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

WoBlocks: programación orientada a objetos con bloques

Tesis de Licenciatura en Ciencias de la Computación

Alejandro Ferrante

Director: Matías López y Rosenfeld
Codirector: Alfredo Sanzo
Colaborador: Nahuel Palumbo
Buenos Aires, 2022

WoBlocks

La expansión tecnológica de las últimas décadas, junto con el aumento de su potencia y alcance han transformado el mundo. La penetración de las computadoras en cada vez más ámbitos y tareas transformó la necesidad de aprender a utilizarlas en algo indispensable. En ese sentido, entender qué sucede dentro de las computadoras nos permitirá interactuar de mejor manera con este mundo. En este trabajo nos centramos en el aprendizaje de la programación. Esta es una tarea difícil en diferentes niveles, y hemos tomado en consideración las propuestas existentes para formular una propia que constituya un avance en el área.

La presente propuesta se materializa en el desarrollo de una herramienta educativa para la enseñanza de la programación basada en bloques bajo el paradigma de la programación orientada a objetos a través del desarrollo de videojuegos para nivel universitario: **WoBlocks** La propuesta se basa en la experiencia y didáctica del lenguaje Wollok. Buscando tener una cantidad minimal de bloques donde cada uno ilustre un concepto fundamental del paradigma. Se reportan la evolución de la herramienta en sus versiones preliminares hasta llegar al resultado final. Se realizó una prueba piloto dónde los usuarios lograron resolver el desafío propuesto y reportaron su experiencia como usuarios. Por último, se incluyen reflexiones y desafíos futuros a abordar para que **WoBlocks** siga creciendo y pueda ser utilizado en un contexto áulico.

Palabras claves: Programación en Bloques, Programación orientada a objetos, Visualización de Programas, Educación, Wollok, Herramienta Pedagógicaa

WoBlocks

The technological expansion of the last decades, together with the increase in its potency and reach, has transformed the world.

Computers have permeated and are still permeating multiple aspects of our daily lives, and thus the need to learn to use them is more important than ever. Aside from learning its use from a user's point of view, a deeper understanding of the internal workings and fundamentals will allow a better interaction with them. This work is about learning to program. This is by no means an easy task for several reasons, and that is why we have taken into consideration the existing approaches and tools to formulate our own one, so that it represents a step forward in this area.

We set up to assemble an educational video game building tool designed to teach the fundamental concepts of object-oriented programming using a block-based visual language, targeted at university level students. We called it **WoBlocks**. Our approach is grounded on the experience and the educational decisions made by the developers of the Wollok language. One of our main goals is to have a minimal number of blocks available, and that each one represents a single paradigm's key concept we want to teach. We describe the evolution of the tool we built up to its final preliminary form. Additionally, we performed pilot test runs where users were able to solve a fairly complex programming challenge and reported their experience as users of this tool. Finally, we include the corresponding discussions and future work needed for **WoBlocks** to continue growing and be effectively used in classrooms.

Keywords: Block Based Programming, Object Oriented Programming, Program Visualization, Education, Wollok, Educational Tool

AGRADECIMIENTOS

A mi familia, amigos y pareja. A los directores, colaboradores de este proyecto y al sistema educativo argentino.

A quienes se vean náufragos en la hercúlea tarea de completar una carrera, o cualquier empresa que parezca inalcanzable por momentos, que este trabajo sirva de prueba que la perseverancia mueve montañas.

Índice general

1. INTRODUCCIÓN

1.1. Contexto general

Tomando en consideración las tendencias globales en cuanto a la expansión tecnológica [1, 2, 3, 4, 5, 6], podemos destacar que una de las características más significativas de las últimas décadas (y de este principio de siglo XXI), es el hecho que la tecnología digital se ha ido expandiendo hasta abarcar cada vez más aspectos de nuestra vida cotidiana. Desde el ámbito profesional hasta el hogareño, nos hemos acostumbrado a interactuar constantemente con distintas formas de computadoras donde antes lo hacíamos con dispositivos analógicos. Hemos presenciado desde el comienzo del nuevo milenio una explosión en la cantidad de usuarios, capacidad, accesibilidad y velocidad de la internet, así como de los dispositivos móviles, entre otras innovaciones.

Adicionalmente, el concepto de *The Internet of Things* plantea el mandato de que las computadoras con las que interactuamos deben estar conectadas entre sí [7]. Nuestro país no escapa a esta tendencia mundial. Encuestas del año 2019 revelan que 8 de cada 10 habitantes tienen acceso de alguna u otra forma a la internet [6].

A diferencia de generaciones anteriores, las pertenecientes a este siglo han sido bautizadas con el adjetivo de nativas digitales [8]. Esto significa que desde muy temprana edad estos individuos son expuestos al uso de distintos dispositivos digitales y, al pertenecer desde el comienzo de sus vidas a un mundo hiper conectado, poseen un gran nivel de acostumbramiento y destreza, y están adaptados a tecnologías que ninguna de las generaciones previas maneja nativamente. Sin embargo, podemos argumentar que esta experiencia nunca escapa al rol de usuario de las herramientas digitales; y que tener el conocimiento resultante de la experiencia de uso de una herramienta no es equivalente a tener el entendimiento de cómo esta es diseñada o construida, ni las consecuencias de su uso.

Diversos autores [9, 10, 11, 12, 13, 14, 15] se inclinan a destacar la importancia de llevar la programación a la curricula obligatoria desde los niveles más tempranos posibles para poder dotar a los alumnos con un entendimiento superior y presentarles las bases sobre las cuales podrían continuar formándose en el área.

Dicho esto, sabemos, basándonos en trabajos de quienes se han abocado este desafío [9, 16, 17], que esta necesidad plantea un desafío descomunal para todo el campo académico. Todos ellos coinciden en que enseñar programación no es una tarea fácil y es algo que nunca se ha hecho a gran escala. Creemos que experiencia y análisis de las distintas propuestas, así como la iteración y el perfeccionamiento de las mismas son requeridos para alcanzar a cubrir las necesidades actuales. Si bien se están haciendo avances en el área, sabemos que estas no siempre son suficientes. Adherimos a la opinión de autores como Lucas Spigariol [11] o Sukirman [9] entre otros de que necesitamos herramientas pedagógicas efectivas, diseñadas específicamente para este fin, así como una propuesta didáctica y metodologías adecuadas para potenciar su efectividad.

1.2. ¿Por qué es importante enseñar programación?

La bibliografía sugiere que las personas con conocimientos de programación suman diversas y útiles habilidades. La más obvia es la capacidad de diseñar, implementar y mantener sistemas informáticos y aplicaciones.

Por más que no se fuera a dedicar a ello o profundizar su conocimiento, podemos suponer que un entendimiento básico de lo que es una pieza de software hará que pueda interactuar más efectivamente con ellas. En el ámbito profesional actual, muchas áreas se apoyan cada vez más en herramientas de software para sus tareas diarias. Esto puede ir desde llenar una base de datos, alimentar una inteligencia artificial, definir los datos a ser analizados o recabados por un programa como sucede en data mining, u otras instancias donde el usuario no tiene que programar el sistema, pero este permite un alto nivel de configuración, por lo que quien posea un buen entendimiento del funcionamiento del mismo logrará sacarle un mayor provecho.

Por último, tenemos por supuesto la habilidad general de resolución de problemas mediante pasos lógicos [18, 19, 9].

1.3. ¿Con qué paradigma de programación se enseña hoy en día?

Uno de los mayores debates asociados a la enseñanza de la programación en el campo académico es el enfoque que se le debe dar [20, 21]. ¿Cómo se le presenta la programación a alguien que no tiene experiencia alguna? Cada paradigma existente tiene una respuesta diferente.

Algunos creen que por su formación en matemáticas, los alumnos están familiarizados con el concepto de funciones, y por tanto pueden empezar reescribiendo algunas que ya conocen y les son familiares utilizando una sintaxis especifica (un lenguaje de programación) como primer aproximación (programación funcional) [22].

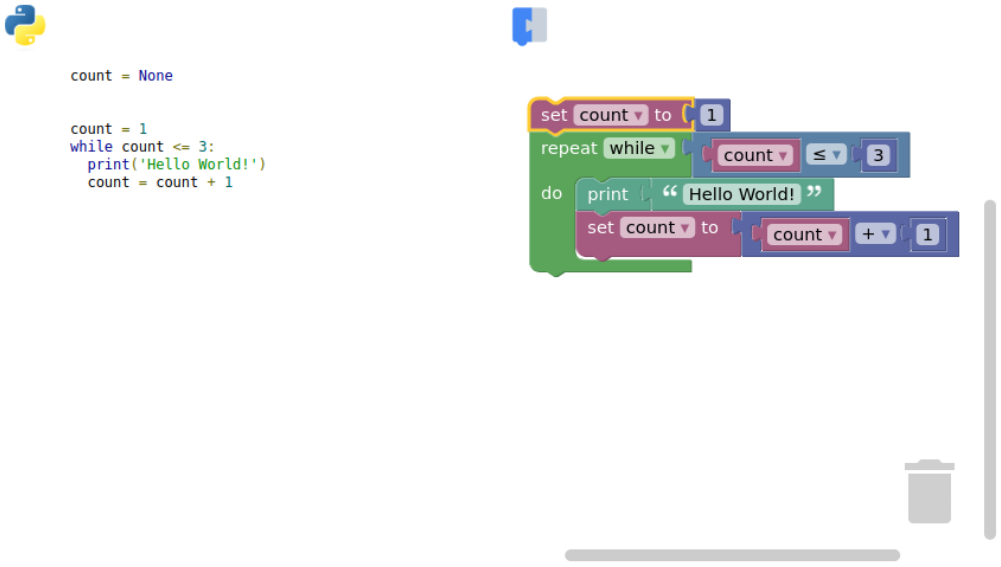
Otros presentan a la programación como instrucciones a ser dadas a una computadora en un orden especifico, y rápidamente presentan estructuras de datos básicas y se dedican a mostrar como manipularlas y recorrerlas para lograr resultados deseados (programación procedural)[23].

Los partidarios de la Programación Orientada a Objetos (POO), al igual que los dos anteriores, tienen su propia forma de interpretar y modelar problemas. Formulan preguntas como ¿Qué atributos deberían tener? ¿Cómo deberían irse modificando? ¿Quién guarda qué responsabilidad? ¿Cómo deben intercomunicarse para garantizar el comportamiento deseado? [24, 10, 16, 25]

Otras posturas son más integradoras y abogan por un enfoque que incorpore múltiples paradigmas [26].

No podemos afirmar que un enfoque sea peor o mejor que otro. La mayoría concuerda con que en cierta medida estas tres formas de entender la programación deben ser al menos presentadas en algún momento, pero se difiere en el cuándo y en cuál debería ser la primera. Diversos autores [21, 20, 27, 28] identifican también el problema del cambio de paradigma, que se da cuando se presenta una forma de pensar la programación y luego se debe introducir otra.

Otra importante decisión a tener en cuenta es, dentro del enfoque elegido, el lenguaje de programación que se debe utilizar para enseñar. Cada lenguaje tiene su particularidad y sus dificultades, como también lo tiene el entorno de desarrollo que se utiliza para



(a) Python *(b)* Google Blockly

Fig. 1.1: Comparación entre un lenguaje textual (a) y uno de bloques (b) para un mismo. algoritmo

codificar. Si bien hay quienes creen que se pueden presentar y utilizar un entorno de desarrollo integrado (*Integrated Development Environment*) (IDE) profesionales para esta tarea [29, 30], adherimos a las posturas que sostienen que siempre es mejor, sobre todo para niveles iniciales, utilizar lenguajes y/o entornos diseñados específicamente con fines didácticos que tomen en cuenta el tipo de usuario por el cual serán utilizados [31, 11, 10, 32, 33].

1.4. Cuando el texto no es suficiente

Es preciso en este punto definir qué son los lenguajes de programación textuales. Estos son aquellos donde la forma de construir programas es a través de diferentes archivos de texto con un formato y sintaxis determinado. La figura 1.1a[a] muestra un ejemplo de código escrito en Python, uno de los tantos lenguajes textuales. En contraposición, existen otros llamados lenguajes visuales que utilizan metáforas gráficas para definir el comportamiento de los programas, también ilustrado por la figura 1.1b[b].

Diversos investigadores especializados en la enseñanza de la programación [9, 31, 25, 34, 35, 10, 16, 11, 36, 32, 15] han concluido que los lenguajes textuales no son ideales para las primeras experiencias de los alumnos.

Atribuyen esto a que el tener que escribir las sentencias acarrea una familia de posibles errores y una curva de aprendizaje completamente ajenas al hecho de pensar lo que el programa debe hacer. En los primeros momentos, el alumno debe estar lo más enfocado posible en determinar qué hacer, y no en cómo escribirlo [37]. Estos autores sostienen que los errores de sintaxis son muchas veces tediosos, frustrantes y desmotivadores, además que el cometerlos y corregirlos no le brinda al alumno ningún tipo de valor educativo. Entender que el orden de ejecución de dos sentencias producen un resultado distinto, o que una sentencia difiere en efecto de otra es un descubrimiento que ayuda al entendimiento

de la naturaleza de los programas. Saber que en un lenguaje la palabra "String" debe escribirse con mayúscula y no será reconocida si se escribe en minúscula no aporta a la comprensión de programación y este detalle particular de un lenguaje podría no aplicar a otro. En el caso de los bloques, tenemos que estos están diseñados para ser intuitivamente fáciles de manipular, además de poseer la característica de que un bloque solo puede ser encastrado donde corresponde, haciendo que una composición de bloques siempre genere código bien formado.

1.5. ¿Qué se busca en un lenguaje o entorno con fines educativos?

Basándonos en distintos trabajos que proponen herramientas educativas para la enseñanza de la programación [24, 38, 39], observamos que los autores priorizan las siguientes características a la hora de elegir o diseñar un ambiente de desarrollo propicio para alumnos sin conocimiento previo:

- Baja complejidad y simplicidad.
- Facilidad de uso.
- Que sean visuales y/o tengan representación visuales de los conceptos a enseñar.
- Interactividad.
- Que posean animaciones.
- Que sean gratuitos.
- Que sean fáciles de configurar y comenzar a utilizar.
- Que utilicen poco o ningún texto.
- Desaconsejan IDEs profesionales.

La figura 1.2 muestra un ejemplo de un lenguaje visual de bloques llamado UNCDuino [40]. Esta composición de bloques ejemplifica muchos de los puntos descriptos anteriormente: muestra (visualmente y sin utilizar texto) una ejecución en secuencia que incluye una repetición. El encastre de bloques resulta intuitivo e interactivo. El entorno donde se utiliza este lenguaje es gratuito y lejos de parecerse a un entorno de desarrollo integrado (*Integrated Development Environment*) (IDE) profesional, se accede a través de un navegador y no requiere configuración.

1.6. ¿Qué propuestas existen hoy en día?

La idea de lenguajes diseñados con fines educativos apuntados a conceptos específicos no es nueva ni exclusiva de lenguajes visuales. Podemos citar el ejemplo de Pascal, un lenguaje textual (desarrollado a principios de los años 70s por el científico suizo Niklaus Wirth) originalmente creado para enseñar conceptos de la programación procedural, o un ejemplo mas reciente de producción nacional como lo es Gobstones [41], un lenguaje en sus inicios textual que también ilustra conceptos de programación procedural, con foco en la representación de información y división en subtareas. Otro ejemplo de producción nacional es el de Wollok [42, 43, 44], un lenguaje textual cuyo objetivo es enseñar conceptos

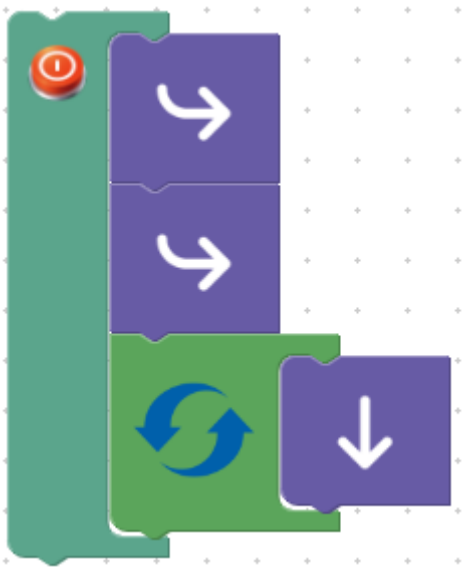


Fig. 1.2: UNC++Duino [40]: Un ejemplo de lenguaje educativo para enseñar programación procedural.

de POO y que al igual que Gobstones fue creado desde la necesidad de poseer herramientas didácticas para enseñar conceptos específicos.

Algunos educadores, sobre todo los que apuntan a públicos universitarios, son menos reacios a la utilización de ambientes profesionales y por tanto optan por utilizar IDEs que posean extensiones que sirvan para fines didácticos como lo son aquellos que incorporan representaciones gráficas y animaciones. Un ejemplo de esto es *BlueJ* [45], que si bien sigue estando basado en texto, utiliza un enfoque más gráfico y amigable para presentar la definición de clases en el lenguaje Java.

Propuestas más controversiales plantean al UML como alternativa para definir objetos y métodos a través de diagramas que utilicen este lenguaje pero no hallamos experiencias de su implementación reportadas [46].

Otra propuesta apuntada principalmente a públicos jóvenes, que lentamente va ganando relevancia entre quienes estudian la didáctica en programación: los bloques [9]. Esta alternativa plantea el uso de coloridos elementos arrastrables y configurables que se encastran entre sí para crear representaciones gráficas de distintos conceptos, típicamente fragmentos de código. Entre los más populares en la actualidad se encuentra *Scratch* (desarrollado por el MIT Media Lab en el 2003), pero distintas variantes se abren camino aprovechando la creciente popularidad de estos lenguajes. Un caso destacable es el de *Pilas Bloques* [33], una creación local que se encuentra en igualdad de condiciones a las propuestas mas utilizadas en cuanto a sus capacidades y calidad visual.

Si hablamos de casos de éxito, las estrategias pedagógicas que más han entusiasmado a los alumnos son aquellas que saben combinar elementos como las animaciones y la interactividad.

Propuestas como *Karel the Robot* [47], *Robocode* [48], *Alice* [49], *Code Combat* [50], *Greenfoot* [51] o los ya mencionados *Scatch* [52] y *Pilas Bloques* [33] han comprendido el poder de los juegos para enseñar y afianzar conceptos, al tiempo de generar un producto

final atractivo en audiencias de todas las edades.

1.7. ¿Por qué estas propuestas no son suficientes?

Ante la existencia de diversidad de propuestas, creemos que cada una tiene sus ventajas y desventajas, y el hecho de que cada una cuente con una especificidad propia del objetivo para el cual ha sido pensada hace que exista un área inexplorada que es importante vislumbrar.

Según nuestra visión, las opciones basadas en ambientes profesionales siguen siendo demasiado complejas y poco amigables para alumnos principiantes. Las extensiones existentes, por más que se propongan remediar esto, no han logrado del todo cambiar el hecho de que la herramienta no fue diseñada con fines pedagógicos.

Además, la amplia mayoría de las herramientas fueron pensadas exclusivamente para el aprendizaje de la programación procedural.

En cuanto a los lenguajes de bloques que existen hoy en día, estos suelen estar destinadas a públicos infantiles. Las propuestas didácticas pueden agruparse en dos grupos: Aquellas que proponen desafíos cerrados con niveles incrementales en su dificultad (como Pilas Bloques, o Code.org), y aquellas donde se da completa libertad (Como Scratch). En las primeras, se presenta para cada actividad una cantidad reducida de bloques que sirven para resolver el problema planteado sin distracciones y con foco en determinados conceptos, pero esos bloques son demasiado específicos y tienen un alcance limitado (por ejemplo, el bloque "Comer manzana"). En las segundas, se ofrecen grandes cantidades de bloques para resolver muchos problemas de animación y juegos, lo cual otorga un alcance y una potencia expresiva mucho mayor, a costa de generar distracciones y diluir la importancia de cada concepto en el amplio abanico de opciones disponibles [33].

Haciendo un repaso por las propuestas en programación en videojuegos, tenemos algunas sumamente complejas como lo es Alice cuya complejidad deriva del gran poder que posee: conforma un motor para la creación de videojuegos tridimensionales, la situación y movimiento en 3 dimensiones de actores y cámara, la ejecución concurrente, etc., mientras que otras poseen las mismas desventajas puntualizadas para los lenguajes de bloques apuntados a alumnos infantiles.

Por último, la producción de didácticas y herramientas asociadas tiene un contexto geográfico, social y político específico. Cada una de estas herramientas fueron creadas desde necesidades particulares para un público particular. Por ejemplo, Alice fue creada por la Universidad de Carnegie Mellon, y fue pensada para enseñar nociones introductorias de programación en Java a ingresantes de esa universidad, una universidad privada del este de Estados Unidos [49, 53], y no es ideal si se intenta enseñar por ejemplo *polimorfismo* a *adolescentes* en una *escuela pública* de *Argentina*.

La elección de pedagogía y didáctica es entonces a nuestro entender también una discusión de **soberanía**, y esta tesis también se elabora desde esa concepción. Como mencionamos anteriormente, en Argentina actualmente existen distintas líneas de trabajo en didáctica con extenso recorrido, como las desarrolladas para enseñar en toda la escolaridad por la iniciativa Program.AR (siendo Pilas Bloques parte de ellas), como las desarrolladas para los cursos introductorios de programación para la Universidad Nacional de Quilmes (siendo Gobstones parte de ellas), o como las desarrolladas por docentes de diferentes universidades nacionales para enseñar POO (siendo Wollok parte de ellas).

Sin embargo, Wollok [42], que es argentino, está pensado para enseñar Programación

Orientada a Objetos y permite crear videojuegos de forma amigable, es textual, y trabaja sobre un IDE industrial.

1.8. Nuestra propuesta

Esta tesis reporta el diseño, implementación y prueba de **WoBlocks**, una nueva herramienta para la enseñanza de la programación desde el paradigma POO utilizando bloques. Este trabajo se basó en el lenguaje Wollok –y su propuesta didáctica– y el entorno con bloques denominado Blockly [54].

A continuación presentamos, en el capítulo 2 un análisis más exhaustivo de Wollok y Blockly, describiendo la herramienta que proponemos en sus distintas versiones, detallando para cada iteración el desarrollo, diseño y aspectos didácticos relevantes. Luego en el capítulo Resultados se reporta la prueba piloto de la herramienta en la que se invitó a docentes a resolver un ejercicio con **WoBlocks**. Por último, se presenta una discusión, conclusiones y trabajo a futuro.

