Numérica*", e innova otros aspectos relativos al framework de Códimo.

A continuación, se describirá en qué consiste la actividad "*Recta Numérica*", cómo se desarrolla "*Hola Códimo*" y cómo es el proceso de construcción de la actividad "*Múltiplos y Divisores*".

1. Recta Numérica

Es una actividad pensada para estudiantes de primer ciclo de primaria (6 a 8 años). Se encuadra dentro del área de matemáticas, más concretamente luego de aprender números naturales, enteros, relaciones de orden y la recta numérica. El objetivo siempre es ubicar el número dado en la recta numérica. La actividad presenta diferentes niveles de complejidad. En el nivel inicial se utilizan únicamente números naturales de una cifra; en el siguiente nivel números naturales de dos cifras; y en el último nivel, números enteros.

[20] Puede accederse a la versión final del prototipo aquí: https://goo.gl/65Fk57.
[21] Puede accederse a la primera versión de "*Hola Códimo*" aquí: https://goo.gl/GECvwc.
[22] Puede accederse a la primera versión de "*Múltiplos y Divisores*" aquí: https://goo.gl/hEEFY2.

La consigna es desplazar cada número a través de un laberinto y ubicarlo dentro de la recta numérica, con la restricción de que solo es posible mover el número utilizando un conjunto limitado bloques. Cada bloque tiene una función específica, como mover el número en una dirección, o hacer que salte. Los bloques pueden encastrarse formando un conjunto de instrucciones que se aplicarán sobre el número. De esta forma, utilizando uno de los aspectos del 'pensamiento computacional', el diseño de algoritmos, los estudiantes realizan la actividad.

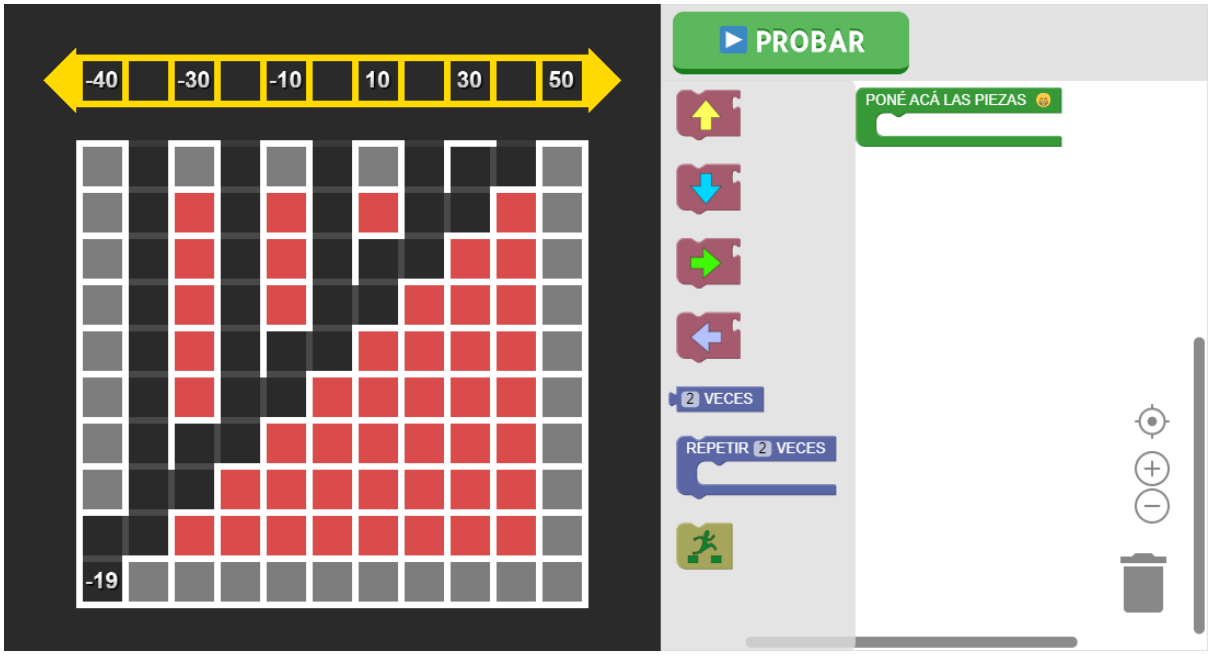

Figura 10. Muestra uno de los ejercicios de nivel avanzado de la actividad.

En la parte izquierda de la figura 10 se puede observar el laberinto y la recta numérica encima de él. La actividad inicia siempre con varios números sobre la recta y un número dentro del laberinto que hay que mover y llevar hacia la recta. En el ejercicio de la figura, el número es el `-19`. La recta numérica posee huecos en los cuales el número puede ser ubicado. A partir del último ejercicio del nivel inicial en adelante, la recta numérica posee huecos donde el número podrá ubicarse correctamente o no.

En la parte derecha de la figura 10 se encuentran los bloques que se utilizarán para mover el número a través del laberinto. Cada uno de ellos posee una función específica que produce un efecto diferente sobre el movimiento que realizará el número. Las funciones de los bloques se dividen en tres categorías:

- Simples:
  - *Flecha hacia arriba*: mueve el número un paso hacia arriba.
  - *Flecha hacia abajo*: mueve el número un paso hacia abajo.
  - *Flecha hacia la derecha*: mueve el número un paso hacia la derecha.
  - *Flecha hacia la izquierda*: mueve el número un paso hacia la izquierda.
- Dependientes:

    - *N veces*: donde N es un número natural mayor a cero. Sólo puede ser utilizado como acompañamiento de un bloque simple.
- Compuestos:
    - *Repetir N veces*: donde N es un número natural mayor a cero. Ejecuta N veces los bloques simples que están dentro de él.
- Especiales de la actividad:
    - *Saltar*: es un bloque que finaliza la ejecución, por ello no permite que se encastre nada debajo de él. Una vez que el número llega a una posición de salida del laberinto, se debe utilizar este bloque para que pueda saltar a la recta.

Los bloques, cualquiera sea su categoría, se deben ubicar dentro del *'bloque de ejecución'*, el cual se identifica por su color verde y su descripción *"PONÉ ACÁ LAS PIEZAS"*.

Al oprimir el botón *"PROBAR"*, Códimo interpreta los bloques uno a uno, realizando el movimiento del número a través del laberinto. Terminada la ejecución, si el número finalizó dentro del hueco correcto de la recta numérica, se sugiere continuar con el siguiente ejercicio, en caso contrario se da una pista de qué error ocurrió. Al reiniciar el ejercicio, el número volverá a su posición inicial, y permitirá volver a ejecutar todo el ejercicio.

La versión del prototipo puede probarse accediendo a este link: https://goo.gl/iLQGZ9.