2. DESARROLLO

2.1. Objetivos y aspectos generales

Esta tesis tiene como objetivo la construcción de un entorno de aprendizaje de la programación desde el paradigma de Programación Orientada a Objetos (POO) utilizando bloques como forma visual de presentar la construcción de programas.

Esta propuesta no pretende ser pensada desde cero, sino que tiene como objetivo basarse en otras afines a POO y con especial atención en las desventajas que se encontraron en desarrollos previos mencionadas en la introducción de este trabajo –como pueden ser la utilización de IDEs profesionales o la falta de generalidad, apuntando a evitarlas o minimizarlas. Creemos que muchas de estas pueden ser abordadas con la materialización de un entorno de desarrollo que las tenga en cuenta a la hora de su construcción.

2.1.1. Características deseadas de la herramienta

Ya hemos mencionado anteriormente algunas alternativas a un IDE profesional con lenguaje escrito. De todas ellas destacamos como fundamentales para nuestra propuesta que la herramienta proponga un lenguaje visual. Esto significa minimizar la distancia entre el modelo mental que un estudiante forma en su mente y el modelo concreto que manipula, facilitándole el proceso de plasmar una idea en el ambiente y eliminando todas las trabas, frustraciones y curva de aprendizaje asociadas a la sintaxis de un lenguaje escrito. Como mencionamos anteriormente, además de visual, la herramienta debe ser interactiva y debe garantizar un feedback inmediato y claro cada vez que se realiza una acción.

Además de esto, consideramos su uso debe ser lo más intuitivo y sencillo posible. Es sabido que este aspecto en todos los casos facilita el proceso de aprendizaje y de entendimiento del ambiente que se está utilizando.

Dos características que creemos fundamentales para lograr esto es tener una cantidad mínima de bloques, maximizando el poder de los mismos. A su vez, estos deben contar con la menor cantidad de texto posible. El fundamento para la primera característica es que buscamos crear una correspondencia entre un concepto que queramos que la herramienta enseñe y un bloque que lo ilustre. En el caso de la segunda característica, buscamos crear metáforas visuales de fácil entendimiento para los usuarios. Así, formaremos una fuerte identificación entre un bloque claramente distinguible con un concepto fundamental de la POO.

2.1.2. Objetivos Didácticos

En cuanto a nuestros principales objetivos didácticos, en primer lugar queremos poder enseñar los conceptos clave de POO a audiencias que carezcan de cualquier formación en el área, es decir, que puedan aprender desde cero. En este sentido, apuntamos principalmente a jóvenes universitarios o de nivel secundario, aunque no descartamos que dependiendo de las capacidades de la herramienta esta pueda serle amigable también a un público más joven.

Finalmente, si bien como ya comentamos los lenguajes escritos no son idóneos sobre todo para un principiante, reconocemos que por la situación actual de la industria y en

parte la academia, tarde o temprano un programador tendrá que toparse con ellos. Es por esto que otra característica deseable es que la herramienta facilite la transición hacia el lenguaje escrito. De esta manera además de enseñar los conceptos de POO propuestos, el alumno podrá ver como aquello que ve plasmado en el ambiente se traducirá a un lenguaje escrito. Así la herramienta serviría para cimentar el puente entre el modelo de objetos que se está expresando y la traducción escrita del mismo en un lenguaje específico [53].

2.1.3. Conceptos de objetos que se quieren enseñar

Es necesario destacar aquí también cuáles son los conceptos de POO que consideramos que un alumno debería terminar manejando, aunque sea en un nivel básico, como resultado del aprendizaje a través de nuestra herramienta: conceptos de objeto, método y mensaje, atributos, referencias y estado interno, y polimorfismo. Qué es y como funciona el envío de mensajes, la diferencia entre método y mensaje, estado interno visto como conjunto de referencias, y el polimorfismo como herramienta fundamental para el modelado de soluciones flexibles.

Si bien el concepto de clase e instancia son importantes, decidimos no incluirla al menos en esta primera versión. Siguiendo la propuesta didáctica de Wollok [43, 44], esta propone hacer mucha ejercitación con los conceptos arriba mencionados antes de introducir el concepto de clase.

Cabe destacar que diversos proyectos didácticos en POO [39, 24, 11, 34, 36] también consideran muchos de estos conceptos como los esenciales.

2.2. WoBlocks

Tomando en consideración todos los objetivos previamente discutidos, decidimos darle un nombre a nuestra propuesta: **WoBlocks**.

El nombre proviene de la conjunción de las dos tecnologías principales que elegimos para implementar nuestra herramienta: *Wollok* y *Blockly*.

2.2.1. Blockly

Blockly [54] es una biblioteca desarrollada en Javascript por Google que incorpora un espacio donde se pueden manipular, ensamblar y combinar bloques dentro de una página web. Nativamente provee un toolbox que se divide en categorías desde las cuales se pueden seleccionar los bloques a arrastrará al workspace, que es donde viven los bloques activos. Cada bloque representa (según la concepción de Blockly) un fragmento de código que puede ser configurado y encastrado con otros bloques para generar el código deseado. Este puede luego ser visualizado y ejecutado.

Blockly comenzó como un proyecto de una sola persona en 2011, inspirado en un proyecto similar del MIT que utilizaba el lenguaje Java llamado Open Blocks. Para 2012 ya tenía una versión preliminar donde se podían crear y combinar bloques. A partir de su lanzamiento en aquel año lleva una década de ser utilizado para distintos proyectos en variedad de áreas. En concreto, Blockly constituye una biblioteca que permite crear un lenguaje visual e interactivo, por lo cual lo consideramos de gran utilidad como base de nuestro proyecto. Blockly aporta simplicidad, facilidad de uso, baja complejidad, interactividad y representaciones visuales a nuestra propuesta.

2.2.2. Wollok

La otra mitad constitutiva elegida de nuestro proyecto fue Wollok [42]. El ya mencionado lenguaje de programación es el resultado de un proyecto colaborativo de código abierto alojado actualmente en GitHub [55]. Sus primeras versiones surgieron a partir de docentes que trabajaban enseñando POO en diversas universidades y que luego formaron parte de la Fundación Uqbar [56]. Los objetivos pedagógicos eran:

- Aplanar la curva de entrada a un lenguaje Orientado a Objetos para alumnos en su segunda (y alguna veces primera) experiencia con lenguajes de programación en la universidad.
- Permitir practicar los conceptos fundamentales de objeto, método y mensaje desde la primer clase.
- Tener una visualización gráfica del estado del sistema (un diagrama automático de objetos)
- Esconder en la interfaz del IDE numerosas opciones destinadas a la programación profesional que generaban confusión en novatos.
- Permitir avanzar y practicar los conceptos fundamentales en las primeras clases sin salir del entorno cuidado (polimorfismo, estado interno, colecciones)

Todo comenzó el año 2008 cuando se creó una extensión de Dolphin Smalltalk llamada Object Browser que permitía crear objetos sin clase, y así proveer una herramienta para poder enseñar utilizando la didáctica desarrollada. Esta fue incorporada rápidamente a la enseñanza por su practicidad [57]. Hacia 2009, tras la deprecación de Dolphin Smalltalk, la herramienta fue migrada a Pharo Smalltalk, una tarea para nada sencilla [58] realizada nuevamente por docentes de la Fundación Uqbar. Luego esta nueva versión fue rebautizada como Ozono.

Esta herramienta fue utilizada en el ámbito educativo para las clases de POO. Durante este periodo los docentes fueron obteniendo experiencia y pudieron determinar puntos de mejora para futuras versiones. Estas mejoras necesarias tenían naturaleza pedagógica y técnica:

- Ozono bajaba la curva de entrada, pero producía dos barreras "de salida". Primero, el cambio de Ozono a Smalltalk puro (de objetos a clases) era traumático. Segundo, Smalltalk era un lenguaje muy poderoso para enseñar y para hacer investigación, pero estaba basado en imagen en lugar de archivos, no tenía una clara separación entre la parte estática y la dinámica -de hecho las clases eran a su vez objetos-, su IDE era diferente al de la mayoría de los lenguajes industriales, al igual que su sintaxis. Entonces, la "salida" de Smalltalk a otros lenguajes de objetos basados en imagen también era poco sencilla.
- Por más que la herramienta ayudara con las pautas didácticas, el lenguaje seguía siendo Smalltalk, pensado desde su IDE y framework íntegramente para el uso profesional en industria o investigación. Ozono no podía escapar de ello.

Fue entonces en el año 2014 que se decidió que Ozono debía trascender el ambiente Smalltalk y se decidió utilizar el lenguaje XText para crear una versión que corriese sobre el entorno de desarrollo Eclipse, incorporando todos los puntos de mejora obtenidos en los períodos anteriores. Esa tarea fue construir un lenguaje de programación desde cero, y entonces también tuvo un cambio de nombre, rebautizándose Wollok.

Tener el poder de desarrollar un lenguaje de programación permitió tomar muchas decisiones finas sobre la didáctica. Algunas de ellas fueron:

- Usar un IDE industrial basado en archivos y con clara separación entre momento estático (definir objetos y clases) y momento dinámico (ejecutar).
- Usar sintaxis similar a C como muchos lenguajes actuales (Javascript, Java, C#).
- Usar palabras clave fuertemente asociadas a los conceptos importantes: `program`, `object`, `method`.
- Proveer chequeos automáticos, mensajes de error, advertencias y ayudas conceptuales orientadas no a la programación profesional sino al aprendizaje inicial.
- Usar un IDE que proveyera todas esas ayudas en idioma español.

Inmediatamente luego de creado el lenguaje una tesis dotó a Wollok con los elementos necesarios para programar y ejecutar videojuegos. Fue así que nació *Wollok Game*. Con él se puede seguir enseñando de manera efectiva todos los conceptos de POO inicialmente propuestos, pero con la ventaja de que lo que se está programando es algo atractivo, interesante y divertido para cualquier alumno como lo es un videojuego [59]. Esto hace que no solo el alumno esté comprometido en el proceso de creación y aprendizaje, sino que como producto de este proceso tendrá un juego creado por él, el cual probablemente querrá seguir desarrollando, agregándole features o niveles y por tanto queriendo aprender más y ganando mayor experiencia en la programación. Estos resultados fueron y siguen siendo constatados en las encuestas de fin de año cuando se les pregunta a los alumnos sobre Wollok Game.

El desarrollo más reciente y todavía en progreso es la migración del lenguaje base de implementación de XText y Eclipse a una tecnología web como Typescript y una adaptación a un IDE más moderno como Visual Studio Code [60].

WoBlocks constituye una continuación de la serie de decisiones puntualizadas, sumado a lo especificado en la sección 1.5, y toma como aprendizaje toda la experiencia previa en el desarrollo de Wollok.

2.2.3. Features Principales

Habiendo definido la utilización de un lenguaje visual (más específicamente, un lenguaje de bloques) como lo es Blockly, que se traducirá a un lenguaje de objetos (Wollok), nos es preciso puntualizar concretamente lo que nuestra herramienta permitirá hacer al usuario.

Nuestra propuesta inicial, previa a la introducción de Wollok, fue en primer lugar que la herramienta permita combinar bloques para definir los conceptos de la POO que queremos que el alumno aprenda. Esto es, que un armado de bloques sirva para definir un Objeto, un método y el envío de un Mensaje. En su momento, consideramos que estos tres son

los elementos fundamentales con los cuales un alumno se debe familiarizar desde el primer momento, y son los que debe aprender a crear y combinar para resolver un problema.

Estos objetivos de la propuesta inicial se ven reflejados en la primera versión de la aplicación. Luego de los intercambios con el equipo de Wollok, se afinaron los objetivos de acuerdo al modelo de la propuesta pedagógica detrás del lenguaje Wollok [44], donde los comportamientos son en realidad métodos definidos en los propios objetos. Esto se ve reflejado en las versiones subsiguientes de la aplicación.

Es entonces en función de los objetos que se debe pensar y plantear dicha resolución. No se debe pensar "¿Qué función resuelve este problema?" o "¿Cuál es la sucesión de instrucciones que deben darse para llegar de un input a un output?", sino que debe primero poder entender y modelar el dominio del problema con objetos, respondiendo preguntas como "¿Quién tiene la responsabilidad de resolver este problema?" y "¿qué mensajes deben enviarse?".

Lo siguiente que la herramienta debe poder hacer es mostrar claramente el resultado de la ejecución del ambiente plasmado en ella. Este debe ser claro, puesto que el alumno debe poder entender cuál es el resultado y por qué se da el mismo. Dicha comprensión le servirá para entender sus errores cuando el resultado de la ejecución no refleje el comportamiento esperado.

Además de que el alumno pueda entender su error, la herramienta debe facilitarle el poder modificar el ambiente y volver a ejecutar para tener así un feedback inmediato que compruebe sus suposiciones de resolución de un error. El feedback claro e inmediato puede ser de gran ayuda en el proceso de aprendizaje.

Como ya hemos mencionado, los usuarios de la herramienta estarán manipulando objetos la mayor parte del tiempo y querrán que ellos intercambien mensajes. Es por esto que la herramienta deberá poder indicarles clara y fácilmente cuáles son los mensajes que cada objeto puede responder.

Por último, teniendo en cuenta la transición hacia un lenguaje textual, la herramienta deberá tener la capacidad de mostrar el código Wollok generado. Remarcamos que esto lejos está de ser el objetivo primario de nuestra propuesta. Queremos que el alumno piense en término de objetos y que entienda que una composición de bloques es partes de una definición (ya sea de objeto, método o envío) y no lo interprete como una porción de código.

Estas traducciones, por tanto, podría permanecer ocultas en los primeros momentos del proceso educativo para sólo ser habilitados una vez que el alumno sea experimentado en el uso de la herramienta, tenga claros los conceptos necesarios y en concecuencia ya pueda interesarse en transicionar hacia un lenguaje escrito.

2.2.4. ¿En qué se diferencia nuestra propuesta a lo existente?

Ahora bien, luego de haber planteado una descripción inicial de lo que queremos de nuestra herramienta, cabe preguntarnos, ¿En qué se diferencia nuestra propuesta a lo ya existente? Creemos que esta pregunta no es menor, ya que nuestra intención no es hacer una mera crítica del estado actual de las cosas, o tener un prototipo que sirva como prueba de concepto para ilustrar dicha crítica. Lo que querríamos como resultado de este trabajo es tener una herramienta que, aunque requiera desarrollo futuro como cualquier proyecto de software, sea algo con utilidad práctica y tenga algún impacto en el campo educativo. Creemos que nuestra propuesta tiene ese potencial.

En primer lugar, está pensada para enseñar Programación Orientada a Objetos (POO) con una didáctica específica desarrollada en Argentina con foco en los mensajes y métodos [43]. La mayoría de las propuestas existentes apuntan a enseñar el razonamiento lógico o la programación en términos más generales [52, 49], a la programación imperativa [41], o una visión de la programación orientada a objetos a la que le faltan conceptos fundamentales como el polimorfismo, o los postergan hasta el final [32, 10, 25].

Nuestro público objetivo es el universitario o pre-universitario. La mayoría de las propuestas parecen tener como objetivo a una audiencia específicamente infantil.

Proponemos usar una cantidad minimal de bloques para definir un objeto y la interacción entre ellos. Una de las más fundamentales disimilitudes que percibimos es que a diferencia de otras propuestas, no queremos que un bloque sea percibido como un fragmento de código, sino como parte de una definición de un elemento primario del paradigma. Queremos que nuestros bloques empleen la menor cantidad de texto para hacerlos intuitivos y entendibles. Elegiremos siempre que sea posible otras metáforas visuales como el color o los iconos por sobre el texto para comunicar. Por último queremos que no sea necesario ser una persona técnica con previos conocimientos de ningún otro lenguaje de programación para usar o entender la herramienta.

2.3. El proceso de construcción

2.3.1. Primera iteración: JavaScript

La primera iteración estuvo marcada por una etapa inicial de experimentación y exploración que tuvo como objetivo familiarizarnos con Blockly e intentar comprender tanto el alcance de este como su flexibilidad. De hecho, en esta etapa inicial todavía no teníamos en claro a que lenguaje se iba a traducir el código generado por los bloques, por lo que elegimos Javascript por su flexibilidad.

Como muestra la figura 2.1, estos primeros intentos lejos se encontraban de seguir los lineamientos planteados, ya que los bloques se amparaban mucho en el uso de texto, dando una descripción quizás familiar para una persona con conocimiento técnico sobre programación, pero poco intuitivo para un neófito.

A pesar de esto, aún podemos distinguir aquí los elementos básicos que queríamos poder constituir a través de bloques: un objeto (*create object named*), un método (*method from instructions*) y un envío de mensaje (*let object execute method*). Todos estos traducibles a código Javascript.

El Ambiente

Al concluir la etapa de experimentación terminamos obteniendo un ambiente propio con las principales features que detallamos en nuestros lineamientos.

El ambiente cuenta, como muestra la figura 2.2, con (1) una sección de pestañas que muestra que sección del ambiente se está utilizando. En cualquier momento el usuario puede estar definiendo objetos, método o trabajando sobre la escena principal que luego será ejecutada. Dependiendo de la sección en la que se encuentre tendrá su propio (2) toolbox y workspace. El código generado se puede ver en (3) la sección inferior y la escena (4) en la sección derecha. Si bien esta sección permanece esencialmente vacía a excepción del botón de ejecución, es en esta sección donde estaba proyectado mostrar el resultado de la ejecución de manera gráfica.

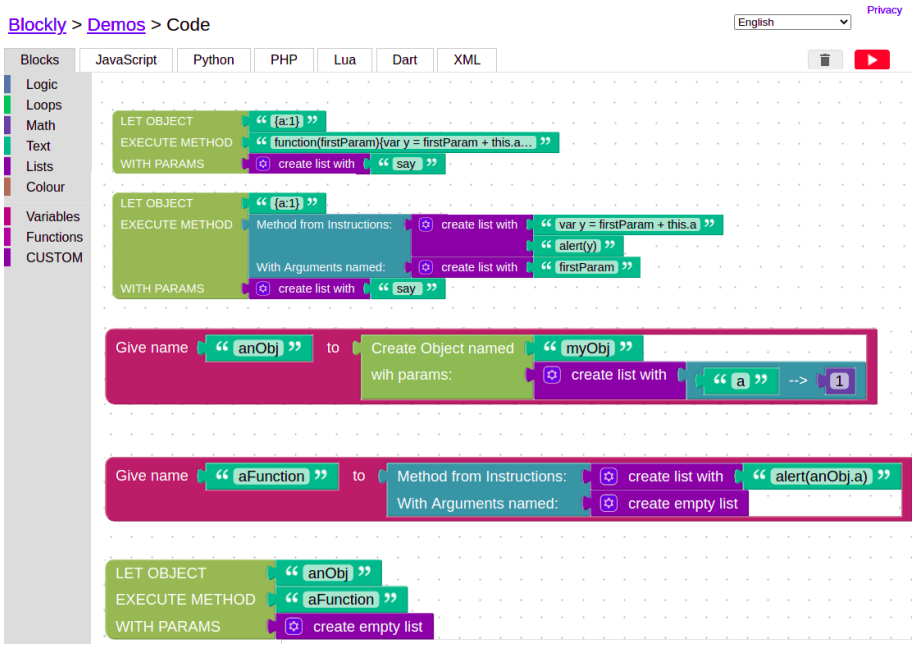


Fig. 2.1: Bocetos producto de la primera etapa expetimental para testear el alcance y flexibilidad de Blockly.

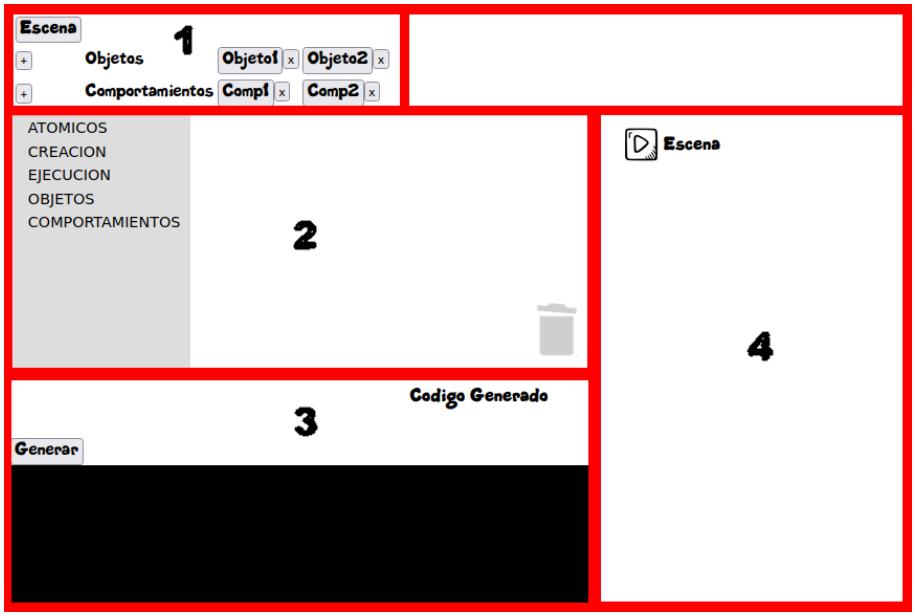


Fig. 2.2: El Ambiente cuenta con una sección de pestañas de escena/objetos definidos/métodos definidos, el workspace y toolbox correspondientes a la pestaña actual, una sección de código generado y una para la futura ejecución de la escena.

Ejemplo básico

La figura 2.3 muestra los tres conceptos básicos que queremos poder expresar utilizando bloques, así como el resultado de la ejecución y el código generado.

El primer bloque define un método. Utiliza un icono que representa una acción y requiere un bloque de texto para definir su nombre, un bloque lista para definir el nombre de sus parámetros y bloques de texto apilables verticalmente para definir sus instrucciones.

El siguiente bloque define un objeto. Requiere un bloque de texto para especificar su nombre, y se le pueden acoplar bloques de atributo que pueden definir tanto un atributo como el nombre de un método ya definido.

El último bloque representa el envío de un mensaje. Sólo requiere un bloque de texto para especificar el nombre de un objeto ya definido, uno para especificar el mensaje a serle enviado y opcionalmente los parámetros que este recibe.

Se puede observar también el código generado y el resultado de la ejecución.

Fig. 2.3: Ejemplo básico con definiciones explícitas de objetos y métodos, junto con código generado y ejecución.