2. Hola Códimo

Como la mayoría de los frameworks de la actualidad, Códimo ofrece como introducción una actividad muy sencilla: se trata del *"Hola Mundo"* de Códimo. A continuación, se describirá paso a paso cómo construir *"Hola Códimo"*:

1. Instale NodeJS en su versión 8 o superior.
2. Instale Yarn en su versión 1.3.2 o superior.
3. Cree una carpeta y póngale `HolaCodimo`.
4. Acceda a dicha carpeta a través de una interfaz de línea de comandos (puede ser desde la PowerShell o CMD de Windows, o la terminal de Linux).
5. Ejecute `yarn init`. Siga los pasos para que Yarn se encargue de configurar la carpeta como un proyecto de NodeJS.
6. Ejecute `yarn create codimo-activity`. Este comando se encargará de instalar todas las dependencias de producción y de desarrollo para que pueda empezar a programar una actividad de Códimo.
7. Abra la carpeta `HolaCodimo` con su IDE favorito.
8. Cree un archivo `index.jsx`. La extensión JSX se utiliza para diferenciar qué archivos utilizan únicamente JavaScript y cuáles JavaScript + XML (exclusivo de la UI).
9. Escriba el siguiente código:

```
import React from 'react';
import Activity from 'codimo-core/ui/Activity';

import motor from './motor';
import metadatos from './metadatos.json';

export default function HolaCodimo() {
  return (
    <Activity
      engine={motor(metadatos)}
      metadata={metadatos}
      hasNoEnd={true}
    />
  );
}
```

Debe tenerse en cuenta que los archivos `motor.js` y `metadatos.json` no existen todavía. Además, se puede observar que la función tiene el mismo nombre que la carpeta que lo contiene. Esto es obligatorio para que cuando Códimo trate de importar el archivo raíz, el nombre de la función coincida con el nombre de la actividad.

El código anterior es todo lo que necesita el archivo `index.jsx`, y su única funcionalidad es retornar un HoC de React que a su vez devuelve una instancia de una actividad de Códimo, con una instancia de un `motor`, con un conjunto de `metadatos` y con el atributo `hasNoEnd` en verdadero, lo que significa que la actividad no tiene un objetivo en particular, sino que es infinita.

10. Cree el archivo motor.js. Como habrá notado, será importado por el archivo raíz de la actividad sin definir la extensión, mientras que para el archivo de metadatos sí. Esto se debe a que la herramienta encargada de procesar y ejecutar los archivos considera por defecto que el archivo que se importa es `.js` o `.jsx`.
11. Escriba el siguiente código en el archivo `motor.js`:

```
import { Container, Sprite } from 'pixi.js';
import engineGenerator from 'codimo-core/engine/engineGenerator';

export default function motor(metadatos) {
  const vistaPrincipal = new Container();
  const vistaDelActor = Sprite.fromImage('/images/logo.png');

  // Este paso es exclusivo para PixiJS y lo que se está diciendo
  // es que el Sprite del logo de Códimo sea parte de la vista
  // principal.
  vistaPrincipal.addChild(vistaDelActor);

  return engineGenerator(vistaPrincipal).build();
}
```

Hasta este punto, lo único que se hizo fue desarrollar la función del motor de "*Hola Códimo*", importar las dependencias necesarias para construir la vista principal, y generar una instancia —sin funcionalidades— del motor.

12. Aún no funcionará nada porque es necesario crear el archivo de metadatos. Cree el archivo `metadatos.json` con el siguiente contenido:

```
{
  "blocklyData": {
    "blockDefinitions": [
      "move_forward",
      "move_backward",
      "move_right",
      "move_left"
    ],
    "elements": [{
      "define": "block",
      "type": "move_forward"
    }, {
      "define": "block",
      "type": "move_backward"
    }, {
      "define": "block",
      "type": "move_right"
    }, {
      "define": "block",
      "type": "move_left"
    }]
  },
  "engineData": {
    "canvas": {
      "height": 589,
      "width": 700
    },
    "width": 3,
    "height": 5,
    "margin": 0,
    "size": 128
  }
}
```

El primer atributo de este JSON, `blocklyData`, contiene toda la información de los bloques se usarán para el ejercicio. Dentro de él están los atributos `blockDefinitions` y `elements`. El primero define específicamente qué bloques se utilizarán. El segundo determina cómo y en qué orden se listarán los bloques.

El segundo atributo, `engineData`, contiene toda la información con respecto al motor del ejercicio. Existen los atributos obligatorios y los opcionales. Para "*Hola Códimo*" no

requeriremos nada en especial, por lo que basta con lo mínimo y obligatorio. Esta información es la siguiente:

a. `engineData.canvas`: determina el tamaño del elemento `canvas` donde se dibuja todo.
    i. `canvas.height`: determina la altura.
    ii. `canvas.width`: determina el ancho.
b. `engineData.height`, `engineData.width`: determinan el tamaño del *Engine* basado en la unidad de medida para este ejercicio.
c. `engineData.margin`, `engineData.size`: definen la unidad de medida de los elementos del motor. El tamaño (size) determina cuánto ocupa de ancho y de alto un elemento, como un número, una pared, o un hueco en una recta; el margen (margin) define el espacio que hay entre los elementos.

Esto es suficiente para que pueda visualizarse la actividad. Sin embargo, por más que se utilicen los bloques, no habrá forma de hacer que el logo de Códimo se mueva.

13. Para finalizar "Hola Códimo" necesitamos dos cosas. La primera es que el logo de Códimo deje de ser un simple sprite (una imagen estática), pase a ser un actor de Códimo, y que permita moverse. La segunda es que necesitamos que el motor sea capaz de procesar las instrucciones de movimiento. Actualice el archivo motor.js con el siguiente fragmento de código:

(ver página siguiente).

```
import { Container, Sprite } from 'pixi.js';
import engineGenerator from 'codimo-core/engine/engineGenerator';
import componentGenerator from 'codimo-core/engine/componentGenerator';
import {
  positioningFunctionality,
  positioningProcessor,
  positionResetter,
} from 'codimo-core/engine/functionalities/positioning';

export default function motor({ engineData }) {
  const vistaPrincipal = new Container();
  const vistaDelActor = Sprite.fromImage('/images/logo.png');

  vistaPrincipal.addChild(vistaDelActor);


  // Paso 1. Configuro el logo como un actor.
  const actor =
    componentGenerator(
      vistaDelActor,
      engineData.size,
      engineData.margin,
    )
      .addFunctionality('actorPositioning', positioningFunctionality)
      .build();


  // Paso 2. Configuro el motor para que sepa procesar las instrucciones
  // de posicionamiento como también qué hacer cuando se apriete el
  // botón "Intentar de nuevo"
  return engineGenerator(vistaPrincipal)
    .addExecutionProcessor(
      'actorPositioning',
      positioningProcessor(actor),
    )
    .addResetProcessor('actorPositioning', positionResetter(actor))
    .build();
}
```

Con este último cambio del archivo `motor.js` se terminó de configurar la actividad "*Hola Códimo*". Como puede observarse, lo único que se desarrolla es la configuración de la actividad, es decir, qué actores intervienen y cómo deben procesarse las instrucciones que se reciben del usuario. El framework de Códimo se encarga por dentro de instanciar la vista de PixiJS, inyectarle los componentes visuales, procesar y ejecutar cada instrucción y mover todo acorde con ello.

3. Desarrollo de una actividad utilizando el framework de Códimo

Como última sección de este capítulo se describirá el proceso de desarrollo de una actividad utilizando el framework de Códimo.

3.1. Selección de la actividad a desarrollar

Para el proceso de selección se utilizó la minuta de una reunión con los docentes[23]. En ella se plasmaron ideas que dieron los docentes para las actividades futuras, entre las cuales está la que se decidió desarrollar como segunda en esta tesis:

> *Definir ejercicios con múltiplos y divisores donde también haya intrusos, es decir, números que no son divisores ni múltiplos. Para esto sería necesario otra actividad, donde en el fondo haya muchos números, se los deba tomar y luego mover al lugar indicado.*

Esta actividad debería realizarse luego de terminar, al menos, el nivel inicial de la Recta Numérica. Porque, por el lado del pensamiento computacional, tendrá bloques que representarán tareas o procesos; y por el lado de la Matemática, además de ubicar los números en los huecos correctos, serán más de uno, y habrá reglas matemáticas específicas, como "todos los múltiplos de dos".

La actividad "Múltiplos y divisores" consistirá en ejercicios con tres tipos de dificultad diferente que estarán dadas por la cantidad de números a mover, como también por el rango de números válidos.

[23] Ver Anexo 2017-08-17 Meeting N. º 03.

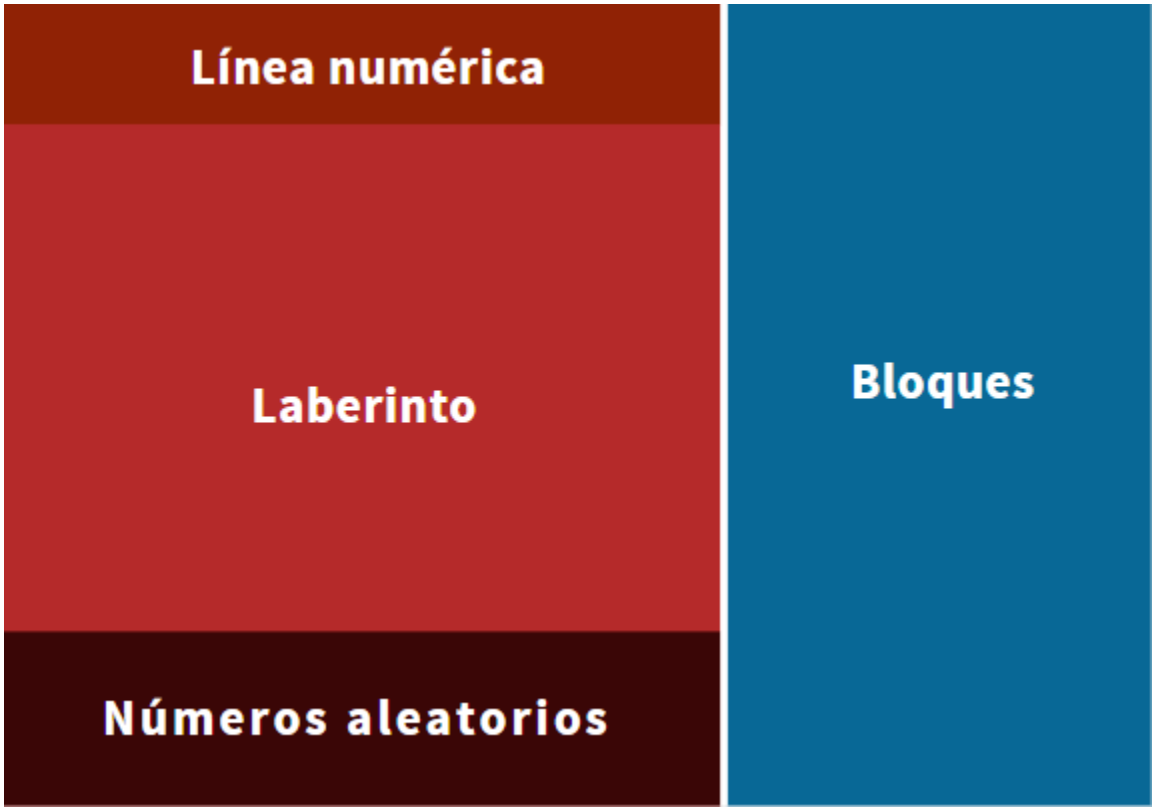

Figura 11. Mockup de los componentes visuales de la actividad.

Como puede observarse en la figura 11, la diferencia entre esta actividad y "Recta Numérica" es sutil, sólo hay una sección nueva, aparte de lo que ya existe. Sin embargo, lo que sí varía es el grado de dificultad que posee, tanto para el estudiante que trate de resolverla, como para su desarrollo.

Para su desarrollo será necesario construir un nuevo tipo de bloque. Éste debe permitir seleccionar un número entre los que están en el espacio de "Números aleatorios" como se aprecia en la figura 11. Una vez seleccionado el número, se utilizarán los distintos bloques ya conocidos para mover un número a través del laberinto y para saltar a la recta numérica.

A su vez, el *Engine* debe adaptarse para que identifique qué número debe ir en qué posición, teniendo en cuenta si se quieren todos los múltiplos de X o todos los divisores de Y. Por último, debe haber nuevos mensajes de error que avisen al estudiante qué pasó.

3.2. Bitácora del proceso de desarrollo

Esta actividad tiene como fin demostrar el potencial del framework de Códimo ya que es la primera actividad que se desarrolla desde el inicio utilizando esta herramienta. En este sentido, se detallará cómo fue el proceso de desarrollo, qué dificultades hubo, qué facilidades y qué debió refactorizarse del framework.

El proceso de cambios puede observarse en el siguiente link: https://goo.gl/cgE7KX.

3.2.1. Creación del esqueleto básico de la actividad

El primer paso es crear una carpeta dentro de la carpeta de actividades. Debe cumplir con las siguientes reglas:

Su nombre no puede tener espacios, y debe estar en `CapitalCase`.

- Debe contener al menos un archivo `index.jsx` dentro suyo que exporte un componente de React, el cual debe devolver un único elemento de tipo `codimo-core/ui/Activity`.

En el caso de esta actividad, el nombre de la carpeta contenedora será: `MultipliersAndDivisors`.

3.2.2. Definición de las dependencias

Una vez definida la carpeta raíz de la actividad junto al archivo index.jsx, es necesario definir qué dependencias posee. Para ello se debe crear el archivo `package.json`. Éste no tiene funcionalidad en la versión actual del framework, sin embargo, es útil para tener un seguimiento de qué necesita la actividad para funcionar adecuadamente, así como también algunos metadatos, como el ID que posee, la descripción o la versión.