Creando Objetos y métodos

Si bien en el ejemplo anterior podemos ver como el método y el objeto están definidos en el mismo lugar donde definimos el envío del mensaje (la escena principal), desde el principio nos quedó claro que en ejemplos donde se requieran definir una cantidad mayor de objetos y métodos esto puede resultar engorroso y difícil de leer.

Para solucionar esto, decidimos dotar a la herramienta con mecanismos para favorecer la modularidad. Es por eso que se pueden definir objetos y métodos en secciones diferenciadas, para luego utilizar solo un bloque de objeto autodefinido que represente dicho objeto o métodos.

Las figuras 2.4 y 2.5 muestran como la herramienta permite la creación de un objeto y método respectivamente. Una vez definido el nombre del mismo se accede a un workspace diferenciado donde no solo se define el elemento en sí, sino que también se personaliza el modo de mostrarlo en la escena principal.

Para ambos elementos se puede seleccionar un icono de una lista predefinida y un color, que conformaran el bloque de objeto autodefinido. Adicionalmente, se agregó un feature que permite dejar el bloque incompleto, pudiendo parametrizar en la escena el contenido faltante.

En el ejemplo mostrado por la figura 2.4, se define un objeto llamado manzana con un atributo energía. Sin embargo, en vez de agregar un bloque numérico que especifica la cantidad inicial de este atributo, se deja incompleto. De esta manera, cuando el bloque de objeto autodefinido sea generado en la escena, tendrá el hueco correspondiente para que se encastre allí un bloque numérico. El usuario podrá ver entonces desde allí la energía que ha definido sin tener que dirigirse a la pestaña del objeto, lo cual creemos que agiliza el uso de la herramienta.

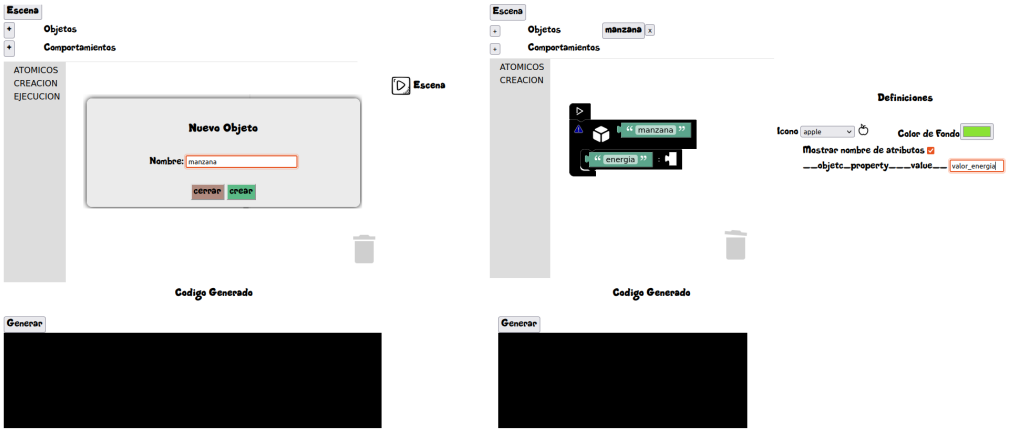


Fig. 2.4: Definición de un objeto. Popup y workspace resultante con el bloque generado por default y las configuraciones del objeto siendo definido.

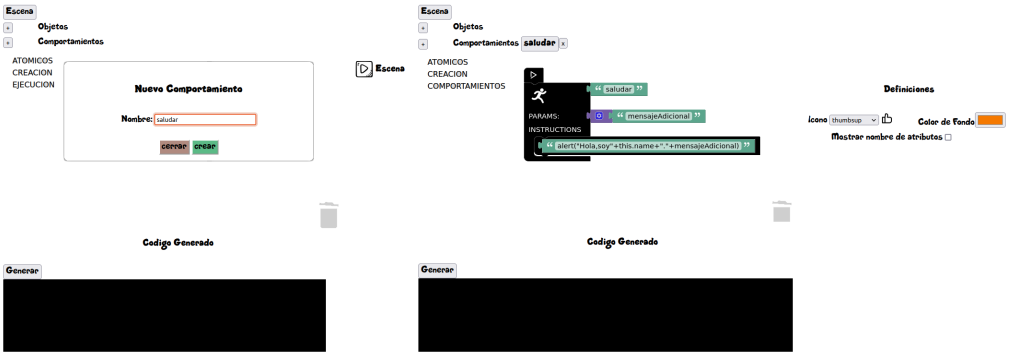


Fig. 2.5: Definición de un método. Popup y workspace resultante con el bloque generado por default y las configuraciones del método siendo definido.

Ejemplo integral

La figura 2.6 muestra un ejemplo que ilustra todas las principales características de la herramienta.

Aquí encontramos un objeto y un método que fueron creados mediante las pestañas, para que su representación interna permanezca oculta en la escena. El objeto manzana posee una ranura para poder definir su energía desde afuera.

También tenemos un objeto y un método creados de manera explícita, donde podemos ver y configurarlos en la misma escena. Creemos que como pasa siempre en el desarrollo de software, a la hora de diseñar un objeto o método el alumno primero enfrentara una etapa de descubrimiento, prueba y error donde no está seguro de lo que tiene que construir y por tanto la definición del objeto sufrirá repetidos cambios. En esta etapa es útil tener el objeto definido explícitamente. En una etapa posterior donde ya entienda lo que el objeto debe ser y este familiarizado con su comportamiento y lo que representa, es más cómodo visualmente verlo como un bloque de objeto autodefinido, con nada más que un icono y un color.

El ejemplo muestra además el envío de dos mensajes y el resultado del mismo: pepita come la manzana, sumando 50 unidades de energía correspondientes a esta a su propia energía inicial de 100 unidades. Por tanto, su energía termina siendo 150 como lo indica a través del mensaje “decirEnergia”.

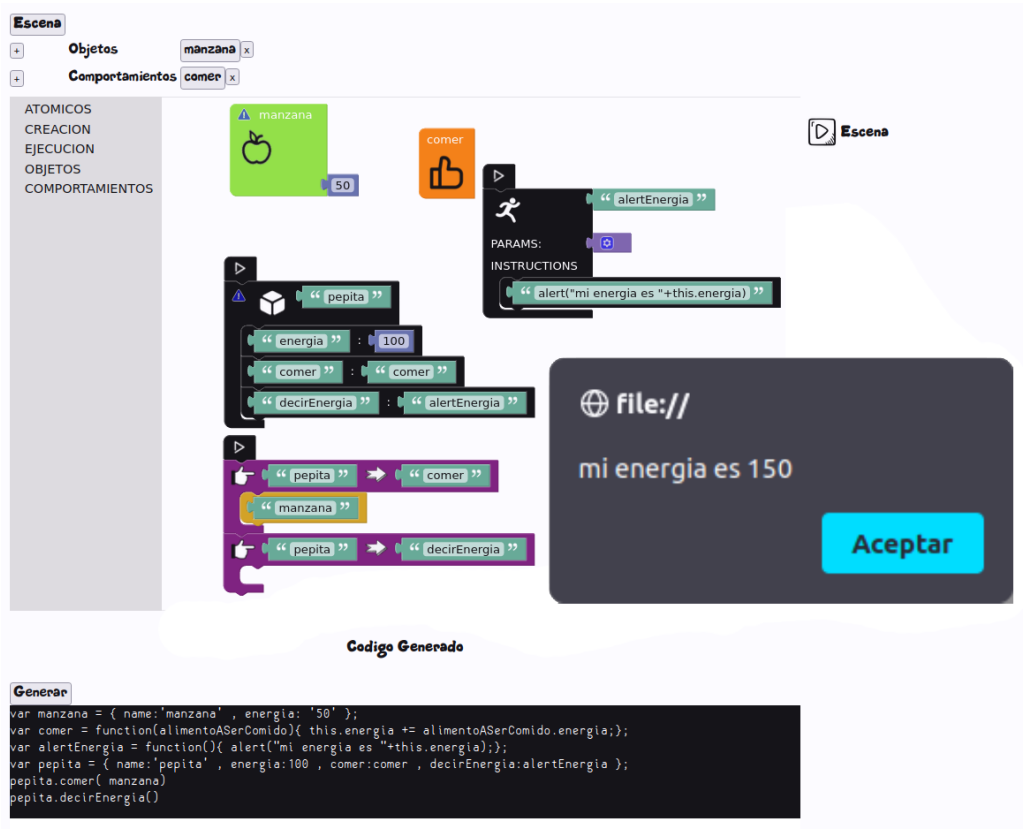


Fig. 2.6: Ejemplo integral mixto.

2.3.2. Segunda iteración: Wollok

Iniciamos la segunda etapa en el momento que decidimos que nuestra herramienta iba a traducir a lenguaje Wollok.

Se llegó a esta decisión luego de entrar en contacto a través de repetidas entrevistas con desarrolladores del proyecto e identificar la afinidad de las ideas que poseíamos hasta ese momento con el espíritu general y objetivo de Wollok.

Si bien puede parecer que este es un mero cambio del traductor de bloque a código, elegir Wollok implicó adentrarse en su didáctica, en su propuesta y en su forma de ver el mundo.

Uno de los cambios más importantes es que a diferencia de nuestro planteo anterior, los métodos (es decir la implementación de los mensajes) nunca se definen por fuera de un objeto. Nos enfocamos en los *objetos autodefinidos* de Wollok, donde cada objeto tiene los métodos con los que se responde a los mensajes que le llegan [43].

Si bien consideramos que las clases son sumamente útiles como concepto en el contexto de la POO, decidimos dejarlas afuera de esta primera versión de la herramienta, postergándolas como propone la propuesta didáctica de Wollok [43].

A pesar del hecho de que los métodos se definan dentro de un objeto, no nos apartamos de nuestra concepción de cuáles son los elementos básicos que se pueden definir en el ambiente. Seguiremos teniendo objetos métodos y envío de mensajes como las tres cosas definibles a través de bloques; solo que en vez de que un método sea definido para luego ser referenciado, este se encontrara siempre dentro de la definición de un objeto. Esto hizo que se elimine las pestañas de métodos del ambiente, dejando solo las de creación de objetos.

Por otro lado, una de las razones para elegir Wollok fue que posee la extensión Wollok Game, y comenzamos a trabajar en incorporarla. Wollok Game es una extensión para crear pequeños juegos en 2D con Wollok, pensada respetando la propuesta didáctica. Entonces el alumno puede no solo intercambiar mensajes entre objetos que defina, sino que al hacerlo puede crear un videojuego, lo cual constituye una mejora categórica en la calidad educativa de la herramienta que nos propusimos producir.

Mejoras

Presentamos ahora todas las modificaciones hechas a la herramienta a partir de esta iteración, que incluyen no sólo las metáforas propias de Wollok sino mejoras generales también.

En principio, nos propusimos reemplazar los textos de los bloques por representaciones gráficas.

Adicionalmente, incorporamos un bloque auxiliar que permite introducir texto que se traducirá directamente como código. Identificamos la utilidad práctica de dicho bloque al comenzar a usar la herramienta, pero al mismo tiempo comprendimos que su necesidad era un reflejo de la falta o inefectividad de los bloques disponibles. Una desventaja adicional es que este bloque no representa ningún concepto. Nos referiremos a él como bloque "comodín", y más adelante en la figura 2.19 se muestra cómo se logró hacerlo desaparecer tras las distintas iteraciones.

Otro inconveniente que solucionamos es que en la versión previa, luego de definir un objeto había que arrastrarlo a la escena para que este exista. Notamos que esto podría volverse engorroso en ejemplos más extensos y que rara vez se define un objeto que no se necesite utilizar. Por esta razón decidimos que todo objeto definido en las solapas estará

implícitamente definido sin necesidad de arrastrarlo a la escena. Más aun, para cada objeto definido existirá un bloque de *objeto autodefinido* (con el icono representatorio del objeto), el cual puede utilizarse en vez de un bloque de texto con el nombre del objeto. Esto refuerza la semántica visual de enviarle un mensaje a un objeto. En la versión anterior, un objeto definido tenía su representación, pero para enviarle un mensaje era necesario hacerlo utilizando un bloque de texto con su nombre.

Adicionalmente, hay ahora disponible un bloque de *objeto autodefinido* por default que representa el objeto Game de Wollok Game.

En cuanto a los colores, decidimos que todos los objetos tengan un mismo color de fondo. Creemos que el color es un lenguaje sumamente poderoso y que dejar que el usuario defina el color de cada objeto constituye un pobre uso de dicho poder. En cambio, en esta iteración todos los bloques de objetos autodefinidos tienen un mismo color. Esto ayuda a visualmente identificarlos en el ambiente y le da al color un sentido mucho más claro. Más aún, consideramos que aporta a la armonía cromática.

Rediseñamos también el bloque de envío de mensaje. En esta nueva versión posee una ranura donde debe ir un bloque que resuelva al nombre de un objeto (por ejemplo un objeto autodefinido) y el nombre del mensaje a ser enviado pasó de ser un bloque de texto a un input de texto. Ahora no sólo se tiene un bloque de envío de mensaje sino que para cuando el mensaje enviado tiene algún tipo de retorno, un bloque encastrable a izquierda. Esto es porque un envío de mensaje a veces se usa como comando, y a veces como valor (cuando me interesa su retorno). Destacamos que esto forma parte de nuestra propuesta pedagógica, y que aporta claridad de conceptos.

Agregamos además un bloque condicional que posee una ranura que debe contener un bloque que evalúe a un resultado booleano. Este bloque define también una secuencia de comandos para el caso positivo y negativo, similar a otros lenguajes de bloques existentes [33, 52, 49]

Otros bloques nuevos que fueron incorporados fueron tres bloques específicos de Wollok Game. Si bien estábamos reacios a agregar demasiados bloques específicos a nuestra herramienta, entendimos que existe cierto *tradeoff* que tuvimos que aceptar en favor de hacerla más practica, clara y entendible. Estos bloques definen tres eventos básicos en Wollok Game que resultaron de gran utilidad a la hora de generar un primer ejemplo. Se trata de bloques que definen el evento de botón de teclado, evento temporal y evento de colisión.

La decisión de utilizar Wollok Game trajo apareado una nueva dimensión de cosas que manejar desde la herramienta como lo es el tener que administrar las imágenes (sprites) que se utilizaran en el juego. Nuestro primer enfoque fue el de la carga manual, pero fue rápidamente migrado a un sistema mixto donde hay una serie de imágenes predefinidas, más la posibilidad de agregar otras.

Finalmente, al dar con una versión más acabada de la herramienta y poder probar más intensamente su uso, pudimos darnos cuenta de la ventaja que representa el poder guardar o cargar el trabajo realizado. Es por esto que dotamos a la herramienta con la posibilidad de importar y exportar la configuración actual del ambiente.

Versión resultante de la iteración

La figura 2.7 muestra como se ve la nueva versión de la herramienta. Conserva la misma disposición que la versión anterior con algunas modificaciones.

La parte inferior derecha ahora guarda un tablero con las imágenes disponibles para utilizar (ya sean las precargadas o las agregadas a mano), así como las dimensiones del tablero del juego.

La sección derecha ahora si es un espacio reservado para el juego siendo ejecutado, en contraste con su concepción anterior.

La sección de código generado ahora es minimizable, lo cual refuerza visualmente la idea de que lo importante no es el código que se termina ejecutando sino los objetos e intercambio de mensajes que definen.

La sección de pestañas, por otro lado es bastante similar, pero ahora solo muestra objetos definidos.

Y por último la parte superior derecha tiene dos componentes que cumplen la función de guardar o cargar el proyecto actual.

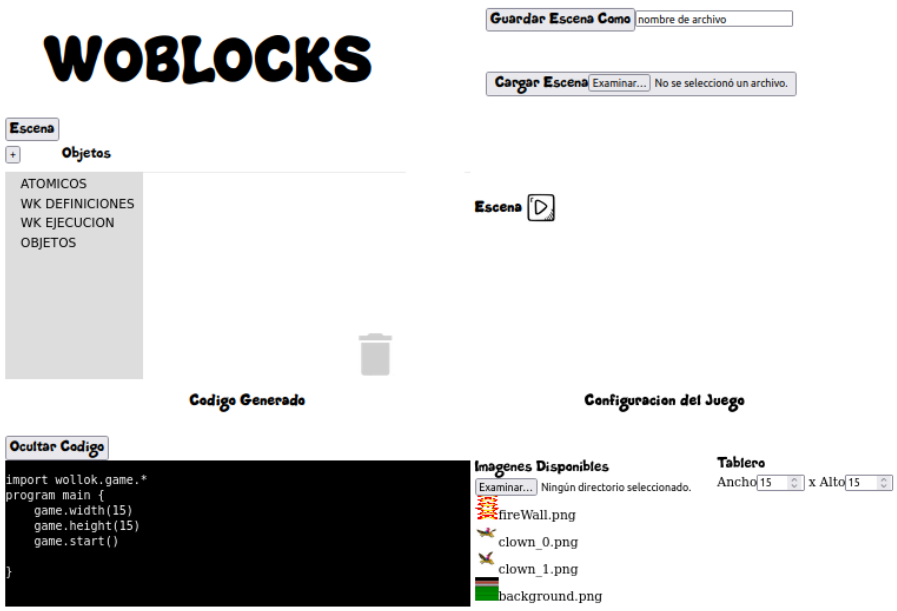


Fig. 2.7: Segunda versión del ambiente, ahora con Wollok.Conserva la disposición original agregando guardado/cargado de proyectos, configuraciones del juego e imágenes utilizables.

Podemos observar en la figura 2.8 un ejemplo básico con un objeto definido implícitamente. El primer bloque que tiene encastrado define un atributo mientras que el siguiente es la definición de un método. A continuación se ve el compendio de bloques que representa el programa principal donde al objeto se le envía un mensaje.

La definición de objetos es similar a la versión anterior, pero en este caso no se cuenta con la posibilidad de elegir un color para el mismo. También se eliminó la funcionalidad asociada a cuando un objeto está incompleto, ya que con la nueva concepción del bloque de objeto autodefinido esto dejaba de tener sentido.

El resultado puede verse en la figura 2.9. Una vez creado el objeto, al volver a la escena principal el bloque de objeto autodefinido ya se encuentra disponible para su uso. Cuenta con un color predefinido y el icono que se eligió para representarlo, haciendo el envío de mensajes más simple, elegante y ya no dependiente del texto para identificar un objeto.

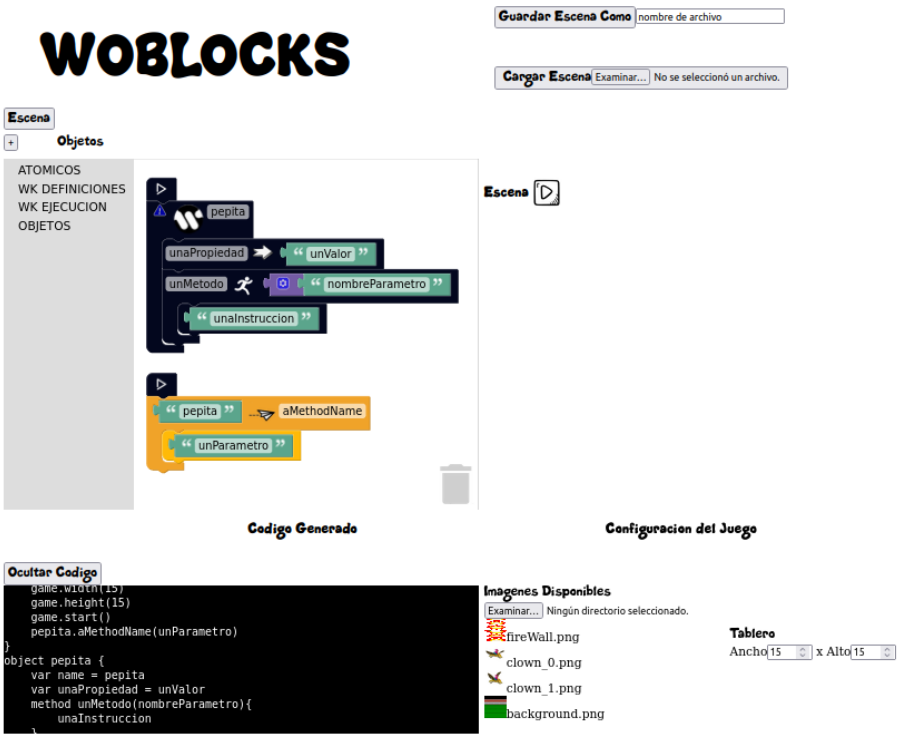


Fig. 2.8: Ejemplo Básico con Definiciones Explícitas para la primera versión de la herramienta que utiliza Wollok. Un objeto donde los métodos son definidos dentro de ellos y un envío de mensaje.

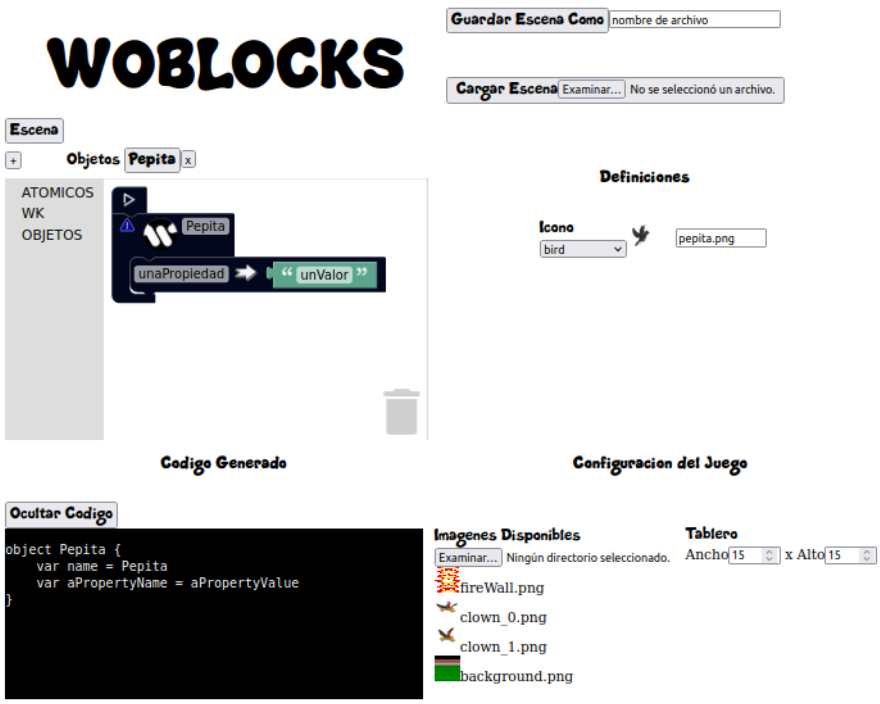


Fig. 2.9: Definición de un objeto.

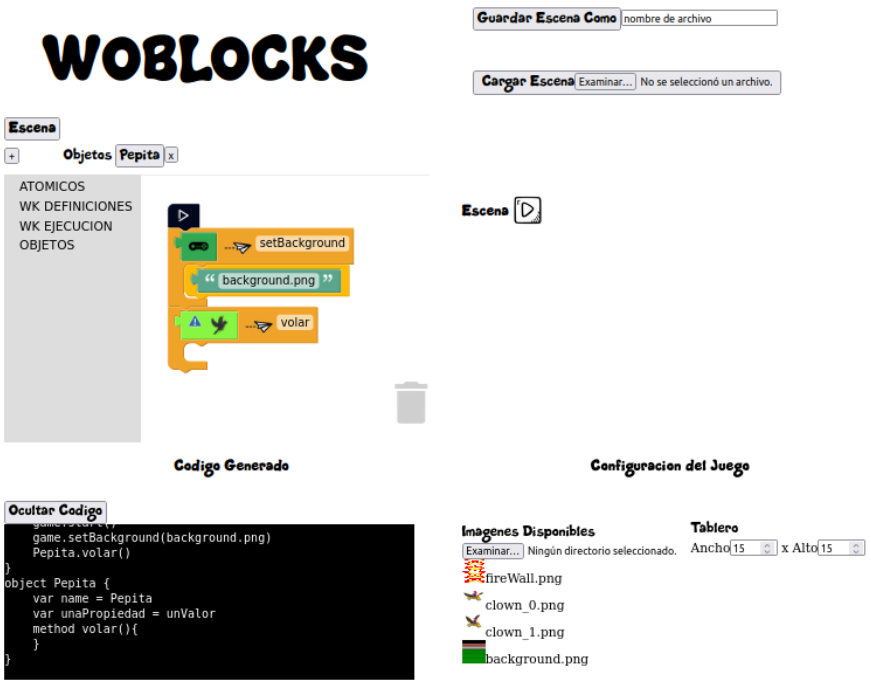


Fig. 2.10: Ejemplo básico con objeto definido.

Primer juego con **WoBlocks**

Una vez que nos encontramos conformes con que la herramienta contaba con todos los features que nos habíamos propuesto y todo lo que podíamos hacer con la versión Javascript era posible para la actual, decidimos poner a prueba la practicidad y poder de uso de esta nueva iteración. Nos propusimos, con el estado actual de la herramienta, crear un juego básico de Wollok Game. Uno que se podría pedir a un alumno luego de haber superado un breve curso inicial sobre POO y el uso de **WoBlocks**. Esto nos permitiría testear la herramienta en su utilización esperada y poder hacer un uso integral de cada funcionalidad, pudiendo ensayar de primera mano la experiencia que tendría un usuario utilizándola.

El resultado fue un juego quizás un poco más complejo que lo que se le podría exigir a un principiante (dado el nivel técnico de quien se dio a la tarea de realizar esta prueba), pero sirvió para ilustrar el tamaño y legibilidad de los compendios de bloques necesarios para generar el resultado deseado.

Elegimos imitar un conocido juego para móviles creado por un único desarrollador hanoiense que se hizo viral a mediados del 2013: Flappy Bird. Este juego se suele utilizar para comenzar también en la propuesta de Wollok Game [59]. El jugador controla un pájaro que es arrastrado hacia abajo por la gravedad y su única acción consiste en darle un impulso ascendente mientras se mueve hacia adelante automáticamente. Cada cierto tiempo y moviéndose en dirección contraria, aparecen en pantalla dos tubos que dejan un espacio entre ellos a cierta altura donde el jugador deberá posicionar al pájaro al momento que este se encuentre con los tubos. Si esto pasa, el jugador incrementa su puntaje. En caso contrario, el juego termina.

Aprovechamos la oportunidad para hacer un cambio estético al juego, rindiendo homenaje a cosas más antiguas que aquel juego de la pasada década: un programa humorístico ingles de principios de los años 70 y un clásico videojuego circense japones publicado en 1984 para la consola Nintendo Entertrainment System. Fue así que dimos con el primer juego de Wollok Game creado con nuestra herramienta: ***Monthy WoBlocks Flying Circus***.

Sorprendentemente, reproducir este juego en Wollok Game nos requirió únicamente de dos objetos principales. Uno es el avatar del jugador que debe impulsar, y el otro un aro de fuego por el cual debe pasar para evitar chocar con una pared de fuego. Así como en Flappy Bird, cada vez que pasa por el mismo incrementa su puntaje, mientras que al no poder hacerlo termina el juego.

La figura 2.12 muestra el juego en funcionamiento. La sucesión de paneles muestra el momento inicial del juego junto con sus dos posibles desenlaces. En el caso exitoso, el aro de fuego reaparecerá en una altura aleatoria una vez alcance el borde de la pantalla, mientras que en el no exitoso la ejecución se detendrá.

La figura 2.11 muestra los tres compendios de bloques que definen respectivamente al avatar del jugador, el aro de fuego y el programa principal que se encuentra en la escena.

Según lo requiere Wollok Game, para que un objeto pueda aparecer (visualmente) en la escena debe saber responder dos mensajes básicos: `position()` y `image()`. En este caso, también tiene mensajes y atributos asociados a su animación y su puntaje. El aro de fuego, además de position e image sabe moverse a la izquierda y puede identificar cuando ha alcanzado el borde de la pantalla. Cuando esto sucede, tiene un mensaje para resetear su posición con una altura aleatoria.

Finalmente tenemos el programa principal. Se tiene básicamente un trigger temporal

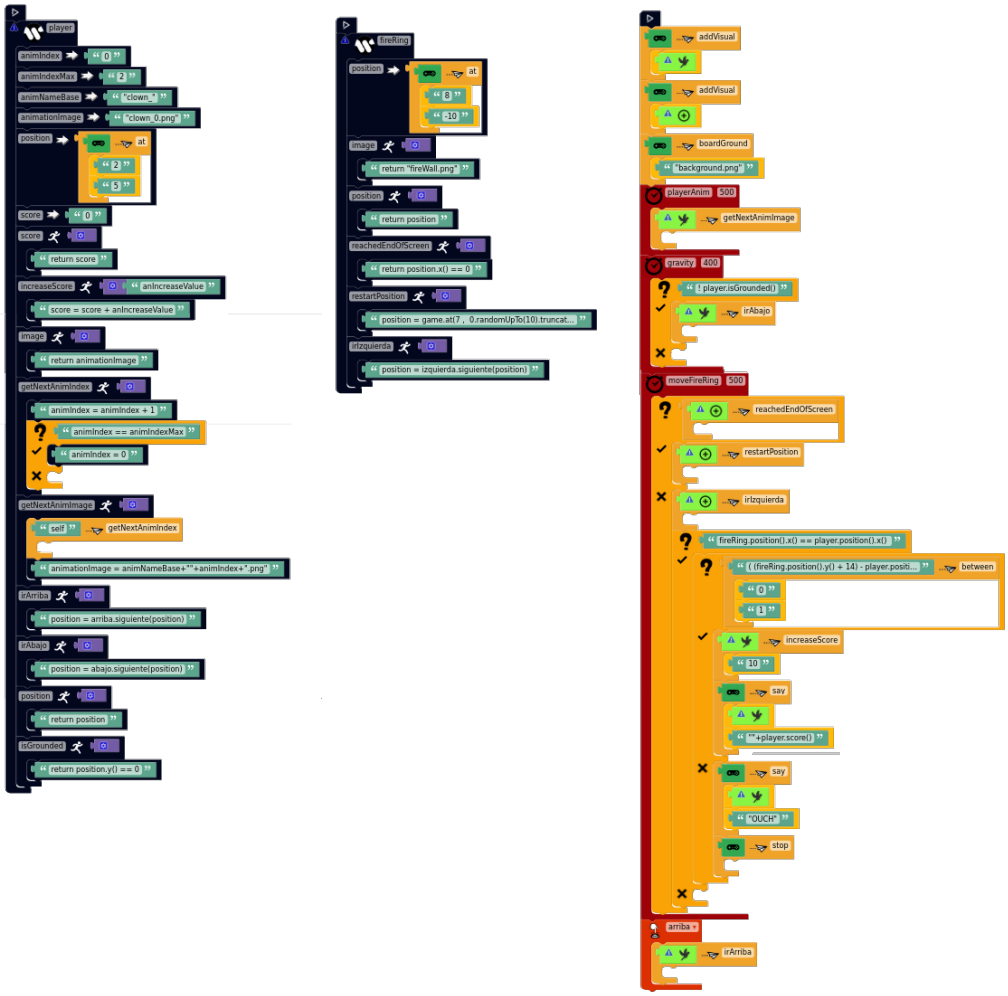


Fig. 2.11: Los tres Bloques utilizados para definir el Juego de ejemplo. El avatar del jugador con sus mensajes referidos a su animación y posicionamiento, el aro de fuego con sus mensajes de `position()` y su mensaje `image()`, y el bloque de la ejecución principal con los eventos de animación, gravedad y movimiento del aro de fuego.

para la gravedad, uno para la animación del avatar del jugador y uno para mover el aro de fuego. Los primeros dos solo mueven al jugador una posición hacia abajo y avanzan la animación (o la resetean si ha llegado al final) respectivamente. En el caso del trigger temporal del aro de fuego, este contiene la lógica que identifica si se ha llegado al final de la pantalla o se ha llegado a estar a la misma altura del jugador. Si ninguna de estas cosas suceden, solo se mueve a izquierda. Si ha llegado al final de la pantalla se resetea su posición. Mientras que si el aro y el avatar están en la misma posición en el eje vertical, se fija si se encuentran a la misma altura. En caso afirmativo incrementa el puntaje. De lo contrario, finaliza la ejecución.

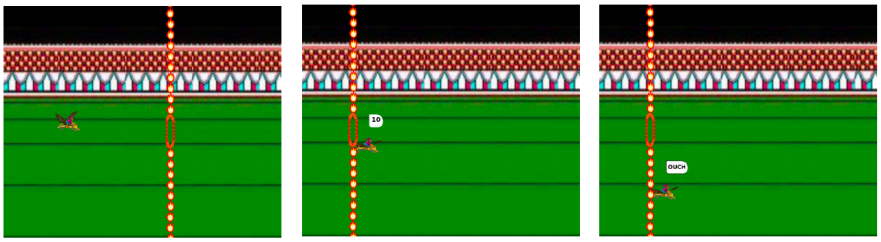


Fig. 2.12: El juego obtenido en ejecución. El avatar del jugador en su etapa previa al encuentro con el aro de fuego y los dos posibles desenlaces posibles. Posición en el eje vertical coincidente e incremento de puntaje, y posición vertical no coincidente resultando en la finalización del juego.

2.3.3. Tercera iteración: ReactJS

Habiendo cerrado el primer juego creado con la herramienta y habiéndonos convencido de que a nivel funcional habíamos cubierto satisfactoriamente los objetivos propuestos, nos volcamos a otro aspecto que identificamos como fundamental para que nuestra propuesta trascienda a un prototipo de prueba de concepto y se convierta en una herramienta que si bien necesitara seguir desarrollándose, sea llevada a un ambiente pedagógico real y genere un impacto. El próximo paso sería un rediseño desde el punto de vista visual con el objetivo de llevar **WoBlocks** a los niveles de mantenibilidad, *look and feel* y usabilidad esperados de una pieza de software producida en el año 2022.

Para lograr este objetivo, siendo que ninguno de nosotros tenía demasiado expertise en el área, realizamos una consulta a un especialista en usabilidad que nos marcó diferentes puntos de mejora y nos recomendó hacer uso de un framework para dar con una nueva versión.

Adicionalmente, recomendó para guiar el proceso de construcción el uso de la técnica de paper prototyping [61], que fue incorporada para el diseño de la interfaz gráfica de la nueva versión. La figura 2.13 muestra un ejemplo de la técnica siendo aplicada para el diseño de las pantallas de la aplicación.

Luego de un proceso de selección e investigación en las distintas alternativas en cuanto a frameworks, nos decidimos por ReactJS, una de las tecnologías más elegidas en la actualidad para el desarrollo de aplicaciones multiplataforma que mayor versatilidad y relativa baja curva de aprendizaje presenta. React extiende Javascript permitiendo con relativa facilidad crear aplicaciones modernas y dinámicas. Está basado en componentes y cuenta con la ventaja de bajar el nivel de procesamiento requerido, ya que su filosofía principal dicta que ante un cambio en el estado de la aplicación, solo va a re-renderizar aquellos componentes que fueron afectados por el mismo en vez de refrescar todos los componentes. Al tiempo de escribir este trabajo, este framework es ampliamente utilizado incluso en proyectos como Instagram, Facebook, Reddit, Paypal, entre otros. Esto implica también la existencia de una enorme comunidad y una gran cantidad de material disponible. Este framework cumple además con un objetivo crucial: nuestra intención no es que **WoBlocks** sea un mero *proof of concept* de un conjunto de ideas con respecto a las herramientas educativas existentes que quede confinado a este trabajo únicamente. Queremos que la herramienta resultante sea mantenida, corregida, adaptada y expandida siguiendo nuestros principios directrices para cumplir su cometido de educar en conceptos de POO.

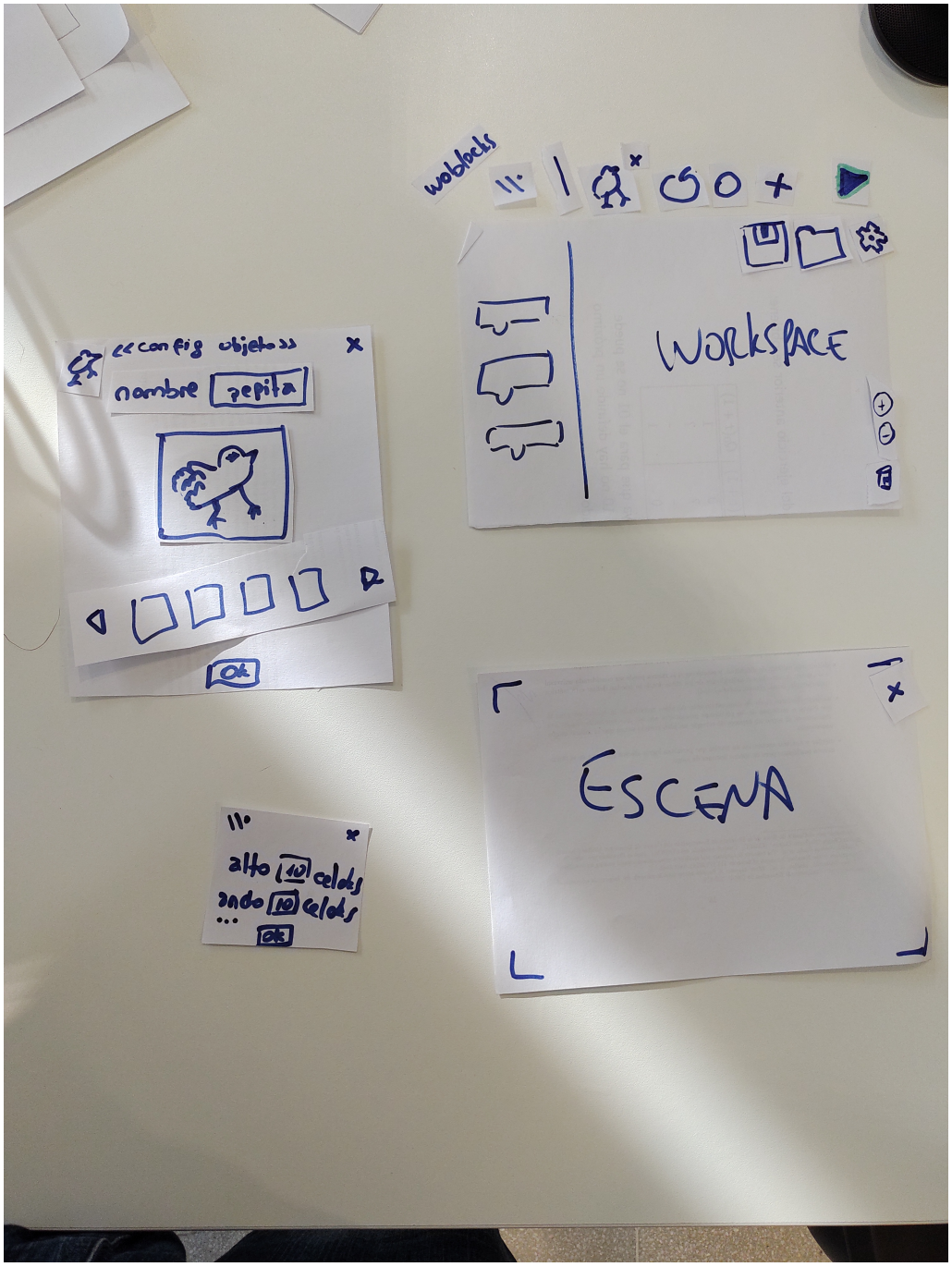


Fig. 2.13: Muestra de los prototipos de papel.

El proceso de desarrollo significó un desafío grande aunque esperable como lo es tener que familiarizarse con un nuevo framework. En el caso de ReactJS, el cambio es mucho más profundo que en otros. React representa un cambio de paradigma en la lógica del comportamiento e interrelación de los componentes visuales que lo componen. Su filosofía principal es que un componente solo se redibujará cuando sea modificado, cosa que React se compromete a manejar. A diferencia de otros frameworks, el control que el programador

tiene sobre los componentes visuales es sumamente limitado, debiendo delegar al propio motor una serie de responsabilidades que en otros casos se encontraban del lado del programador. A su vez, el desarrollo en esta tecnología hace especial énfasis en el empleo minimal de componentes anidados reutilizables, priorizando siempre el uso de elementos preexistentes ya definidos.

La construcción de esta nueva versión requirió en primer lugar familiarizarse con el lenguaje TypeScript (una extensión de JavaScript con chequeo y anotaciones de tipos) y JSX (una extensión de JavaScript para generar interfaces HTML), para después, entender el funcionamiento interno de React y como reproducir los comportamientos ya existentes. La comunicación entre componentes se puede manejar de diferentes maneras. Nosotros optamos por manejar un estado global al cual los componentes acceden para reflejar los cambios que cada uno realiza.

Otro gran proceso de ajuste que requirió varias iteraciones tuvo que ver con el planteo de la arquitectura general de la herramienta. Como ya mencionamos, React tiene su propia filosofía y modo de desarrollo. Aprenderlo y traducir esto en un correcto sistema de interacción entre componentes fue una tarea sumamente difícil. Esta comunicación puede lograrse a través de diferentes estrategias como lo son la herencia de atributos, los eventos o los estados globales. Nuestra solución utiliza principal y casi exclusivamente los estados globales como forma de comunicación entre componentes.

El ya mencionado cambio de paradigma manifestó también la necesidad de separar completamente el modelo de la aplicación de la vista y control. Estas últimas dos serian ahora manejadas desde React, mientras que el primero debería permanecer independiente y debería ser accedido desde afuera como si se tratase de una caja negra. Si bien esta separación existía en gran medida en las versiones anteriores, un retrabajo fue necesario en este sentido.

Como ha sucedido en cada iteración, el planteo de una versión nueva no representa una mera traducción de lo ya existente, sino una oportunidad de analizar el estado de herramienta y repensarla, teniendo en cuenta sus objetivos primordiales y de qué manera estos están siendo satisfechos. En este caso el mencionado proceso trajo no solo un rediseño de la interfáz de usuario (con un fuerte enfoque en usabilidad) sino que trajo como conclusión la creación de dos bloques nuevos, así como la modificación de otros.

Como muestra la figura 2.14, la nueva interfaz gráfica presenta una apariencia moderna y elegante, donde en vez de presentar todas las secciones simultáneamente a todo momento, estas funcionalidades se encuentran ocultas y solo se activan mediante ventanas emergentes al accionar el botón correspondiente. También se puede apreciar a primera vista un uso más minimalista del texto.

La aplicación cuenta con una barra principal y una sección de toolbox y workspace. El icono de **WoBlocks** representa la pestaña de la escena principal, mientras que todo objeto creado se encolará a la derecha de este. Otro paso en favor de reducir el texto priorizando otro tipo de representaciones fue la utilización de iconos en las pestañas de objetos definidos, en contraste de la utilización de sus nombres en las versiones anteriores (si bien los nombres se conservan en forma de tooltip al posarse sobre la pestaña, ya que dos objetos distintos pueden tener el mismo icono). Esto agrega claridad y hace a la herramienta más agradable a la vista y fácil de leer.

A continuación de las pestañas se encuentran una serie de botones que al ser presionado habilitan un popup que permite realizar diferentes acciónes :

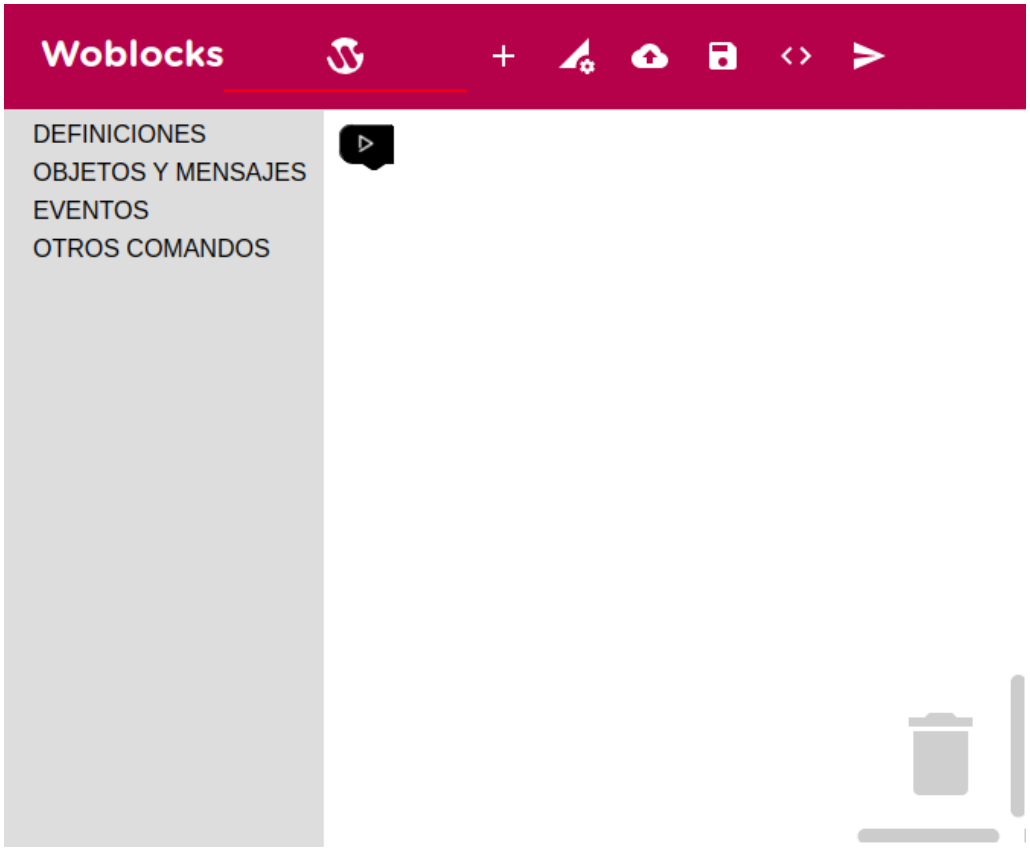


Fig. 2.14: La nueva interfaz principal de la versión de **WoBlocks** utilizando ReactJS.