La única dependencia que tendrá esta actividad es:

- `codimo-core` en su versión `^1.0.0`.

Como se mencionó antes, esta actividad comparte muchos componentes con la "*Recta Numérica*". Sin embargo, se decidió que no se hará una refactorización al mismo tiempo que se desarrolla la actividad. Para la primera versión de la actividad se hará duplicación de código. Una vez identificados los puntos exactos de duplicación, se decidirá qué componentes merecen ser integrados al framework.

3.2.3. Primera extensión del componente Número

El primer desarrollo será la animación que el número necesita para salir del contenedor de *Números aleatorios* al *Laberinto*. Esta animación consistirá en un efecto *fade out*, una transformación espacial y un efecto *fade in*. El primero hará que el número "desaparezca" del contenedor de números aleatorios, luego deberá ser transportado a la posición de inicio del laberinto, y por último el efecto *fade in* hará que el número "aparezca" en esa posición. Como resultado, se espera que estudiante sienta que el número se teletransportó.

Para el desarrollo aislado de esta animación se utiliza Storybook. De esta forma, las carpetas y archivos que debemos crear son las siguientes:

- `./engine`: esta carpeta contiene todos los elementos del *Engine* que son propios de la actividad.
- `./engine/components`: ésta contiene todos los componentes del *Engine*.

- `./engine/components/numberGenerator.js`: es el constructor del componente. Se creó a partir de duplicar el archivo del mismo nombre de la actividad "*Recta Numérica*".
- `./engine/components/numberGenerator.stories.jsx`: este archivo contiene todas las historias relacionadas con el componente `numberGenerator`.

Una vez creadas las carpetas y los archivos, se migró parte del contenido original de `numberGenerator.js` de la actividad "Recta Numérica". Para esta etapa inicial se eliminó del componente todo lo que no era necesario de ser renderizado. Entre los elementos que se identificaron fueron:

- El generador de números estáticos.
- Los parámetros y funciones para generar números aleatorios.
- Las funcionalidades extras del número.

A partir de esta configuración básica, se construyó una historia de Storybook. Ésta contiene la animación de "selección" del número y se titula "*Opening the portal*" porque simula un portal de teletransportación.

Luego, se desarrolló la funcionalidad que hizo "teletransportar" al número, llamada "`openThePortal`". Para que una funcionalidad de un componente sea posible asociarla, debe respetar el tipo de dato `FunctionalityBuilder` que define una función con ciertos parámetros de entrada, y un objeto como salida.

3.2.4. Desarrollo de la plataforma de teletransportación

Cuando un número se teletransporta, siempre será al mismo lugar. El estudiante que intente resolver la actividad deberá conocer de antemano qué bloque es la plataforma, y así saber qué instrucciones utilizar para cada uno de los números que debe mover.

Es por ello que el segundo elemento a extender es el 'bloque del laberinto'. Es necesario darle la capacidad de configurarle un color y (posiblemente un ícono) que ayude a identificarlo como la posición inicial.

A diferencia del componente del número, éste ya está integrado al core, por lo que un cambio podría causar problemas en otras actividades, en otras palabras, un *breaking change*. Sin embargo, esto es aceptable actualmente porque Códimo está en una etapa de desarrollo alfa, la cual determina que la versión es inestable y que su interfaz puede variar.

Por otra parte, si Códimo ya estuviera en una versión estable, estos cambios tendrían un proceso diferente: cualquier función que tuviera que eliminarse, debería (en lo posible) llamar a la función que la reemplaza junto a una advertencia que diga que la función inicial está *deprecada* (castellanización de la palabra *deprecated*, que equivale a obsoleto), lo que significa que está en desuso y que en la próxima versión mayor será eliminada.

Dicho esto, se analizará cómo extender o refactorizar el componente de los bloques. El color para la plataforma de teletransportación es constante, y este parámetro ya lo acepta el constructor

de bloques. El único elemento que no considera es el ícono. Sin embargo, es posible asignarlo sin necesidad que cambiar la firma del constructor de bloques.

Dentro de la carpeta `engine/components` de la actividad, se creará un componente llamado `platformBlockGenerator` y tendrá la lógica para extender el constructor de bloques original. Esta extensión no es a través de herencia, sino de composición. La firma de la construcción es exactamente la misma; sin embargo, el constructor de bloques de la plataforma se hace cargo de instanciar y personalizar el bloque final.

El resultado final es el siguiente:

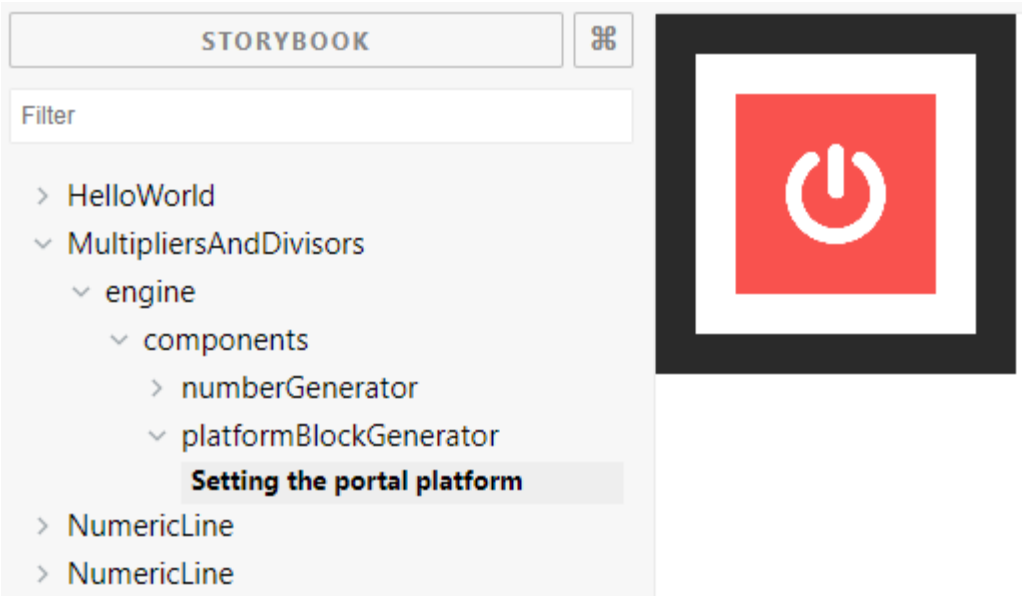

Figura 12. Desarrollo del bloque plataforma.

3.2.5. Construcción del laberinto

El paso siguiente fue duplicar el componente `mazeGenerator` de la actividad Recta Numérica para que tenga en cuenta el bloque de plataforma desarrollado. Éste fue clonado del desarrollo de la actividad "*Recta Numérica*", y luego se le agregó la lógica para detectar el nuevo bloque.

Para que el generador de laberintos sepa dónde renderizar la plataforma de teletransportación, necesita un metadato exclusivo para este tipo de actividades: `platformCoords`.

El resultado final es el siguiente:

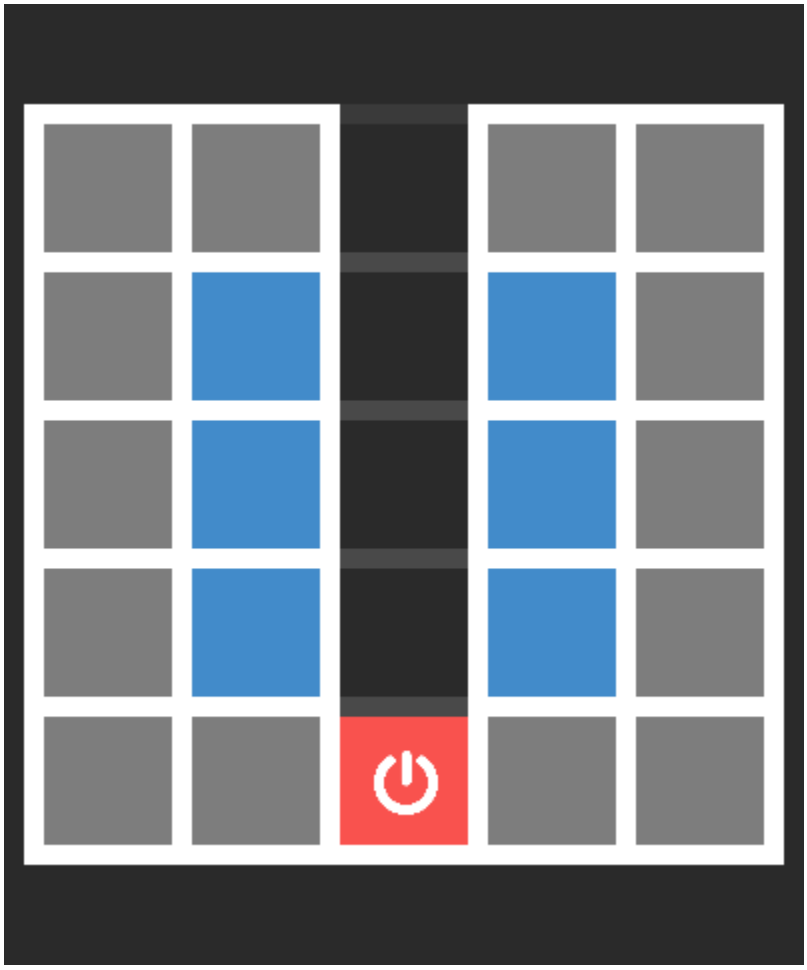

Figura 13. Laberinto de la actividad Múltiplos y divisores.

Para que el bloque de la plataforma sea más fácil de distinguir, se modificó el color de los bloques catalogados como pared. Se cambió la tonalidad rojiza por la azulada que se puede apreciar en la figura 13.

3.2.6. Definición del área de números aleatorios

Los números de esta actividad están ubicados en el lugar de manera desordenada. El área de este componente debe ser opaca y permitir resaltar bien los números de forma que no sobresalga más que ellos. Su desarrollo se basa en el componente de la línea de la recta numérica.

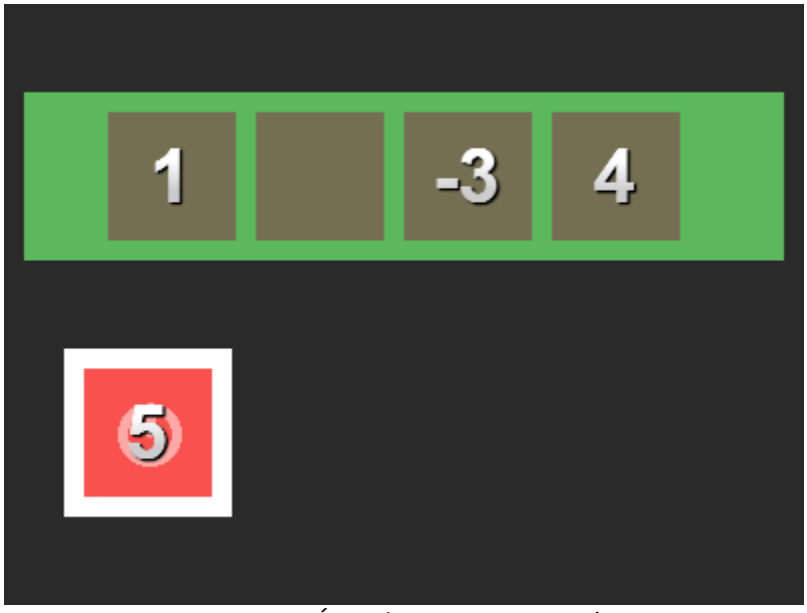

Figura 14. Área de actores en acción.

El resultado es una recta de color verde (haciendo referencia al césped) con cuadrados de color amarronado, contrastando fuertemente con la tonalidad gris de los números. En la figura 14, también puede verse el número cinco en la plataforma, luego de haber accionado el evento de teletransportación.

3.2.7. Segunda y definitiva extensión del componente Número

Hasta este punto, el componente de los números únicamente sabe cómo teletransportarse desde la línea de números hasta el portal. Ahora es necesario implementar la última funcionalidad: el salto a la Recta Numérica. Como ya fue desarrollada para la actividad *Recta Numérica*, lo único que se necesita es copiar el archivo de la funcionalidad desde una actividad a la otra y agregar la construcción de esta en el constructor del número.

En el caso de que muchas actividades utilizaran la recta numérica, el número y estas funcionalidades, lo más correcto sería desarrollar una librería con esos componentes exclusivamente, de forma que una actividad no sólo dependa del core de Códimo, sino que además pueda importar otras librerías que provean un conjunto específico de componentes.

Con los avances tecnológicos en Webpack, actualmente es posible realizar *tree-shaking*, una característica que permite eliminar código que no se utiliza. De esta forma, la inclusión de múltiples librerías de componentes no afectaría al tamaño final de la aplicación en una relación

uno a uno, sino que únicamente los componentes importados estarían incluidos en la versión final.

Al solucionarse el problema del tamaño final de la actividad debido a la importación de múltiples librerías, resulta más factible la realización de ellas.

3.2.8. Implementación de la Recta Numérica

Como este componente no requiere ninguna personalización, lo único que debe hacerse es copiar los archivos desde la actividad *Recta Numérica* a ésta.

3.2.9. Integración de los componentes del Engine

En este punto, todos los componentes del *Engine* ya están desarrollados de forma aislada. Ahora es necesario construir el objeto `engine` que integrará todos los componentes y proveerá la instancia del *Engine* final a la actividad.

El primer paso será instanciar los componentes visuales, verificar que en su integración todo esté bien posicionado y, de ser necesario, corregir los componentes. Luego deben desarrollarse los procesadores de instrucciones que utilizará el *Engine*. Por último, debe generarse una batería de tests visuales que aseguren que el *Engine* funcione según lo esperado.