Nuevo objeto

De todas las nuevas funcionalidades, esta sin duda fue la que presenta un mayor avance en términos de diseño y usabilidad. Anteriormente, para definir un objeto nuevo solo era necesario proporcionar un nombre. Una vez creado se contaba con la posibilidad de manipular mediante una sección aparte el icono que representa al objeto. En esta nueva versión, según muestra la figura 2.15, el icono se define por primera y única vez al generar el objeto.

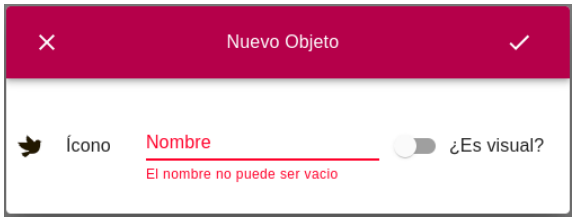


Fig. 2.15: La ventana utilizada para crear un objeto no visual.

Adicionalmente, se destiló el concepto de "Objeto visual". En Wollok Game, para que un objeto sea visible en el juego debe contar con dos métodos: image y position. Notamos que si bien tanto los objetos que serán rendereados como los que no son esencialmente

objetos, los primeros son conceptualmente distintos a la hora de pensar un videojuego. Es por esto que dotamos a la herramienta con la capacidad de definir y distinguir estos dos tipos de objetos. El modo de creación se define a través de un switch binario que cambia la conformación del popup. El modo de definición de objeto no visual es el ya ilustrado por la figura 2.15, mientras que la figura 2.17 muestra el modo de definición de objetos visuales.

Fig. 2.16: Al clickear el icono por default se abre la ventana de selección de iconos.

Fig. 2.17: El switch provisto por esta ventana emergente cambia entre modos, dejando ver para el caso visual el sprite utilizado para la representación.

Podemos ver aquí claras diferencias. En la definición no visual, al clickear el icono aparece un nuevo popup para seleccionar cuál se quiere utilizar como muestra la figura. En el caso de los objetos visuales, el icono no es seleccionable, sino que cambia automáticamente al cambiar de sprite. La figura muestra este proceso y también ilustra que a diferencia del caso de los objetos no visuales, donde la pestaña correspondiente al objeto solo contendrá un bloque de definición de objeto con el nombre completado, automáticamente se agregan los métodos image y position. Así lo ilustra la figura 2.17. Tal y como en la versión anterior, a partir de ser definidos de esta manera, bloques con el icono del objeto se disponibilizarán en el toolbox. A diferencia de la otras iteraciones, se decidió que el color del objeto no sea definible por el usuario, sino que todo objeto tenga un mismo tono unificado. Esto aporta tanto a la integridad estética de la herramienta como a una facilidad a la hora de identificar visualmente este tipo de bloques.

Fig. 2.18: Luego de crear un objeto visual, automáticamente se añadirán el método position y el método Image con el sprite correspondiente a la representación seleccionada.

Configuración

La sección de configuración permite, como en versiones anteriores, definir la altura y ancho de la pantalla de juego. Decidimos sacar del control del usuario también las imágenes disponibles de manera dinámica. Entendimos que representaba una complejidad adicional que para una primera versión resultaba innecesaria. En contraste, dotamos al usuario con la capacidad de elegir (de una lista de imágenes preestablecidas) la imagen de fondo que será utilizada en el juego.

Código generado, salvar, guardar

Las secciones de código generado, así como la de guardado y cargado de proyecto, no sufrieron más que un cambio estético. La única funcionalidad que se agregó es que al cargar un proyecto, la herramienta recordara el nombre de archivo para sugerirlo a la hora de guardar el archivo.

TOOLBOX Y BLOQUES

Bloque inicio ejecución

Este bloque era en versiones anteriores necesario a la hora de definir un objeto, programa principal o método. Si bien esto pareció un concepto interesante en un principio, notamos que en cierto punto prestaría para la confusión. Es por esto que este bloque ya

no se encuentra disponible en el toolbox, sino que aparece automáticamente en la escena principal una única instancia de este bloque (no se puede copiar ni borrar). Este bloque cumple entonces la única función de representar la ejecución principal en la cual encastraran todos los envíos de mensajes. Funciona de manera similar al bloque "Al empezar a ejecutar" de Pilas Bloques [33] o el bloque "bandera" de Scratch [52]

La manera de agrupar los bloques en el toolbox también fue ampliamente discutida. Llegamos a definir para esta iteración las siguientes secciones:

Definiciones

Esta sección contiene todos los bloques necesarios para definir un objeto junto con sus atributos y métodos. Contiene los bloques de definición de objeto, definición de atributo, definición de método, parámetro y return. Como ya se mencionó, el bloque objeto ahora no posee la ranura para el bloque inicial, sino que ahora existe por sí mismo. El bloque atributo no sufrió modificaciones, mientras que el de método ahora recibe un bloque especial que representa más claramente el hecho de que se está definiendo el nombre del parámetro recibido por el mismo. Ciertamente, no nos sentimos satisfechos con la metáfora de lista nativa de Blockly para definir parámetros por no ser lo suficientemente clara a nuestro entender, prefiriendo una acción que modifique dinámicamente el bloque, agregándole inputs de texto para definir un nombre de parámetro. Por razones de scope en la versión inicial tuvimos que conformarnos con la propuesta de la lista. El último bloque es el de return, un bloque con una ranura para encastrar otro bloque, a cuyo contenido le antecede la palabra reservada "return". Además de la ranura posee un icono ilustrativo. Si bien no estamos del todo conformes con la representación gráfica elegida, nuevamente por cuestiones de scope debimos apegarnos a esta versión.

Objetos y mensajes

Esta sección encierra los bloques necesarios para el envío de mensajes. Estos bloques tampoco han sufrido modificaciones significativas. La única relevante es el rediseño del bloque universal de text input. Antes poseía comillas rodeando el text input y era de color verde. Ahora es solo un bloque gris encastrable a izquierda con el input. Esta sección, como en versiones anteriores, también contiene los bloques que representan los objetos definidos y al objeto Game.

Eventos

Esta sección ahora encierra los bloques de evento de teclado, temporal y colisión, que no sufrieron modificación.

Otros comandos

Aquí se encuentran los bloques de menor importancia conceptual en lo que refiere a la enseñanza de POO, tal como el bloque condicional, la definición de variable, el bloque sprite y los bloques básicos de string, booleano y numérico. Por último también contiene un bloque de operador binario con las principales operaciones para estos tres tipos. En cuanto al bloque sprite, este surgió de la observación que en la versión anterior los sprites debían ser referidos por el nombre de archivo con el que fue definido. Esto representa una distancia grande e innecesaria entre el modelo mental del usuario y la forma de manipular un sprite. En su concepción, un sprite no es un nombre, sino simplemente una imagen. De hecho, en rigor, no importa su nombre. Es por eso que con este bloque (que exhibe un

droplist con una versión miniatura de la imagen) el usuario puede manipular directamente el concepto que existe en su razonamiento. Es el bloque el que se encarga luego de traducir la imagen al nombre asignado.

Los bloques restantes encierran conceptos básicos presentes en cualquier lenguaje de programación como lo son los tipos elementales de string bool y número (junto con un bloque de operaciones binarias básicas para ellos), la asignación de variables y el condicional simple.

2.3.4. Evolución de los bloques

La figura 2.19 muestra la evolución de los tres bloques principales (objeto, método y envío de mensaje). Cada tipo de bloque esta representado por una fila y cada columna una versión de la herramienta, siendo Javascript (primera iteración), Wollok (segunda iteración) y ReactJS (tercera iteración).

Fig. 2.19: Evolución de los bloques.

Recordemos que los principios que guiaron el desarrollo y evolución de las distintas versiones fueron los de simplicidad, intuitividad, facilidad de uso, representación clara de conceptos y minimalidad en el uso de texto.

En términos generales, las modificaciones destacables son:

- El reemplazo de texto por iconos: creemos que al ser una herramienta visual, debe utilizar este tipo de lenguaje siempre que sea posible, en vez de utilizar texto.
- El reemplazo del uso del bloque de texto por un text input: En las primeras versiones, vario bloques poseían una ranura para encastrar un bloque de texto. Entendimos que en la mayoría de los casos, lo que el bloque requería no era un bloque externo sino un input de texto. Una ranura de bloque indica que diferentes tipos de bloques podrían ser colocados allí, mientras que un text input solo permite introducir el (único) tipo de input que corresponde en aquel caso.

- Identificamos que en gran cantidad de ocasiones durante los usos de prueba de la herramienta, el bloque de texto suplía la falta de otros bloques. Esto nos sirvió no solo para identificar bloques necesarios, sino que también notamos que el bloque de texto representa un concepto claro y definido: un string. Mientras que el uso que se le estaba dando era la de representar pedazos de código (no representables a través de ninguna combinación de bloques). Es por esto que conservamos el bloque de texto para que cumpla su función primordial, y para uso auxiliar agregamos el bloque "comodín".

- La reducción del bloque con el icono de triangulo a una única instancia: Inicialmente, este bloque representaba un bloque de ejecución . Esto significa que una serie de envíos de mensaje podían ser armados por separado y modificar su posición vertical cambia el orden de ejecución. Adicionalmente, los bloques de objeto y método también requerían este bloque inicial, dando la facilidad de que el desencastrar este bloque inicial hace "invisible" a esa definición de método, objeto o bloque de ejecución. Se decidió suprimir esto por falta de claridad. Este era un bloque que no resulto intuitivo y brindaba mas confusión que utilidad.

- Finalmente, mejoramos el uso del color. Creemos que es un lenguaje muy poderoso sobre todo en lo visual para agregar claridad a la herramienta. Esto no solo se tradujo en decisiones concretas sobre los bloques existentes, sino que también en la libertad del usuario. En instancias no finales de la segunda iteración, al definir un objeto el usuario tenia la capacidad de definir el color que poseería el bloque de representación. Luego se decidió que todos los bloques que definen un objeto definido a través de pestañas tenga un mismo color, dándole una integridad y familiaridad visual a los armados de bloques.

En la figura 2.19 podemos ver que el bloque de objeto cambió su icono principal, y, al definir una propiedad, se cambiaron los dos puntos por una flecha. Podemos observar un cambio de icono también en el bloque de envío de mensaje, además de su desdoblamiento entre mensaje que devuelve y que no devuelve ningún valor.

En cuanto al envío de mensajes, en la primera versión que generaba código Javascript, los métodos eran definidos externamente. Luego al definir objetos, cualquiera podía referenciarlo. Esto tenía sentido en el contexto de Javascript donde una función es un objeto más. Al hacer el cambio y comenzar a generar código Wollok, se adoptó la decisión de que los métodos se definen únicamente dentro de un objeto y se cambio este comportamiento.

La figura 2.20 muestra la evolución de los bloques incluidos en cada iteración, así como la forma de agruparlos. Destacamos como sumamente interesante la diferencia en cantidad de bloques entre las tres versiones. Inicialmente, se contaban con muy pocos bloques. El cambio de la primera a la segunda iteración marca un aumento significativo en la cantidad de bloques, para terminar con una cantidad menor en la tercera. Esto se explica por el hecho de que hay una importante diferencia entre escasez y síntesis. Para la segunda versión aparecieron bloques nuevos porque había metáforas que los bloques de la versión anterior no expresaban o lo hacían de manera no practica. Mientras que la reducción de bloques de la segunda a la tercera se explica por el mayor entendimiento que se fue formando de la herramienta y por ello se comprendió de mejor manera los bloques necesarios y la mejor manera de presentarlos al usuario, es decir, como agruparlos en secciones que brinden mayor coherencia.

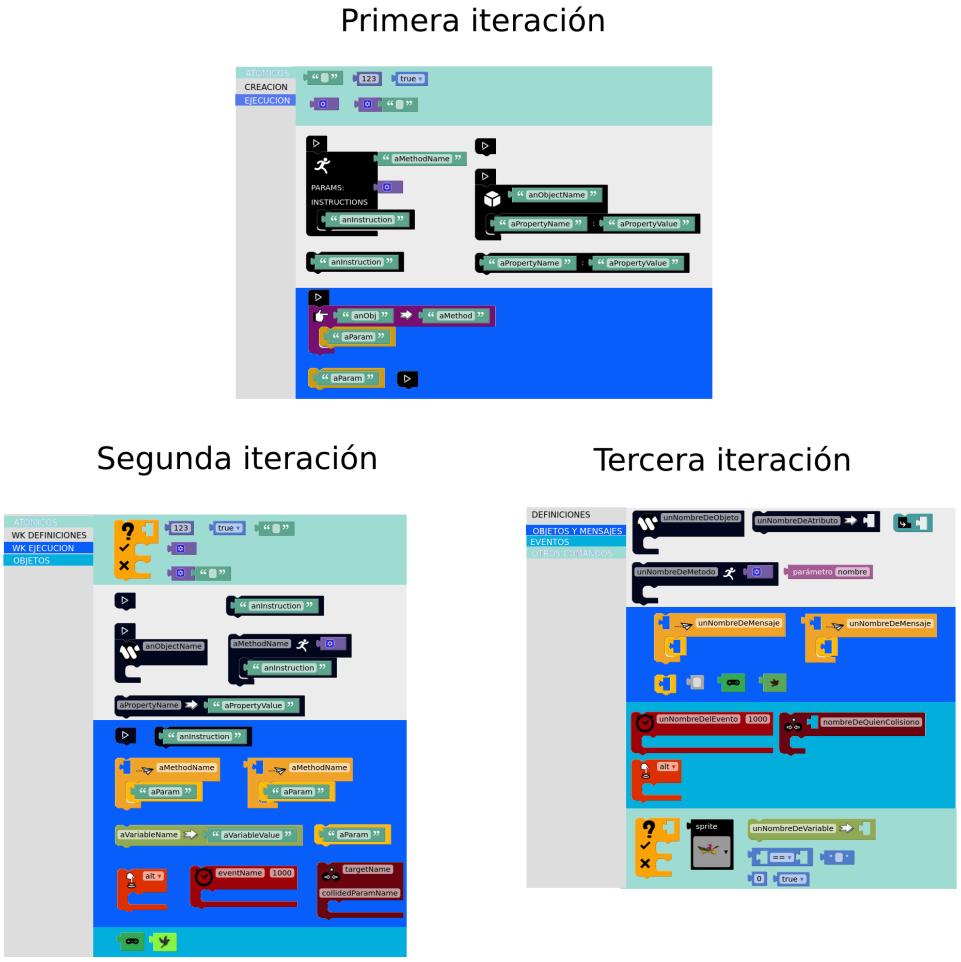


Fig. 2.20: Evolución de las secciones y bloques disponibles.

Con la conclusión de esta versión decidimos frenar el proceso de desarrollo por cuestiones del alcance de este trabajo, y habiendo quedado satisfechos con el estadío logrado, podemos pasar a analizar las pruebas realizadas para validar la herramienta.

3. RESULTADOS Y DISCUSIÓN

Antes de discutir la experimentación que fue planteada para poner a prueba nuestros objetivos iniciales, debemos describir brevemente cuál fue el fruto de este trabajo en términos de proyecto de software. Como ya se ha mencionado, se decidió disponibilizar el código fuente de la herramienta como código abierto, por lo cual se creo un repositorio público `https://github.com/alejandroFerrante/Woblocks/`. El desarrollo fue encarado como lo sería un trabajo profesional con aspiraciones de ser continuado y mantenido por cualquier interesado. El desarrollo fue separado en iteraciones donde cada una contenía una serie de requerimientos que eran implementados y marcados como desarrollados. Se creó un backlog que recopila todos los bugs encontrados en el código del que eran seleccionados los más cruciales para incorporar a las diferentes iteraciones. Esto hace la historia de desarrollo disponible a quien quiera investigarla, actualmente disponible en el proyecto Github asociado al repositorio `https://github.com/alejandroFerrante/Woblocks/projects/1`. El backlog también brinda una guía clara para quien quiera colaborar y continuar el desarrollo, que puede ser a partir de la creación de un *fork* y una *pull request* al repositorio, por parte de quien quiera implementar una funcionalidad nueva.

Al momento de escribir este trabajo el proyecto lleva más de 35 issues importantes ya cerrados.

Cabe destacar además que **WoBlocks** fue presentado en la Jornadas Argentinas de Didáctica de las Ciencias de la Computación (JADICC) en su edición 2022. El póster utilizado para esta presentación se encuentra en el apéndice[62].

Recordemos que nuestro objetivo principal es el diseño e implementación de una herramienta que permita enseñar desde cero conceptos clave de la POO de manera efectiva e intuitiva. La última versión de la misma satisface nuestras expectativas, pero carece de un paso fundamental en cualquier trabajo: la validación de su funcionamiento con usuarios externos.

Es necesario poder ver, aunque sea en una muestra limitada, cómo miembros del público objetivo o similar reaccionan al uso de la herramienta y si esta cumple (y en qué medida) con el objetivo para el cual fue creada.

3.1. Diseño de la experimentación

Para eso propusimos crear una consigna que podría ser utilizada como asignación final de un curso básico inicial sobre POO (y el uso de la herramienta). Esta consiste en producir un videojuego que imite la mecánica principal del título "Flappy Bird", tal y como se utilizó para la prueba inicial. En este juego, el personaje controlado por el jugador está constantemente cayendo, y la acción del jugador lo impulsa hacia arriba. La frecuencia y cantidad de impulsos verticales hacia arriba es la que determina la posición del personaje en todo momento. En nuestra versión del juego, el jugador debe esquivar los obstáculos que se van presentando, acumulando así puntos. Proveemos al sujeto de prueba con tres documentos: La consigna del ejercicio, una breve presentación del uso de la herramienta y un formulario donde deberá volcar su experiencia, además de obviamente entregar el archivo del proyecto terminado.

El formato de cuestionario empleado es el NASA-TLX [63]. Este se trata de una me-

todologías desarrollada por el Ames Research Center de la nombrada agencia aeroespacial a fines de la década de 1980. Constituye un método subjetivo que propone un procedimiento de valoración de la carga mental de un individuo haciendo uso de un valoración multidimensional. NASA-TLX considera los siguientes aspectos: Demanda Mental, que insta al evaluado a puntuar el esfuerzo mental requerido para completar la o las tareas realizadas. Demanda física, que evalúa el esfuerzo físico realizado, y la demanda temporal, que refiere a la propia percepción del tiempo durante la realización de la tarea. Más allá del tiempo real requerido, este aspecto se centra en el aspecto temporal percibido. Si una tarea resulta tediosa, tiende a percibirse como más duradera, mientras que cuando la tarea atrae y atrapa a su realizador parece que su compleción tarda menos del tiempo que en realidad se requirió. Otros aspectos son el rendimiento, que refiere a la percepción del propio desempeño y satisfacción con la performance al efectuar la tarea; el esfuerzo y el nivel de frustración. El método originalmente propone una escala de 1 a 5 para evaluar cada aspecto. Sin embargo, en nuestro caso decidimos ampliar este espectro a una escala del 1 al 10 para lograr mayor detalle en las respuestas.

Adjuntamos en la sección del apéndice estos documentos.

3.2. Resultado obtenido

A continuación analizaremos las respuestas obtenidas. Los ya mencionados cuestionarios fueron entregados a cuatro docentes con amplia experiencia en el campo educativo, sobre todo lo referido a la enseñanza de POO utilizando el lenguaje Wollok.

En términos generales, consideramos que los resultados fueron sumamente positivos. Si bien la herramienta se encuentra en un estado inicial, ha servido para cumplir su cometido. Cuatro personas sin conocimiento previo de la misma lograron completar una tarea razonablemente compleja para un usuario primerizo. Y el resultado de este proceso se trata nada más y nada menos que un video juego funcional.

Los indicadores numéricos arrojaron un promedio de tiempo para completar la tarea de una hora. Consideramos esta cifra satisfactoria ya que de presentar la herramienta problemas graves de usabilidad, este indicador sería mas alto o nos podríamos haber encontrado con la imposibilidad de completar la tarea. En cuanto a los indicadores de las distintas dimensiones de exigencia (representando 0 a una tarea fácil y 10 una difícil), nos encontramos con un promedio de 5,5 para la exigencia mental y 7,25 para la exigencia física. Atribuimos este ultimo indicador a la naturaleza del trabajo con bloques, la cual exige del usuario manipulaciones del tipo “arrastrar” y “encastrar”; en contraposición a un trabajo mas estático como es el simple typeo de texto. Resaltamos además que la falta de acostumbramiento puede aportar a incrementar la sensación de esfuerzo físico. El esfuerzo mental por otro lado se encuentra en los valores medios y lo evaluamos como esperable para el tipo de consigna planteada para el nivel poseído por quienes realizaron las pruebas. El indicador de exigencia temporal se encontró en 4,75 en una escala donde 0 constituía un ritmo rápido y 10 un ritmo relajado. En este caso el valor más esperable es el valor medio, el cual fue prácticamente alcanzado en promedio.

Similares a los valores obtenidos en el esfuerzo mental y físico, el indicador general de esfuerzo se encontró en promedio en 5,75. Nuevamente y por las razones expuestas anteriormente, consideramos satisfactorio esta cifra.