Durante el desarrollo de los procesadores se analizaron las diferencias con la actividad *Recta Numérica*. La principal diferencia se encuentra en la instrucción para saltar del laberinto a la recta numérica. Mientras que para la primera actividad el hecho de invocar esa instrucción representa la finalización del ejercicio, para ésta no. Por ello, debe separarse la funcionalidad propia de la instrucción —hacer que el número salte a la recta—, de la funcionalidad de verificar si el ejercicio terminó. Afortunadamente se diseñó el constructor de motores gráficos considerando la posibilidad de agregar este tipo de eventos al final del procesamiento de las instrucciones.

Por otra parte, se identificaron puntos de conflicto con otras funcionalidades y procesadores del *Engine* desarrollados hasta el momento. La principal dificultad fue la funcionalidad de posicionamiento. Para las actividades *Recta Numérica* y *Hola Códimo*, esta funcionalidad recibe, entre otras propiedades, las coordenadas de inicio. La razón es que, desde un principio, el número aparece en un lugar específico del laberinto. Sin embargo, en esta nueva actividad es diferente: el número inicia en una recta y termina en otra; por lo que para entrar y salir del laberinto necesita de dos funcionalidades. Esto complejizó su desarrollo ya que el core tiene definida la estructura utilizada por las actividades mencionadas. De esta manera, se refactorizaron las funcionalidades y procesadores relacionados a la funcionalidad de posicionamiento.

Sin embargo, fue posible implementar una solución que no afectara a las actividades ya desarrolladas. Se definió un estado en el actor (en estos casos, los números) que determina si está en movimiento o no: cuando está dentro del laberinto, el estado se activa, y cuando está

fuera, se desactiva. Esto permite verificar con certeza si el número quedó perdido o no dentro del laberinto.

3.2.10. Creación del bloque de acciones para cada número

Luego de realizar la integración entre los componentes del *Engine*, se prosiguió al desarrollo del bloque más importante de esta actividad: aquél que permitirá realizar acciones para distintos números.

Para el desarrollo de este y de todos los bloques se utilizó la herramienta de Google para desarrollo de bloques de Blockly: https://goo.gl/dbwzKe.

El primer enfoque que se pensó utilizar para construir el bloque fue el de una función: un bloque donde se le incrustaran todas las acciones para un número en particular, y luego fuera invocado dentro del bloque principal como una acción más. De esta forma podría introducirse el concepto de funciones. Sin embargo, no fue posible por dos motivos: (1) el desarrollo de procedimientos dentro de Blockly no es sencillo y no hay documentación al respecto; (2) la cantidad de pasos que un estudiante debe realizar para mover más de un número con funciones es grande: primero tiene que definir la función y su implementación, y luego invocarla dentro del proceso (bloque) principal.

El otro enfoque consistió en tratar de resolver el segundo punto de complejidad con las funciones. En vez de llamar a la función dentro del proceso (bloque) principal, resulta más sencillo solo agregarla al *Workspace* para que funcione. Esto significa crear un nuevo tipo de bloque de ejecución, donde se permita tener más de uno. De esta forma, es posible manipular más de un número. Para que fuera posible que dos funciones de ejecución manipulen dos números distintos se le incluyó un campo de tipo selección con el listado de los números a mover.

Una vez desarrollado el bloque, debe adaptarse el core. Los tres elementos que debieron modificarse fueron:

- `BlocklyApp`: esta clase de la interfaz de usuario se encarga de definir los componentes de Blockly. Entre las funciones de ella se encuentra la instanciación por defecto del bloque de ejecución que se puede observar en las actividades *Hola Códimo* y *Recta Numérica*. Para que sea posible utilizar bloques de ejecución personalizados se configuró como opcional el bloque que define esta clase.
- La función de análisis y procesamiento de las instrucciones en formato texto que genera Blockly: a partir del uso de más de un bloque de ejecución se generó un *bug*. Cada bloque de este tipo genera un salto de línea en la cadena de texto de las instrucciones que genera Blockly, por lo que la función de análisis y procesamiento, al no contemplar los saltos de línea, fallaba.
- La función de instanciación de los bloques: hasta esta actividad, no existía un solo bloque que cambiara su estado de forma dinámica según la actividad o el ejercicio. Sin embargo, esto cambia con la incorporación del nuevo bloque de ejecución: el campo de selección

del número a mover varía según el ejercicio, y el único objeto que contiene la información acerca de qué números pueden moverse es el de los metadatos del Engine. Se agregó como nuevo parámetro el objeto de dichos metadatos para la instanciación de los bloques. Este cambio afectó esta función, las clases `BlocklyApp` y `Activity` de la interfaz de usuario.

3.2.11. Edición final: Mensaje de introducción

Para el desarrollo de la actividad *Recta Numérica* no fue necesario construir ningún mensaje introductorio para la consigna de cada ejercicio. Por el contrario, en *Múltiplos y Divisores* sí lo fue. En ella se debe explicar si el ejercicio es sobre múltiplos o sobre divisores, y en relación con qué número se está trabajando. Por ejemplo, "encontrá todos los múltiplos de 3". Así, se modificó la clase `Activity` de manera que cuando se instancie, chequee por la existencia del objeto `exerciseDescription` dentro de la propiedad de los metadatos. Este objeto —que contiene el título, el texto y opcionalmente una imagen o un video—, será utilizado para mostrar por pantalla un mensaje introductorio para el ejercicio.

4. Conclusión del desarrollo

El proceso de desarrollo incluyó dos etapas. La primera implicó un gran esfuerzo en el desarrollo del prototipo de Códimo y abarcó aproximadamente seis meses. La segunda consistió únicamente en el desarrollo de una actividad a través del uso del framework de Códimo. Esto permitió demostrar que el uso del framework facilita la construcción actividades con menor tiempo y esfuerzo que si se intentara construir todo desde cero.

Además del tiempo, permitió la identificación de las librerías como mecanismo para la reutilización y la compartición de componentes. Como se describió en la sección anterior, en el apartado "Segunda y definitiva extensión del componente Número", las librerías harían que el framework de Códimo fuera mucho más conciso y pequeño, como así también permitiría desarrollar y compartir componentes sin necesidad de que deba ser previamente integrado al mismo.

Por otra parte, se constató la inestabilidad del framework de Códimo. Para esta actividad fue necesario reescribir bastantes elementos, lo que demuestra que aún no puede considerarse que Códimo esté cerca de una versión `1.0.0`.

Conclusión y trabajos futuros

El ‘pensamiento computacional’ es un concepto que adquiere cada vez más relevancia dentro de la ‘sociedad del conocimiento’, y el sistema educativo debe tenerlo en cuenta. Sería importante que a corto plazo se lo incluya en el sistema educativo de forma tal que no exija grandes cambios. Con esta idea surgió Códimo, una herramienta que combina actividades de la currícula escolar con el ‘pensamiento computacional’, y que le ofrece al docente la posibilidad de incluirla en su currícula sin necesidad de tener conocimientos en el desarrollo de aplicaciones.

En la construcción de Códimo fue necesario definir un marco teórico, el que facilitó el diseño de una metodología de desarrollo de ‘objetos de aprendizaje’. Aunque para este trabajo se desarrollaron dos actividades, fue posible aplicar en su completitud la metodología de construcción de ‘objetos de aprendizaje’ definida por FLOM. Esto es un logro por dos grandes razones. La primera, es que pone al docente en un rol activo para la construcción de las actividades; la segunda, se refiere a que define un proceso de construcción sencillo, permitiendo su implementación sin demasiado esfuerzo, ya que es similar a las etapas de desarrollo de otros tipos de proyectos de software.

La versión alfa de Códimo está formada por tres componentes: las actividades, la plataforma, y el núcleo del framework. Esta estructura permitió definir con certeza cada responsabilidad —que se diseñan y se programan—, lo que facilitó la construcción de la última actividad “*Múltiplos y Divisores*”. De esta forma, se comprobó la eficiencia del framework de Códimo, que incluye: el tiempo requerido para desarrollarla, el esfuerzo técnico, y la identificación de puntos para corregirlo y hacerlo aún más genérico.

Luego de un año de desarrollo del framework Códimo, se puede plantear como incertidumbre la decisión de utilizar PixiJS como *Engine*. Aunque la herramienta es poderosa y su documentación es muy rica, se comprobó que es demasiado compleja y pesada para las actividades de Códimo.

Por último, y en cuanto a la experiencia llevada a cabo con las escuelas primarias N. º 296 y N. º 355 de Viedma, Río Negro, durante la Semana de la Ciencia y la Tecnología, es posible afirmar que la experiencia en el uso de Códimo por parte de los estudiantes de primaria resultó ser intuitiva y de rápido aprendizaje. A su vez, fue posible observar que el grupo que resolvió los

problemas de a pares le resultó más amena la experiencia que al grupo donde cada estudiante tenía su propia computadora.

A continuación, se plantean lineamientos de trabajos futuros:

- Desarrollar los distintos componentes de Códimo como un conjunto de paquetes que puedan ser instalados a través del repositorio de paquetes de JavaScript: NPM[24] (*Node Package Modules*, Paquetes de Módulos de Node).
- Desarrollar actividades de una materia de humanidades. Esto requerirá que deban desarrollarse un nuevo *Engine* y un nuevo *Workspace*. Esto exigirá una refactorización del framework de Códimo. Con la construcción exitosa de una actividad de una materia de humanidades se espera que Códimo sea realmente útil y pueda integrarse en el nivel primario y secundario.
- Separar los componentes y sus funcionalidades del core a librerías que contengan exclusivamente esos elementos, de manera que el framework de Códimo sea lo más sencillo posible.
- Desarrollar un repositorio de actividades. Cuanto mayor sea la cantidad de actividades que posea Códimo, mayor será la necesidad de construir un repositorio de este tipo, similar al que tiene "La Hora del Código". Para poder desarrollarlo será necesario modificar la plataforma web de Códimo, y requerir un archivo tipo JSON que cumpla con el estándar LOM (*Learning Object Metadata*, metadatos de los objetos de aprendizaje) de manera que muestre una pequeña descripción antes de que sea seleccionada.
- Desarrollar una plataforma que almacene datos de las escuelas, docentes y estudiantes con el fin de desarrollar todo el aspecto evaluativo del marco teórico utilizado en Códimo (FLOM). Además, esto permitirá el seguimiento de los avances por partes de los docentes y estudiantes. También permitirá innovar en otros aspectos, como el desarrollo de un sistema de logros que le informe al estudiante cuando alcance un hito.
- Agregar el aspecto evaluativo a las actividades. Para este trabajo, los objetos de aprendizaje no contienen nada al respecto que permitan confirmar que el estudiante asentó conocimientos, porque para poder llevarlo a cabo es deseable que pueda almacenarse información del estudiante que está realizando la actividad.
- Una vez construido el repositorio de actividades y la plataforma con un seguimiento del estudiante, sería posible desarrollar un sistema de secuenciación de actividades, que permita construir pequeños cursos formados por un conjunto de objetos de aprendizaje.
- Desarrollar un mejor mecanismo de validación de metadatos. Para este trabajo, Códimo valida la estructura de los metadatos de forma parcial. Sin embargo, sería mucho más útil para el proceso de desarrollo de una actividad que Códimo informe con errores más descriptivos qué información de los metadatos de un ejercicio es errónea. Afortunadamente, existen mecanismos que posibilitan esto, como el que posee Webpack, por lo que permitiría simplificar la implementación.

[24] Sitio oficial: https://www.npmjs.com.

Bibliografía

Barbosa, E. y Maldonado, J. (2006a). An integrated content modeling approach for educational modules. *IFIP 19th World Computer Congress – International Conference on Education for the 21st Century*, p. 17–26. Recuperado de: https://goo.gl/Yrn2vs.

Barr, V. y Stephenson C. (2011). Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community?. *ACM Inroads*, 2(1), p. 48-54. Recuperado de: https://goo.gl/r18uFv.

Campo, M. y Price, T. R. (1998). Desarrollo de Frameworks Orientados a Objetos. Universidad Nacional de la Plata. La Plata, Argentina.

Carneiro, R., Toscano, J. C., y Díaz, T. (2009). *Los desafíos de las TIC para el cambio educativo*. Madrid, España: OEI – Fundación Santillana. Recuperado de: https://goo.gl/Qi63DK.

Drucker, P. F. (1967). The effective executive. New York, EU.: Collins. Recuperado de: https://goo.gl/omZugg.

Dussel, I., y Quevedo, L. A. (2010). *Educación y nuevas tecnologías: los desafíos pedagógicos ante el mundo digital*. Buenos Aires, Argentina: Fundación Santillana. Recuperado de: https://goo.gl/Laq2vN.

Graciotto. M. A., Barbosa E. F. y Maldonado J. C. (2011). Model-driven development of learning objects. *Frontiers in Education Conference (FIE)*, F4E–1–F4E–6. Recuperado de: https://goo.gl/45GFmv.

Graziani, L., Cayú, G. A., Sanhueza, M. E. y Molinari, E. (2017). Recta numérica: una actividad de códimo. *XXIII Congreso Argentino de Ciencias de la Computación*, p. 400-407. Recuperado de: https://goo.gl/TNCVAG.

ISTE y CSTA. (2011). Pensamiento computacional: caja de herramientas para líderes. Recuperado de: https://goo.gl/PM8Swh.

L'Allier, James J. (1997) Frame of Reference: NETg's Map to the Products, Their Structure and Core Beliefs [Blog post]. NetG. Recuperado de: https://goo.gl/5uJkLp.