Podemos argumentar, adicionalmente, que la comparación entre un IDE profesional y nuestra herramienta en cuanto a la experiencia de uso puede ser engañosa. Atribuimos

esto a que las primeras cuentan con un largo camino recorrido , y por ello cuentan con muchos mecanismos diseñados específicamente para agilizar la experiencia (ventaja de la cual carece nuestra herramienta en su etapa inicial). Esto se vuelve mucho mas marcado tratándose de usuarios experimentados como lo son los sujetos de prueba escogidos. Probablemente un usuario principiante que tenga que utilizar por primera vez una herramienta de desarrollo para lenguajes basados en texto que carezca de funcionalidades como las sugerencias o la identificación de errores sintácticos tendría una experiencia mucho mas difícil.

Sin duda alguna los indicadores mas satisfactorios arrojados por las pruebas realizadas son el de rendimiento (si el realizador siente que ha tenido éxito en su tarea y la ha realizado correctamente) y el de disfrute, que en promedio alcanzaron 8,25 y 8,75 respectivamente. Valores de esta talla apuntan al buen rumbo en términos generales de diseño e implementación de nuestra propuesta, ya que, como indican diversas fuentes [64, 65, 32, 39, 10, 66, 36], estas dos sensaciones son esenciales para un proceso de aprendizaje mas efectivo (a esto podemos sumar el hecho, también indicado por las citadas fuentes, de que se está trabajando con videojuegos). En contraposición, encontramos el indicador de frustración. De este se obtuvo un promedio de 5,5. Podemos destacar que dados los dos indicadores anteriormente mencionados, se esperarían valores más bajos. Sin embargo reconocemos que al encontrarse la herramienta en un estado temprano del desarrollo, todavía cuenta con diversos puntos de mejora en cuanto a la experiencia de usuario y usabilidad. Creemos que estos factores podrían aportar a que la sensación de frustración se acrecente.

Según lo manifestado por quienes la probaron, nuestra herramienta les parece adecuada especialmente para niveles iniciales de aprendizaje como lo son el primario y secundario, y como docentes recomendarían su uso en este contexto. También destacan la potencialidad de la herramienta en aquellos que presenten problemas de lecto-escritura y la practicidad de la herramienta en contextos educativos donde los alumnos no cuentan con acceso a una computadora pero si lo tienen a dispositivos móviles, ya que es una aplicación web.

Entendemos sin embargo que para niveles más avanzados como el universitario, según lo que resulta de las experiencias analizadas, todavía es necesario afianzar de mejor manera algunos aspectos importantes como lo son entre otras cosas el refinamiento de las metáforas de los objetos.

Presentamos ahora un resumen más detallado de los puntos de mejora que logramos identificar a partir de las devoluciones de los sujetos de prueba.

3.2.1. Cuestiones de usabilidad básicas

Sin duda alguna uno de los mayores puntos de mejora identificados por los docentes respecta al feedback que la herramienta provee a los usuarios. Si bien entendemos que en el contexto de las pruebas realizadas quienes testearon la herramienta lo hicieron sin poseer explicación alguna del funcionamiento de la misma, y que esto no sucedería de igual manera en ningún contexto educativo, esta forma de presentarla resultó útil para poder ver el nivel de auto expresividad de la misma. En este sentido pudimos comprobar que la herramienta debe hacer un mejor trabajo en comunicar al usuario en qué estado se encuentra, qué cosas puede hacer y sobre todo si surge algún error o la herramienta no está siendo utilizada correctamente.

En esta misma línea, otro de los puntos interesantes tuvo que ver con buscar que las interacciones estén mas apuntadas a algo familiar al paradigma de objetos. Propuestas como que desde el objeto se pueda generar un bloque de envío de mensaje, y este au-

tomáticamente permita elegir luego los posibles mensajes en vez de tener que escribirlo, marcan una clara dirección hacia un ambiente más inteligente y autoconsciente que sin duda llevaría la herramienta a un nivel superior.

Refiriéndonos a puntos de mejora que teníamos identificados y fueron ratificados por los resultados obtenidos, otro punto de mejora identificado fue la definición de parámetros al definir un mensaje. Actualmente este bloque posee un input de texto para definir el nombre del método y una ranura donde solo puede encastrarse un bloque de tipo lista. Este es un bloque estándar de Blockly con el cual primero hay que agregarle ranuras y luego encastrarle otro bloque especifico de parámetro el cual permite definir el nombre del parámetro. Entendemos este proceso como poco intuitivo y algo tedioso y así lo entendieron también quienes probaron la herramienta. Apuntamos a desarrollar a futuro una solución similar a la implementada por Pilas Bloques, tal como lo muestra la figura 3.1, donde el bloque puede definir dinámicamente la cantidad de parámetros a ser utilizados.

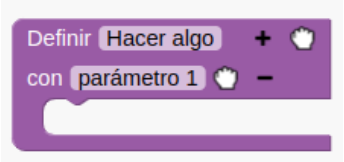


Fig. 3.1: La forma de resolver el pasaje de parámetros de Pilas Bloques.

3.2.2. Metáforas importantes

Otro bloque que causó alguna confusión en los sujetos de prueba fue el bloque de envió de mensaje. Actualmente contamos con dos variantes del mismo. El primero representa un envió de mensaje dentro de un hilo de ejecución, por lo que es encastrable verticalmente y se supone que no devuelve ningún valor. El otro es encastrable a izquierda ya que representa el resultado del envió del mensaje. Esta división no fue del todo comprendida a priori y esta metáfora probablemente deba ser reconsiderada. Una posible propuesta es que se trate de un único bloque mutable (osea que el bloque permita modificar el tipo de encastre).

3.2.3. Capacidades

En esta versión de la herramienta hemos permitido que en la escena principal se puedan crear objetos como se haría en cada tab individual. La inclusión de este feature fue ampliamente debatido ya que si bien otorgaba una manera sencilla de prototipar objetos, también podía aportar confusión al usuario por tener más de una manera de definir un objeto. Los resultados obtenidos aportaron a esta cuestión donde a algunos les pareció innecesario mientras que otros no lo identificaron como algo negativo. No podemos aún sacar conclusiones definitivas a cerca de esta cuestión.

3.2.4. Adiciones

Dentro de las soluciones que se nos fueron entregadas, todas hicieron uso de las colisiones para detectar cuando un obstáculo entra en contacto con el ave que controla el jugador. El bloque de colisión les presentó problemas por su falta de claridad visual, por lo

que nos propondremos como punto de mejora hacerlo mas intuitivo para su uso. También fue indicada la necesidad de un bloque self así como existe un bloque para el objeto Game, ya que es una pseudovariable importante en el lenguaje Wollok y en otros lenguajes de objetos.

3.2.5. Cambios superficiales de la herramienta

Finalmente, incluimos sugerencias de cambios menores que representan pequeños cambios en la interfaz general:

1. Poder ver el código generado en la sección de objeto individual.
2. Mejorar el sistema de guardado.
3. Tener la capacidad de ejecutar sin tener que volver a la escena principal.

Tomando en consideración por último las soluciones provistas por los colaboradores, destacamos nuevamente como éxito el hecho de que en tan solo sesenta minutos de haber sido introducidos por primera vez a la herramienta hayan podido producir un juego de video completamente funcional. Estas respuestas no solo generan el resultado esperado, sino que destacamos que estas resultan sumamente sencillas de leer y entender, así como minimalistas en su armado. Consideramos que se ha logrado en todos los casos un comportamiento de cierta complejidad con un número relativamente pequeño de bloques y configuraciones, tal como lo muestra la figura 3.2. Los cuatro sujetos de prueba consideraron un mismo conjunto de objetos que resuelven el problema ante ellos presentado: Un pájaro, un obstáculo y en el programa principal, un evento de teclado para hacer subir al pájaro, dos eventos temporales para accionar la gravedad y el movimiento de los obstáculos y un evento de colisión.

Todo lo arrojado de este ensayo preliminar resulta sumamente valioso, sobre todo por la calidad de los sujetos de prueba y su enorme entendimiento tanto del paradigma como de la didáctica en programación. Muchos de los puntos de mejora puntualizados confirman sospechas que quienes desarrollamos esta primera versión ya guardábamos.

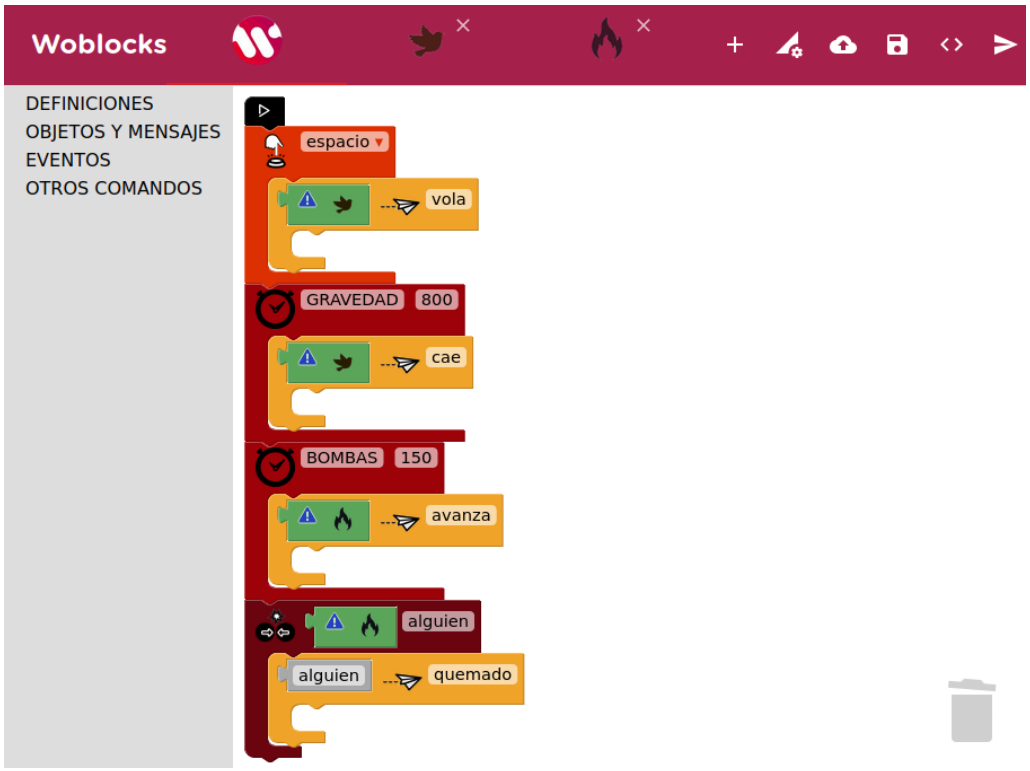


Fig. 3.2: Ejemplo de solución provista por uno de los sujetos de experimentación.

4. CONCLUSIONES

En el trascurso de este trabajo hemos recorrido un largo camino de desarrollo e interesantes discusiones a cerca de la didáctica en POO y la mejor manera de plasmar dichas ideas de una manera intuitiva. Partimos del análisis del estado actual de las diferentes propuestas para enseñar este paradigma y destilamos las características principales deseables (simplicidad, facilidad de uso, interactividad, gratuidad y bajos requerimientos de instalación, entre otros) y los conceptos centrales que queríamos enseñar (objetos, métodos, envío de mensajes, etc.).

Comenzamos familiarizándonos con Blockly y experimentando con los bloques, su funcionamiento y lógica interna. Con una versión preliminar logramos crear unos primeros bloques que se tradujeran en código JavaScript. Con ellos podían definirse objetos, métodos y envío de mensajes. En esta versión, ilustrada en la figura 2.1, podemos ver claramente el exceso de texto, así como una variedad desarticulada de colores. Para una primera versión propiamente dicha, planeamos una interfaz propia con acceso únicamente a los bloques esenciales, donde se puede ver en todo momento el código generado y se pueden crear objetos y métodos accesibles a través de tabs. La figura 2.6 muestra un ejemplo integral de las capacidades de esta instancia.

Más adelante se tomó la decisión de incorporar Wollok. Esto fue sin duda un punto de inflexión ya que con esta decisión se adoptó no solo el lenguaje en sí, sino que también toda una filosofía, una didáctica y una serie de decisiones que así como nuestro objetivo inicial, apunta a una forma efectiva de enseñar objetos que además es utilizada en la actualidad en las aulas.

Es así que surgió una nueva versión que entre otras cosas generaba ahora código Wollok. Adicionalmente se replantearon los bloques existentes y se intentó reducir al mínimo el uso de texto en ellos. Luego de algunas iteraciones se llegó a una herramienta funcionalmente bastante completa, pero que dejaba qué desear en el aspecto visual y de usabilidad, tomando en cuenta el objetivo propuesto de ser una herramienta moderna y atractiva, utilizada en las aulas y fácilmente mantenible y expandible. Por estas razones se construyó una nueva versión utilizando el framework ReactJS.

Para diseñar las pantallas se utilizó la técnica de paper prototyping, logrando una propuesta de usabilidad nueva y más intuitiva.

Habiendo quedado conformes con el resultado obtenido, nos propusimos poner a prueba la herramienta . Cuatro voluntarios, docentes sin conocimiento previo de la herramienta con experiencia enseñando objetos a través de Wollok, fueron dados una consigna que les proponía utilizar **WoBlocks** para crear un juego básico.

Los resultados fueron sumamente satisfactorios ya que todos lograron cumplir la consigna de manera efectiva. Adicionalmente, la experiencia de uso de la herramienta arrojó valiosísimo feedback en cuanto a diferentes aspectos como lo son las capacidades didácticas, y facetas funcionales y de usabilidad.

Para recolectar información a cerca de la experiencia de los usuarios en el uso de la herramienta, se utilizó el modelo NASA-TLX que entre otras cosas mide el esfuerzo físico y mental percibido de los sujetos de prueba. Analizando estos resultados, pudimos observar que todos los valores que miden el esfuerzo se encontraron en general en valores medios, a excepción del esfuerzo físico que la herramienta demanda. Atribuimos esto a que

typear texto es una tarea absolutamente familiar y que cualquier usuario puede realizar con suma facilidad, mientras que la tarea de arrastrar y encastrar bloques es naturalmente mas mecánica y carece de dicha familiaridad por parte de los usuarios. En cuanto a otros indicadores que miden la satisfacción, rendimiento y disfrute derivados de la utilización de la herramienta, estos valores se encontraron por encima de los 8 puntos en promedio.

Es por esto que podemos afirmar que hemos creado una herramienta que cumple con los objetivos iniciales. Tiene relativa baja complejidad y facilidad de uso, utiliza metáforas visuales y animaciones, es interactivo, tiene un uso mínimo del texto para comunicar funcionalidad, y corre sobre un browser, lo que lo hace sencillo de comenzar a utilizar, además de ser gratuito y de código abierto. En cuanto a su utilidad para enseñar objetos, si bien reconocemos que hay trabajo pendiente, podemos afirmar basándonos en el feedback y tomando en consideración el entendimiento profundo en didáctica de los sujetos de prueba que la herramienta cuenta con mucho potencial.

Esta propuesta que toma como punto de inicio el extenso trabajo realizado a nivel nacional con identidad propia en didáctica y provee a Wollok con la posibilidad de aprender a pensar con objetos mientras se construye un videojuego, valiéndose únicamente de metáforas visuales constituye un importante paso hacia adelante en esta área.

Insistimos que si bien esta herramienta todavía requiere desarrollo, es nuestra intención verla en las aulas siendo utilizada por los alumnos para que los importantes conceptos del paradigma se fijen sólidamente en ellos sin la necesidad asociarlos a una codificación textual.

5. TRABAJO FUTURO

Destacamos como lo más importante en cuanto a trabajo futuro el construir una estrategia pedagógica asociada a la herramienta que hemos construido, basada en la estrategia existente de Wollok. Como ya dijimos, creemos que esta tiene todo el potencial para ser usada de manera efectiva en la instrucción de los conceptos de POO a audiencias jóvenes que no posean experiencia previa en el campo de la programación.

Una vez desarrollada esta, el paso natural siguiente será llevar **WoBlocks** a las aulas. Esto pondrá a prueba todas nuestras suposiciones y marcará el camino para identificar todos los puntos de mejora que nuestra herramienta necesita para ser lo más efectiva posible en su tarea.

Luego de esta primera versión sabemos que muchas cosas se pueden agregar y mejorar para que nuestra propuesta crezca en cuanto a sus capacidades. Es nuestra intención que así sea.

A continuación resumiremos los puntos provenientes de los resultados de las pruebas piloto. El punto principal en el que identificamos debemos trabajar es en mejorar el feedback que la herramienta provee a los usuarios, un principio fundamental de la usabilidad.

Otros cambios propuestos van en dirección de que los bloques sean más dinámicos. Por ejemplo que desde un bloque de objeto autodefinido se pueda generar un bloque de envió de mensaje con dicho bloque como receptor. En la misma línea, se propuso que cuando un bloque de mensaje posea el dinamismo de una vez se haya encastrado en él un bloque receptor, el bloque de envio de mensaje genere una lista de selección con los mensajes posibles en vez de tener que ingresarlo como texto.

Algunos bloques no resultaron del todo claros, como el de envío de mensaje que posee dos variantes: uno encastrable verticalmente que representa una sentencia, y otro encastrable a izquierda que representa un envío que arroja un resultado. Se propuso se cuente con un único bloque que pueda mutar entre una versión y otra.

El que más confusión trajo sin duda fue el bloque que define los parámetros que un mensaje recibe. Actualmente se utiliza el bloque nativo de Blockly de tipo lista. Resulta claro que debemos avanzar hacia una solución similar a la utilizada por Pilas Bloques donde el bloque puede agregar dinámicamente parametros. También se comentó sobre la falta de claridad del bloque colisión sobre el que habrá que trabajar.

Otras sugerencias incluyen la capacidad de ver el código generado por un objeto individual cuando se encuentra el usuario en su tab correspondiente, mejor sistema de guardado y tener la capacidad de ejecutar el programa desde cualquier pantalla (es decir, incluso estando en una tab de objeto).

En paralelo con las propuestas de los candidatos, los desarrolladores identificamos otras como dotar a la herramienta con capacidades de debug. Identificamos esta como muy necesaria para el proceso de aprendizaje. También se consideraron características como poder clonar una tab, contar con un bloque de posición y mejorar el sistema de guardado. Por último parte del trabajo futuro además de expandir las capacidades con las que ya se cuentan es solucionar la deuda de bugfixing que acumula la herramienta en su estado actual.

En cuanto al acceso al código, **WoBlocks** ya está disponible como un proyecto de código abierto para que cualquier interesado pueda verlo y proponer mejoras al mismo.

Como punto inicial creemos que en todo lugar donde hoy en día se esté utilizando Wollok, se puede utilizar **WoBlocks**.

Para ello, creemos que más allá de las mejoras que consoliden nuestra propuesta inicial, es fundamental incorporar a **WoBlocks** la capacidad de enseñar el concepto de clase. Esto es algo que tanto Wollok como cualquier educador en POO (nosotros incluidos) entiende como concepto fundamental a ser incluido en cualquier currícula y es algo que nuestra herramienta hoy en día carece de la capacidad de ilustrar.

No queremos tampoco dejar de aprender del camino recorrido por propuestas similares. Scratch posee actualmente por ejemplo la capacidad de que el alumno dibuje sus propios sprites o modifique existentes, entre otras capacidades sumamente interesantes.

Suponemos que una feature sumamente útil asumiendo un uso generalizado de la herramienta es poder importar y exportar paquetes de objetos pre-existentes para poder ahorrar trabajo y focalizarlo según se requiera.

Otro concepto a explorar es que **WoBlocks** sirva no solo para videojuegos, sino que pueda generalizarse para ilustrar gráficamente al ejecutar la escena la interacción de los objetos, incluso utilizando animaciones. Por ejemplo, incluyendo el diagrama dinámico que ya posee Wollok [55].

Apéndice

A. CONSIGNA

Antes de empezar

Revisar la presentación de la herramienta en estas slides

La aplicación se encuentra en https://alejandroferrante.github.io/Woblocks/

Flappy WoBlocks

Se busca hacer un juego con la mecánica de control del conocido “Flappy bird” en el que nuestra ave va cayendo sola y la forma de levantarla es “aletear” (apretando la barra espaciadora). El ave se encuentra fija en la pantalla y su movimiento es únicamente en el eje vertical. En su vuelo se le van acercando obstáculos que surcan la pantalla de derecha a izquierda a velocidad constante, de una a la vez, en diferentes alturas (elegidas al azar). Cuando el ave choca contra un obstáculo, o el ave toca el piso, pierde y finaliza el juego.

Ejemplo de solución:

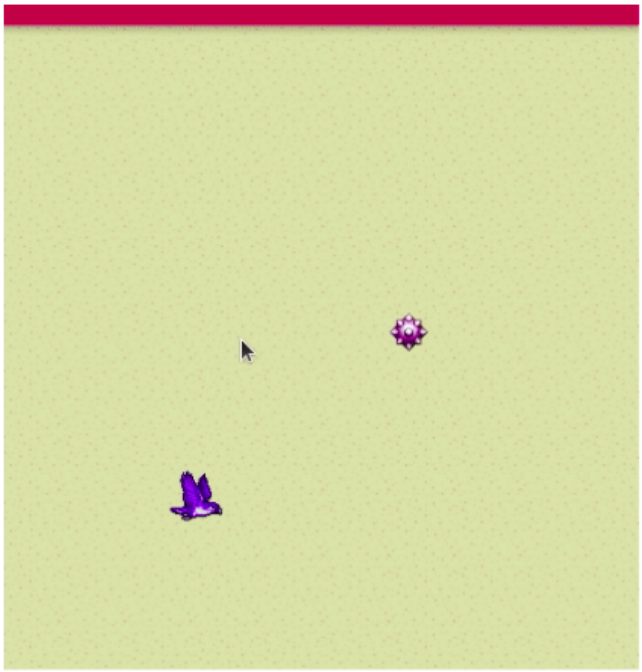


Fig. A.1: Consigna del ejercicio otorgado a los sujetos de prueba.

Breve introducción a WoBlocks

¿Qué es WoBlocks?

WoBlocks es una herramienta que sirve para construir programas de Wollok Game.

Permite definir objetos visuales y cualquier otro objeto necesario para lograr un comportamiento.

Programación con bloques

La programación con bloques consiste en utilizar, arrastrar, ensamblar, configurar y combinar bloques de manera visual de forma de obtener los comportamientos esperados. Tiene como ventaja que los objetos sólo encastran en lugares donde pueden ir, dejando de lado problemas de sintaxis y centrándose en lo semántico.