La Hora del Código. (2017). La Hora del Código está aquí. Code.org: Hora del Código. Recuperado de: https://goo.gl/a96kFN.

Lu, J. y Fletcher, G. (2009). Thinking About Computational Thinking. *ACM SIGCSE Bulletin*, 41(1), p. 260-264. Recuperado de: https://goo.gl/mdRwk1.

Papert, S. (1980). Mindstorms: Children, Computers, and Powerful Ideas. New York, USA: Basic Books. Recuperado de: https://goo.gl/hR8awH.

Park, J. C. (2008). Probeware Tools for Science Investigations. *National Science Teachers Association*, p. 33. Recuperado de https://goo.gl/wGsH9p.

Pérez-Lezama, C., Sánchez, J. A., y Starostenko, O. (2012). FLOD: specifying the composition of learning objects using xml schemas for supporting e learning. *International Conference on Education and New Learning Technologies*, p. 3440-3449.

Piscitelli, A. (2009). NATIVOS DIGITALES: *Dieta cognitiva, inteligencia colectiva y arquitectura de la participación*. Buenos Aires, Argentina: Santillana. Recuperado de: https://goo.gl/8fmCrM.

Sánchez, A. J., Pérez-Lezama, C. y Starostenko O. (2015). A Formal Specification for the Collaborative Development of Learning Objects. *Procedia - Social and Behavioral Sciences*. 182, p. 726–731. Recuperado de: https://goo.gl/ac4hRv.

Tedre, M. y Denning, P. (2016). The Long Quest for Computational Thinking. *Proceedings of the 16th Koli Calling Conference on Computing Education Research*, p. 120-129. Recuperado de: https://goo.gl/XRDjWr.

Universidad de Wisconsin. (2017). About Learning Objects. Wisconsin, EU.: Wisc-Online. Recuperado de: https://goo.gl/izcNKW.

Wiley, D. A. (2000). Learning object design and sequencing theory. Tesis doctoral, Brigham Young University. Recuperado de: https://goo.gl/mccpMc.

Wiley, D. A. (2003). Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy. Recuperado de: https://goo.gl/k2oYFA.

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, 49(3), p. 33-35. Recuperado de: https://goo.gl/DhETSa.

Anexos

2017-08-17 Meeting n. º 03

Fecha:	17/08/2017	**Hora:**	11:00	**Duración:**	01:00:00
Título de la Reunión:	Revisión N. º 01 de Códimo por parte de los Docentes				
Lugar:	Escuela N. º 296				

Participantes:	Graziani Luciano, Emanuel Sanhueza, director Ángel Catrin, docente Mirian Schill, docente Mirian Denison.

Ítems en la agenda de la reunión

- ✓ Mostrar la actividad con el fin de obtener feedback.
- ✓ Definir a qué grados va dirigida.
- ✓ Verificar que cumpla con los objetivos de aprendizaje.
- ✓ Analizar de qué manera será evaluada la actividad.
- ✓ Revisar si es necesario agregar teoría como acompañamiento de la actividad.
- ✓ Confirmar fecha y grados que visitarán la universidad.

Decisiones tomadas en la reunión

- ✓ Hay que confirmar que se dispone de dinero para alquilar transporte y el seguro para poder llevar los chicos a la universidad.
- ✓ Revisar de qué manera darles estilo a los bloques del laberinto, ya que los que actúan como pared marean, o desorientan. Buscar *sprites* de césped u otro elemento agradable o que represente un laberinto.
- ✓ Cada baldosa debe estar dibujada, y no formar un camino homogéneo, para que el estudiante pueda contar cada paso sin estar adivinando. Para esto será necesario buscar algún sprite.
- ✓ Los textos de los bloques no ayudan. Es necesario que sean imágenes grandes y de distintos colores. En caso de haber texto, debe ser en mayúsculas.
- ✓ El número debe seguir siendo el mismo en la dificultad fácil y normal.
- ✓ El bloque que se está ejecutando debe resaltarse. Ver documentación de Blockly (https://goo.gl/QMukSp).
- ✓ La dificultad fácil está orientada a estudiantes de primero y segundo grado.
- ✓ La dificultad media está orientada a estudiantes de tercero a quinto grado.
- ✓ La dificultad difícil está orientada a estudiantes de sexto y séptimo grado.

- ✓ Agregar en todas las dificultades que haya más de un hueco disponible en la línea numérica.
- ✓ Agregar más actividades.

Observaciones para futuro

- Incluir una reflexión al final una actividad entera (no un nivel), donde el estudiante deba marcar, de una lista de opciones, qué conceptos cree que aprendió. Las oraciones deben ser sencillas. Se puede agregar alguna opción de un concepto que no se vio, a modo de trampilla.
- Definir ejercicios únicamente con números pares o impares.
- Definir ejercicios con múltiplos y divisores donde también haya *intrusos*, es decir, números que no son divisores ni múltiplos. Para esto sería necesario otra actividad, donde en el fondo haya muchos números, se los deba tomar y luego mover al lugar indicado.
- Agregar al laberinto las operaciones aritméticas como eventos, ya sea a partir de que el número absorbe un poder, pasa a través de un portal o de otra manera (buscar cuál sería lo más adecuado o divertido).

Acciones para la próxima reunión

- El director se responsabilizó en llamarnos para confirmar la reunión de la semana que viene (martes que viene, si no llama, llamar nosotros).
- Realizar la mayor cantidad de cambios requeridos (según prioridad).
- Confirmar información para el transporte.
- ¿Entregar panfletos de la semana de la CyT?
- Confirmación por parte de los docentes sobre qué grados van a ir a la visita.

2017-08-25 Meeting n. º 04

Fecha:	25/08/2017	**Hora:**	10:00	**Duración:**	01:30:00
Título de la Reunión:	Revisión N. º 02 de Códimo por parte de los Docentes				
Lugar:	Escuela N. º 296				

Participantes:	Graziani Luciano, Emanuel Sanhueza, director Ángel Catrin, docente Mirian Schill, docente Mirian Denison.

Ítems en la agenda de la reunión

- ✓ Mostrar la actividad con el fin de obtener feedback.
- ✓ Definir a qué grados va dirigida.
- ✓ Confirmar fecha y grados que visitarán la universidad.

Decisiones tomadas en la reunión

- ✓ El martes 5 de septiembre los estudiantes de 3er grado van a la universidad.
- ✓ Remarcar las barras de dirección.
- ✓ Marcar el camino como casilleros.
- × Escribir los mensajes de manera más amigable.
- × Poner títulos a los ejercicios.
- ✓ Poner iconos en movimiento.
- ✓ Poner en el menú sólo el primer ejercicio y que pueda avanzarse cuando se complete cada uno de ellos.
- × Hacer que los bloques sean más chillones cuando hay un error.
- ✓ Mouse más grande.

Observaciones

- × Hacer que los bloques sean más chillones cuando hay un error.
  No es posible realizarse ahora porque requiere de un gran esfuerzo de modificación de Blockly.