Pantalla inicial

La pantalla inicial cuenta con una barra de navegación superior y una barra de herramientas en la parte izquierda.

Sobre la barra de navegación aparecen pestañas con los objetos (inicialmente solo el juego) y botones.

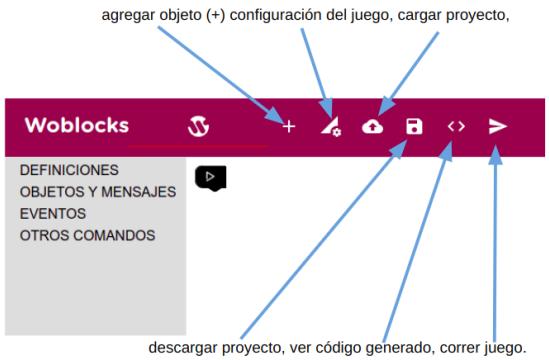


Fig. A.2: Slides mencionados en la consigna(1/3).

Consejo

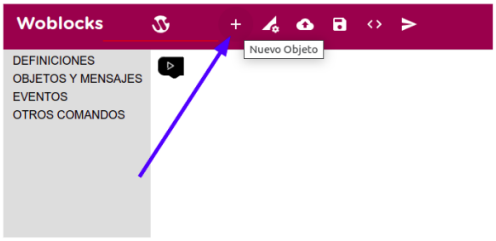
WoBlocks es una aplicación nueva y esta es la primera prueba. Recomendamos ir guardando el proyecto para no perder el avance si es que llegara colgarse la aplicación en algún momento.

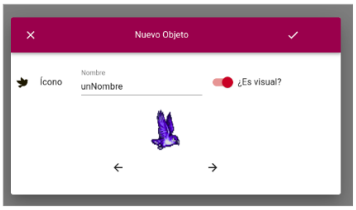
Flecha verde guardar, flecha azul subir un proyecto

Definiendo un objeto

Para definir un objeto se debe apretar el botón (+).

Al hacerlo se preguntará por el nombre del objeto y si el objeto tendrá representación visual en el juego (es decir tienen una posición y un sprite) o no.





Objeto nuevo

En la barra superior tendremos una nueva pestaña correspondiente al objeto recientemente creado

Fig. A.3: Slides mencionados en la consigna(2/3).

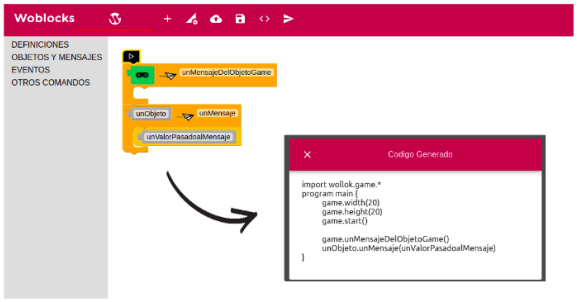
Atributos y métodos

Bajo la categoría “Definiciones” se encuentran los bloques para definir atributos y métodos que se pueden agregar a los objetos creados.

Tip: Si se está familiarizado con Wollok Game entonces puede probar arrastrar un bloque de atributo para agregarlo al objeto, luego uno bloque de método, volver a la pestaña del juego e inspeccionar el código generado para ver su equivalencia..

Mensajes

El envío de mensajes tiene un bloque asociado y en la figura se puede observar el equivalente en el lenguaje Wollok.



Eventos de Wollok Game

Se cuenta con tres bloques que representan eventos específicos de Wollok Game:

- Evento de Teclado
- Evento Temporal
- Colisiones

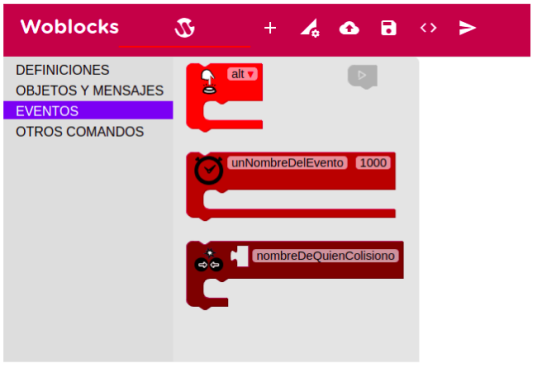


Fig. A.4: Slides mencionados en la consigna(3/3).

B. CUESTIONARIO PRESENTADO

Fig. B.1: Sección 1 del cuestionario

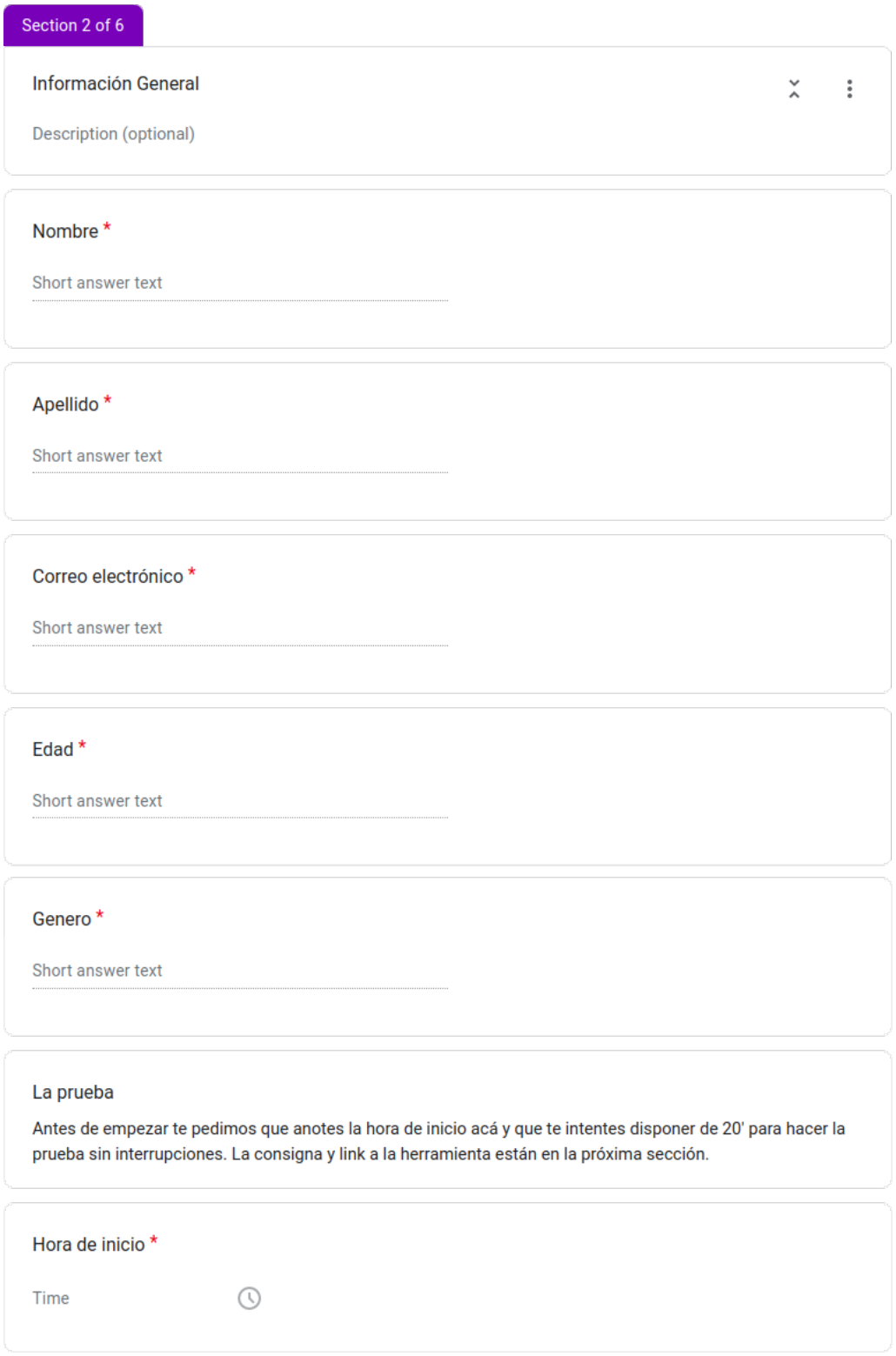


Fig. B.2: Sección 2 del cuestionario

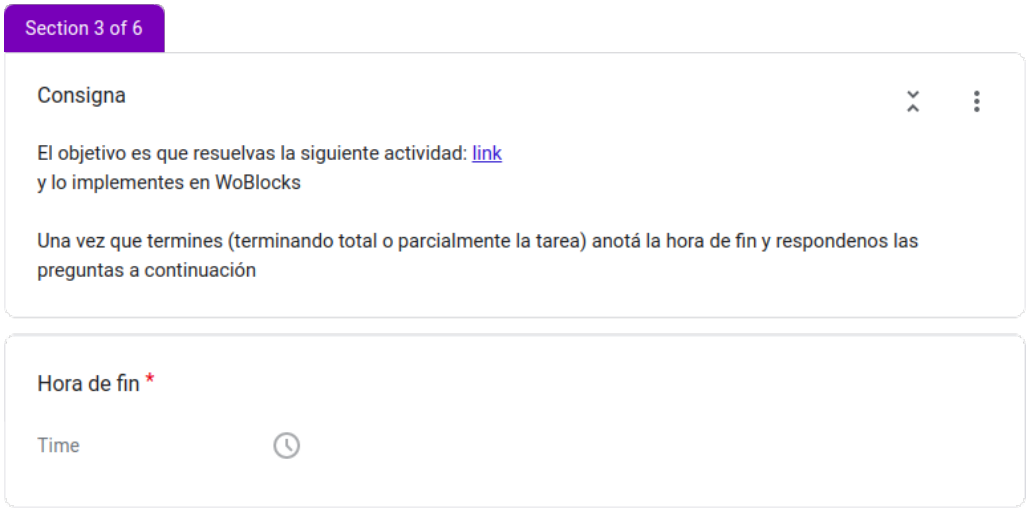


Fig. B.3: Sección 3 del cuestionario

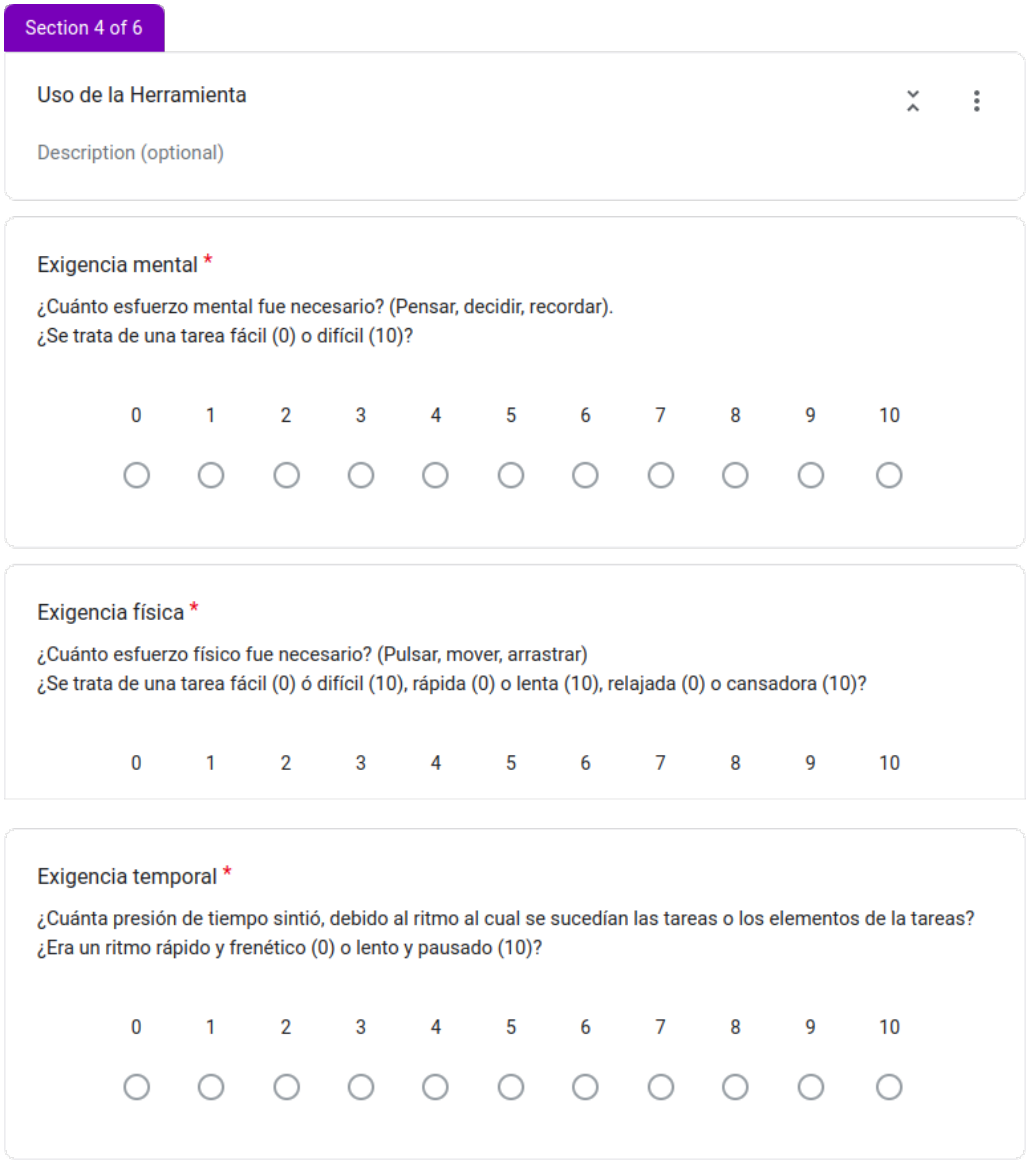


Fig. B.4: Sección 4 del cuestionario

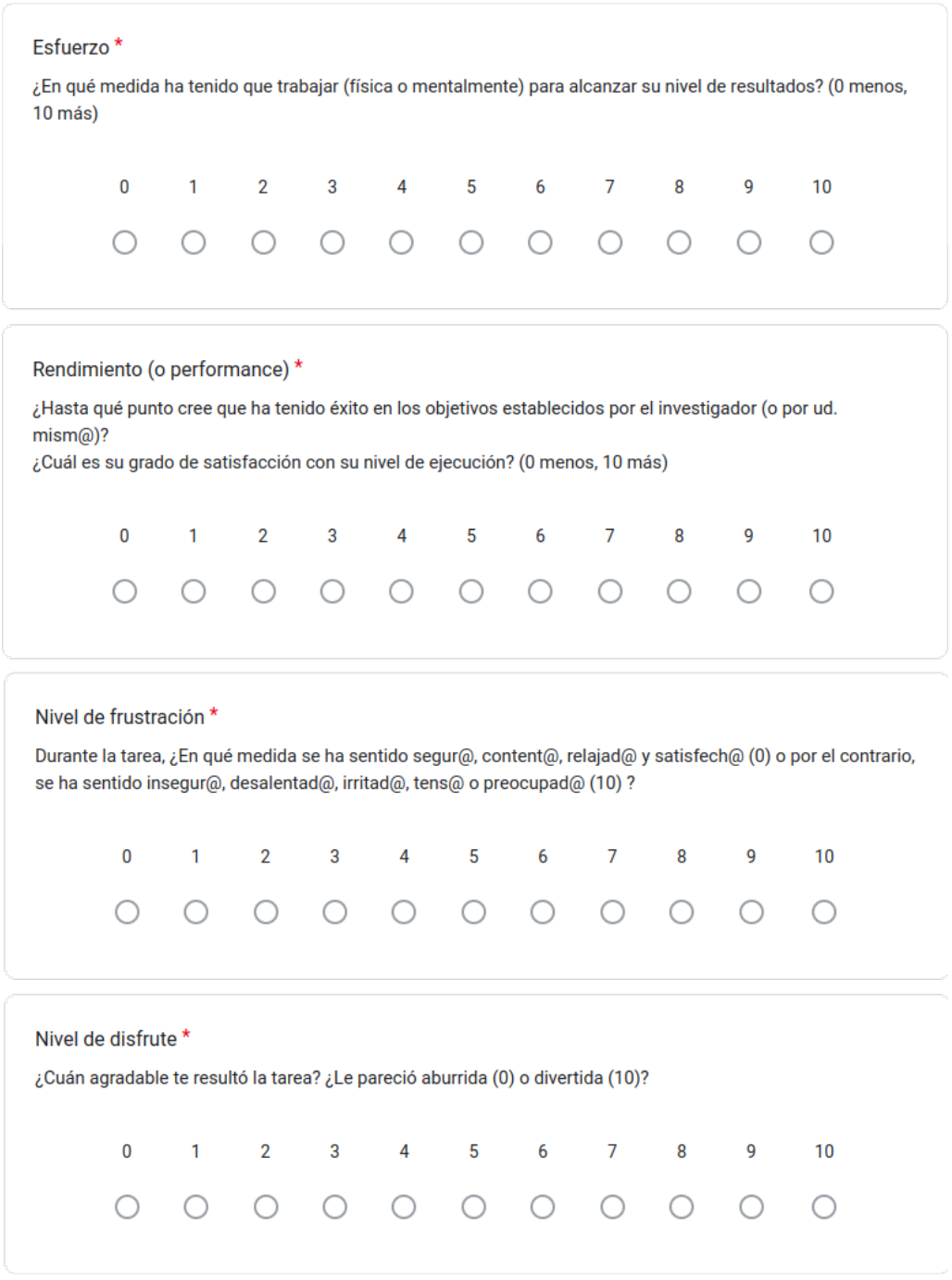


Fig. B.5: Sección 4 del cuestionario

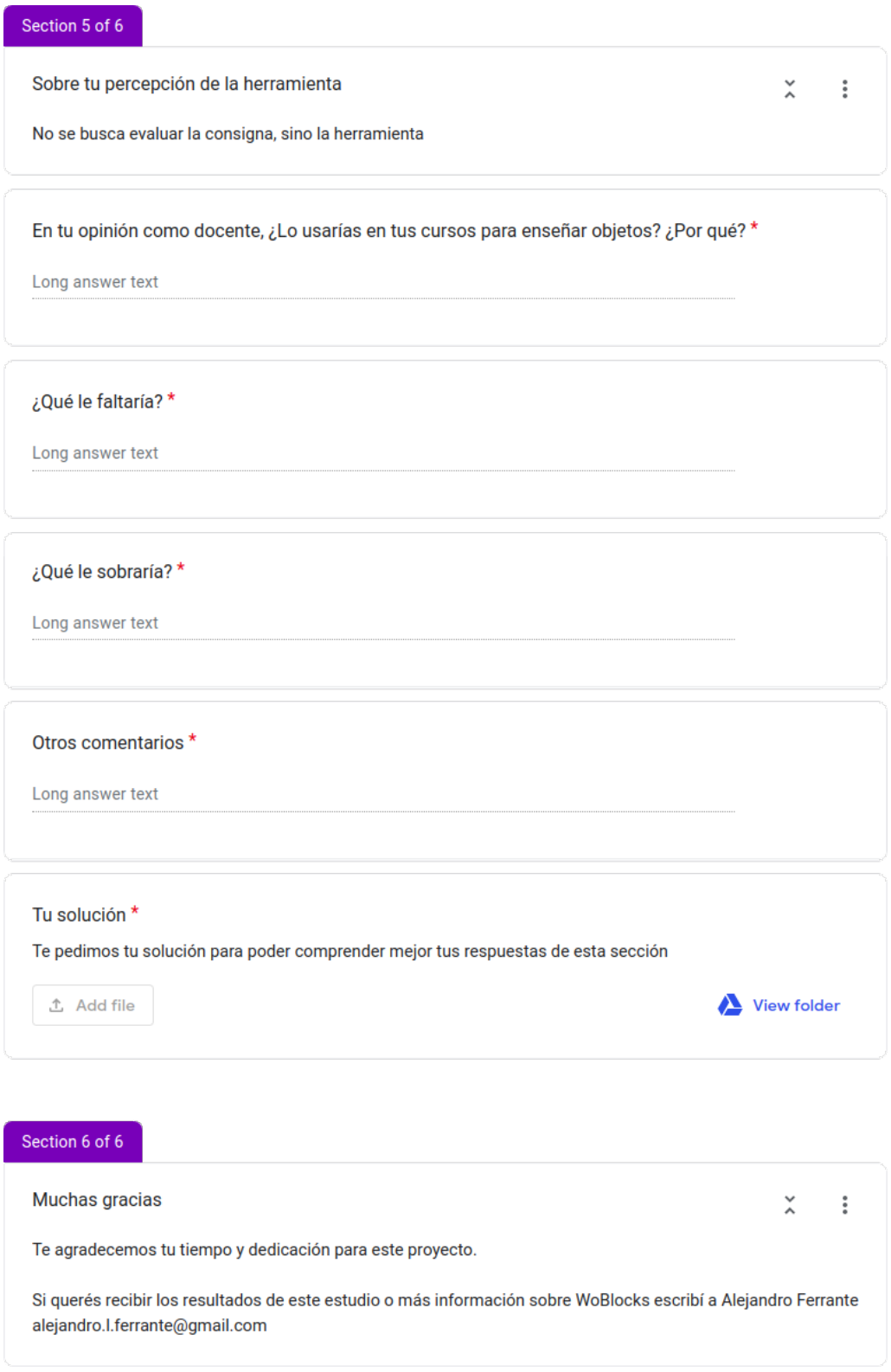


Fig. B.6: Sección 5 del cuestionario

C. RESULTADOS RECIBIDOS

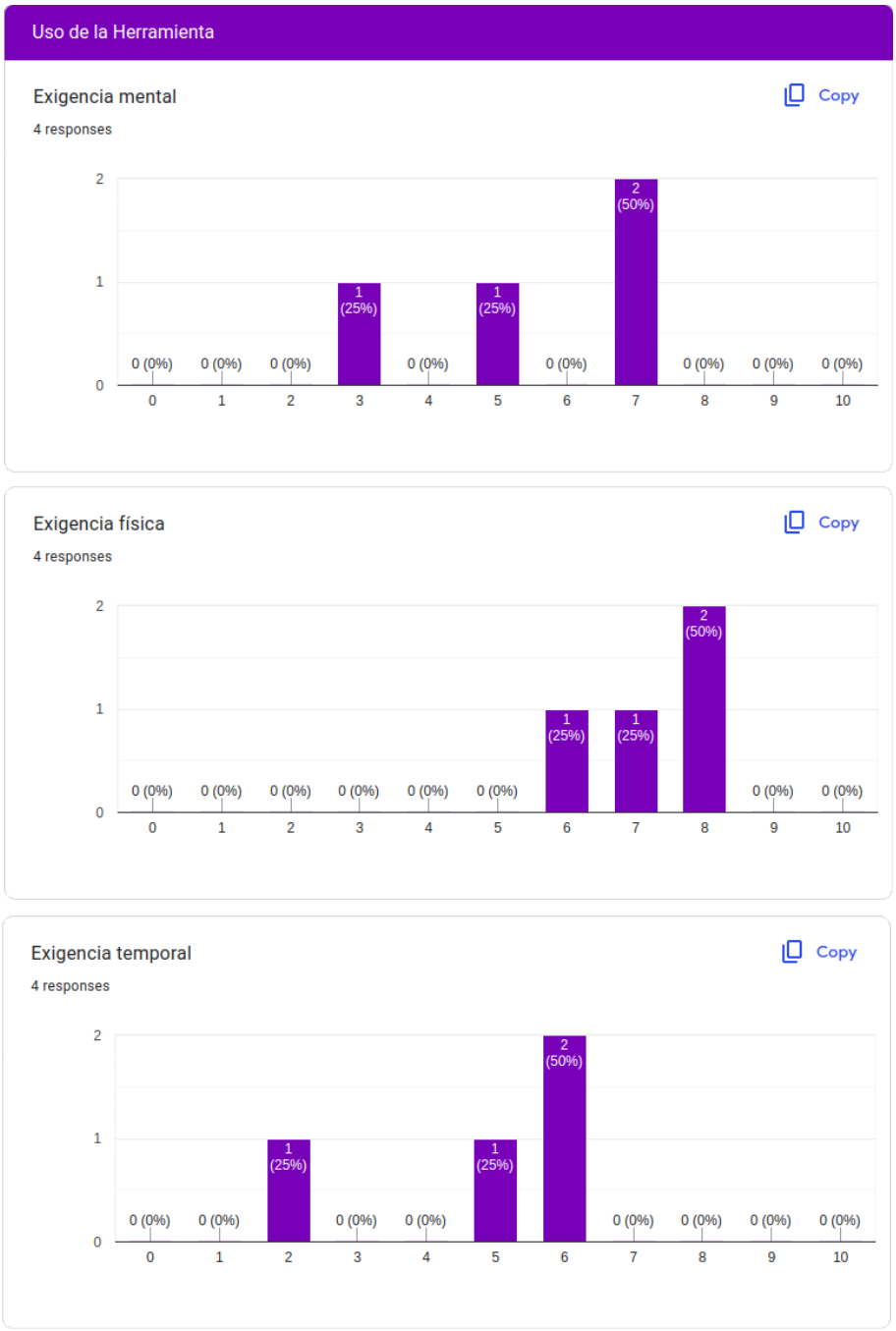
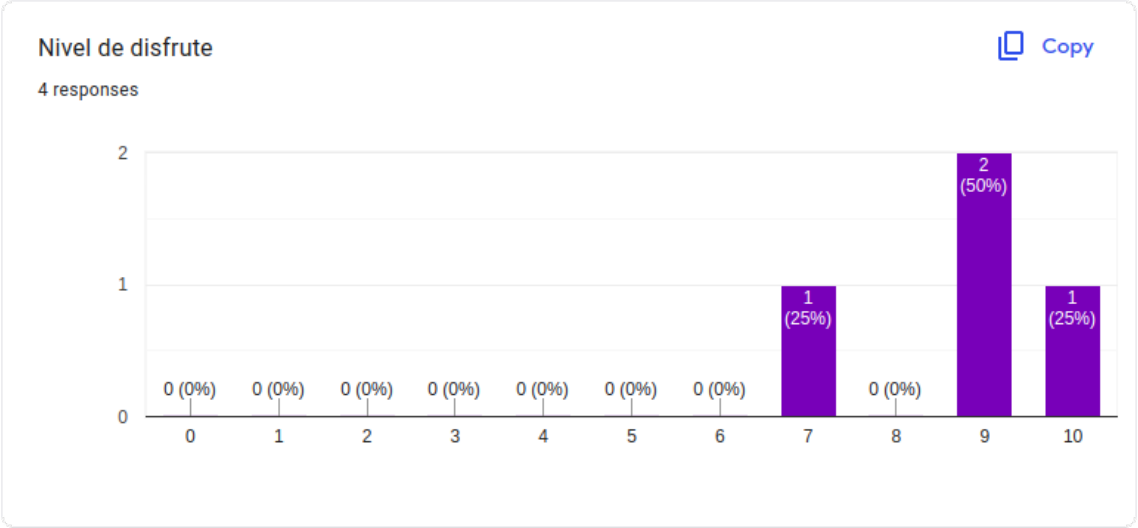


Fig. C.1: Resultados obtenidos

Fig. C.2: Resultados obtenidos



Sobre tu percepción de la herramienta

En tu opinión como docente, ¿Lo usarías en tus cursos para enseñar objetos? ¿Por qué?

4 responses

Para personas con problemas de lecto-escritura (que les cuesta o no pueden escribir código), sí lo usaría. Es la primer herramienta de bloques basada en el envío de mensajes que pruebo

La idea me encanta! Definitivamente coincido en que una forma de programación más visual es super valiosa en el aula. Me parece útil en particular en contextos donde los alumnos no tienen acceso a una notebook, pero sí a un celu/tablet. Usar un editor de texto en esos casos es muy difícil, y creo que una herramienta visual como esta puede ser una opción mucho más accesible.

Dicho eso, en el estado actual no la usaría, dada la falta de feedback que ofrece a quien lo usa

No, no me parece que proponga una forma de pensar objetosa, siento que está muy centrado en las operaciones (definir una variable, enviar un mensaje, etc) que en los objetos que están ahí y con los que me gustaría que se pueda interactuar.

Fig. C.3: Resultados obtenidos

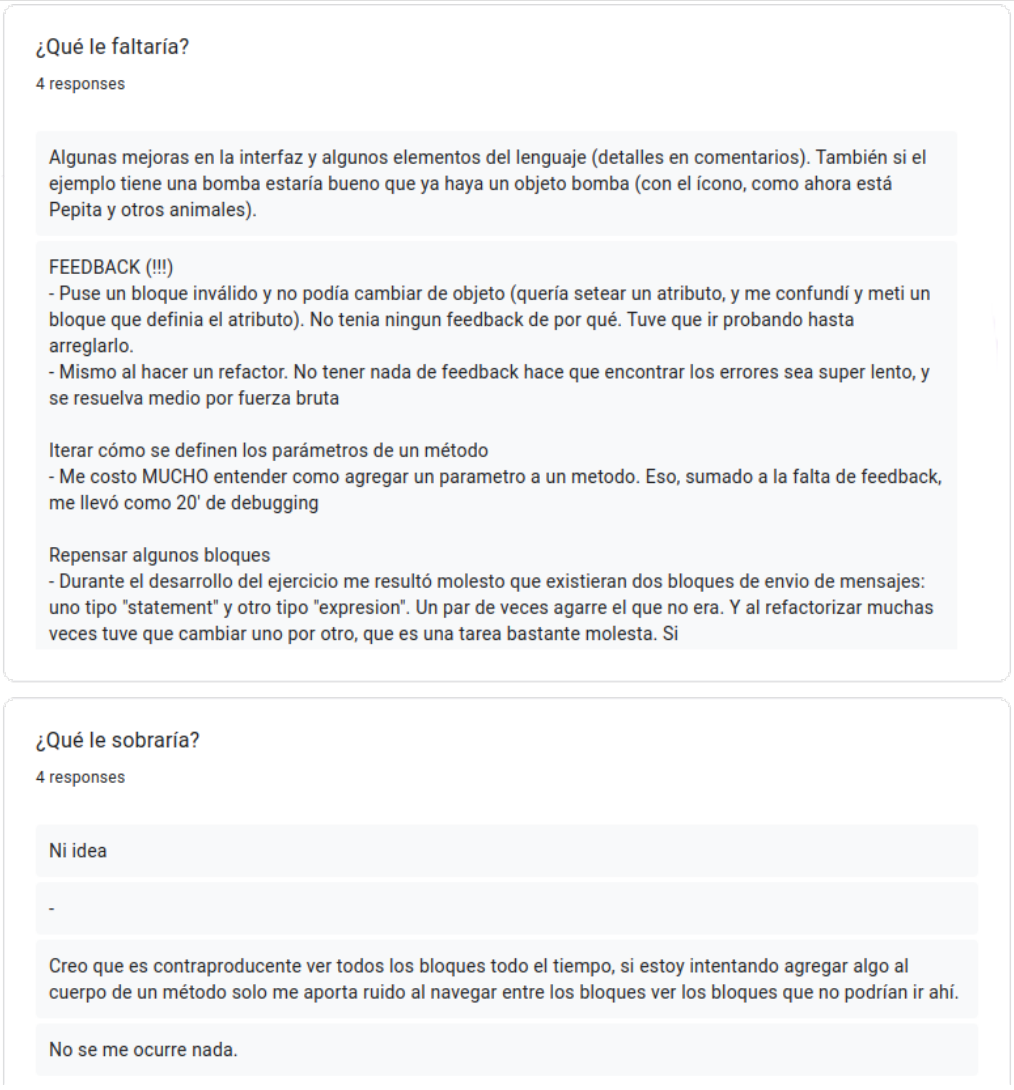


Fig. C.4: Resultados obtenidos

Otros comentarios

4 responses

- Poder ver el código en todas las pantallas.

- Diferenciar “mensaje con efecto” de “mensaje de consulta” en el nombre del mensaje por default.

- Cambiar el mensaje de error: “MENSAJES: ...” por “El objeto no entiende el mensaje <blah>” cuando corresponda.

- Sacar las definiciones del toolbox cuando estoy en el programa.

- El form para guardar un programa es molesto (después de la primera vez queda trulado). También s molesto sobreescribir el archivo... No se podrá guardar en el LocalStorage (automáticamente)?

Respecto al enunciado / tarea:
- A quiénes estará dedicado esto?

- Ya conoce WoBlocks?? Si no le pondría un poco más explicación (cómo se corre un juego, cómo se crea un objeto. Quizá con explicar qué hacen todos los botones de arriba basta)

- Ya usó alguna interfaz con bloques? Sino quizá haya que explicar algo. O quizá poner alguna descripción de qué representa cada bloque (Blockly creo que tiene algo para eso).

- La interfaz para colisiones no me resultó la más intuitiva. Creo que sería más fácil de entender si la colisión fuese siempre entre dos objetos (aunque eso por abajo se traduzca a un bloque que adentro tiene un if que chequea la identidad del colisionado)
- Estaria bueno poder scrollear para moverse por el espacio
- AMO que funciona copy paste de bloques (no le digan a Alf que dije esto :P)
- La experiencia refactorizando (concretamente, haciendo un "extract method") fue muy buena!!

Está re interesante la idea y la propuesta de tener una forma visual de programar en objetos, me encanta eso! Pero no estoy seguro si bloques me convence demasiado para objetos, al menos no como está planteado, me falta eso de poder tocar y hablar con los objetos en vez de ponerlos en cajas.
De paso, les paso un doc en donde estuve escribiendo lo que iba haciendo en WoBlocks (menos al final que creo que me frustré un poco así que me olvidé de ir anotando)
https://docs.google.com/document/d/1lGgwdeorGmpEUtsXGEPLIOPKUAAJTMa_3-GMqtErtUM/edit?usp=sharing

Está muy bueno, gran trabajo, felicitaciones!

Fig. C.5: Resultados obtenidos

D. SOLUCIONES

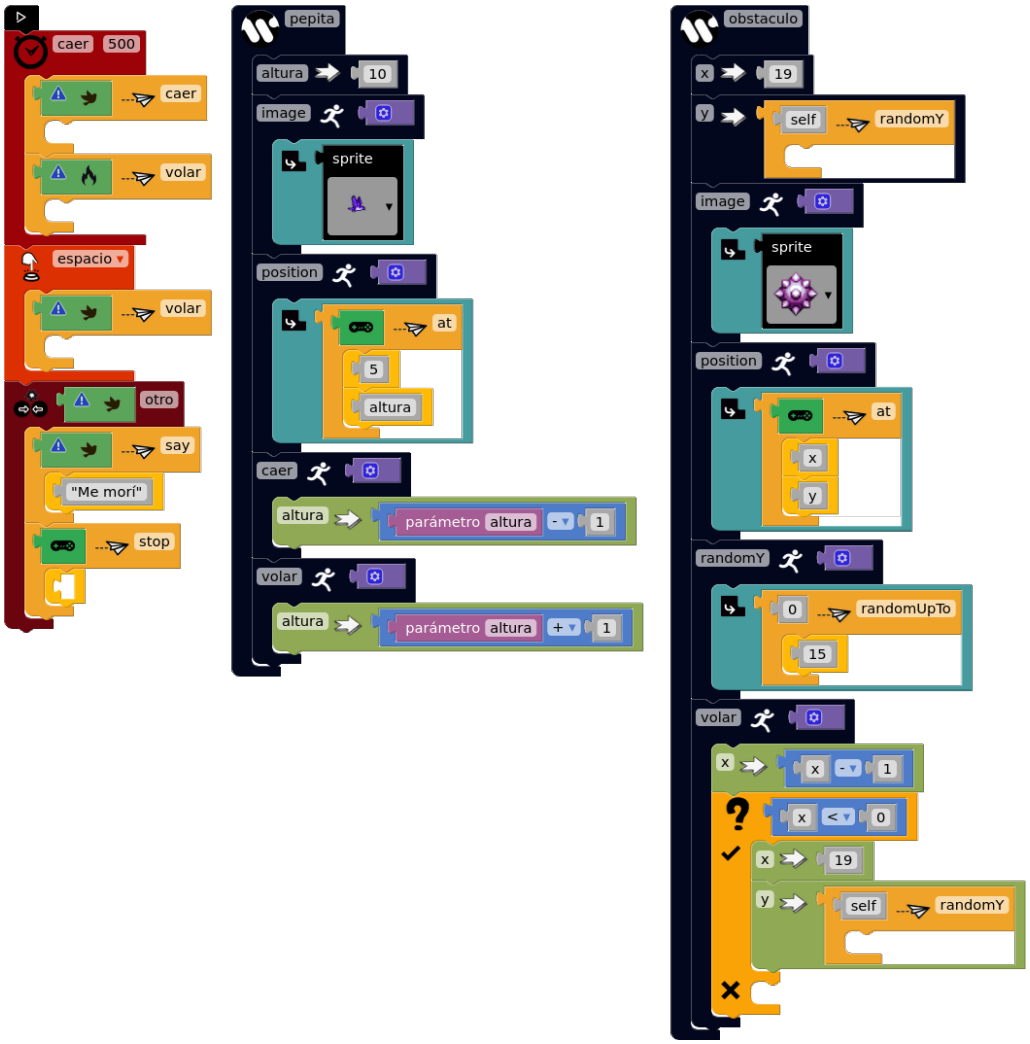


Fig. D.1: Primera solución brindada

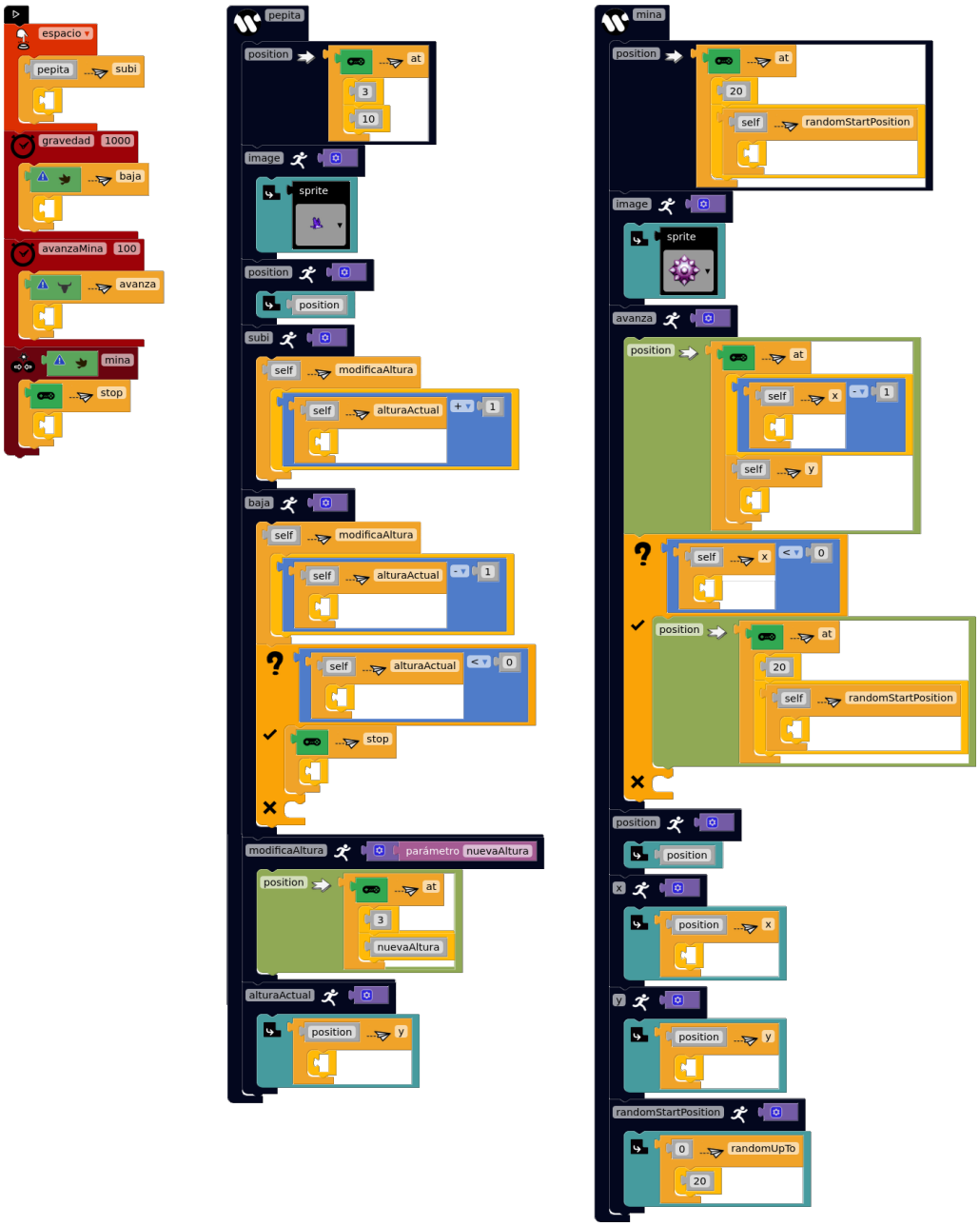


Fig. D.2: Segunda solución brindada

Fig. D.3: Tercera solución brindada

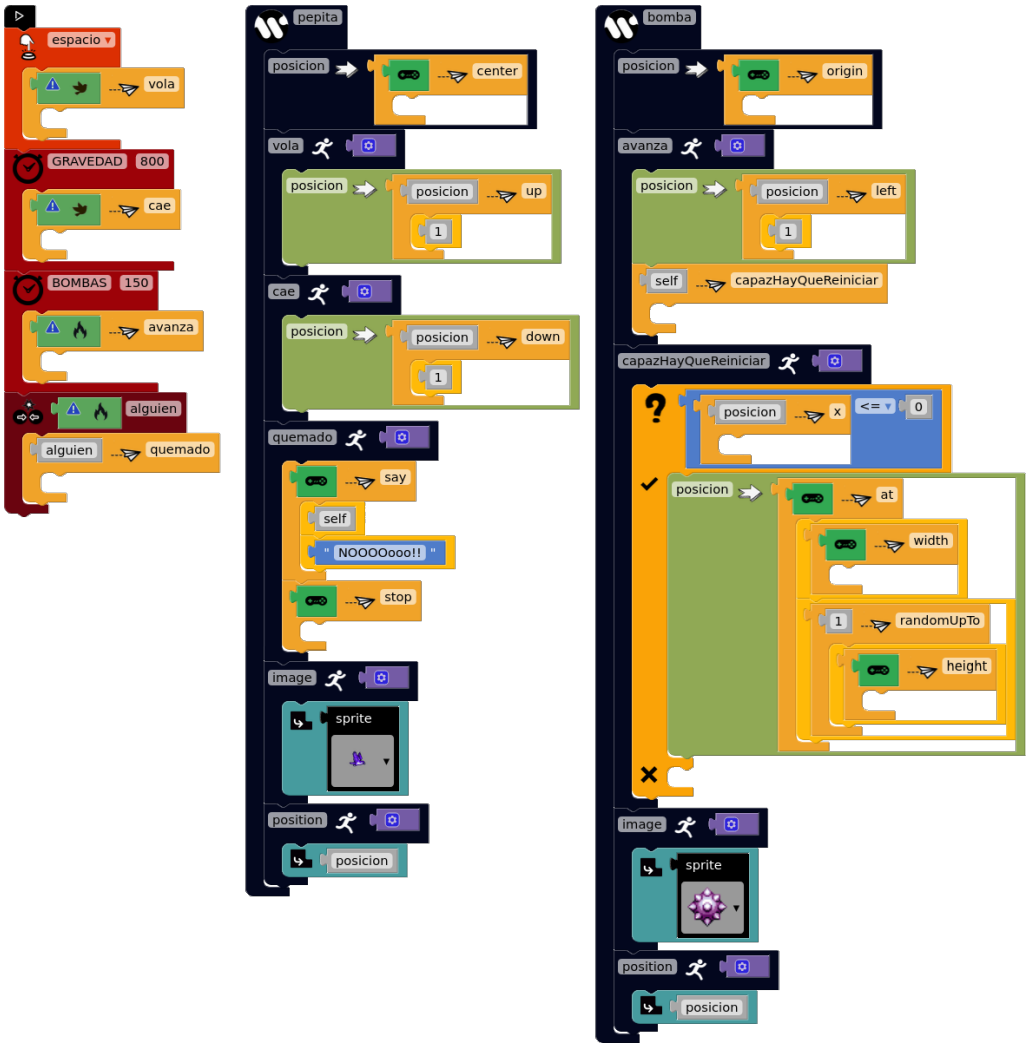


Fig. D.4: Cuarta solución brindada

E. POSTER

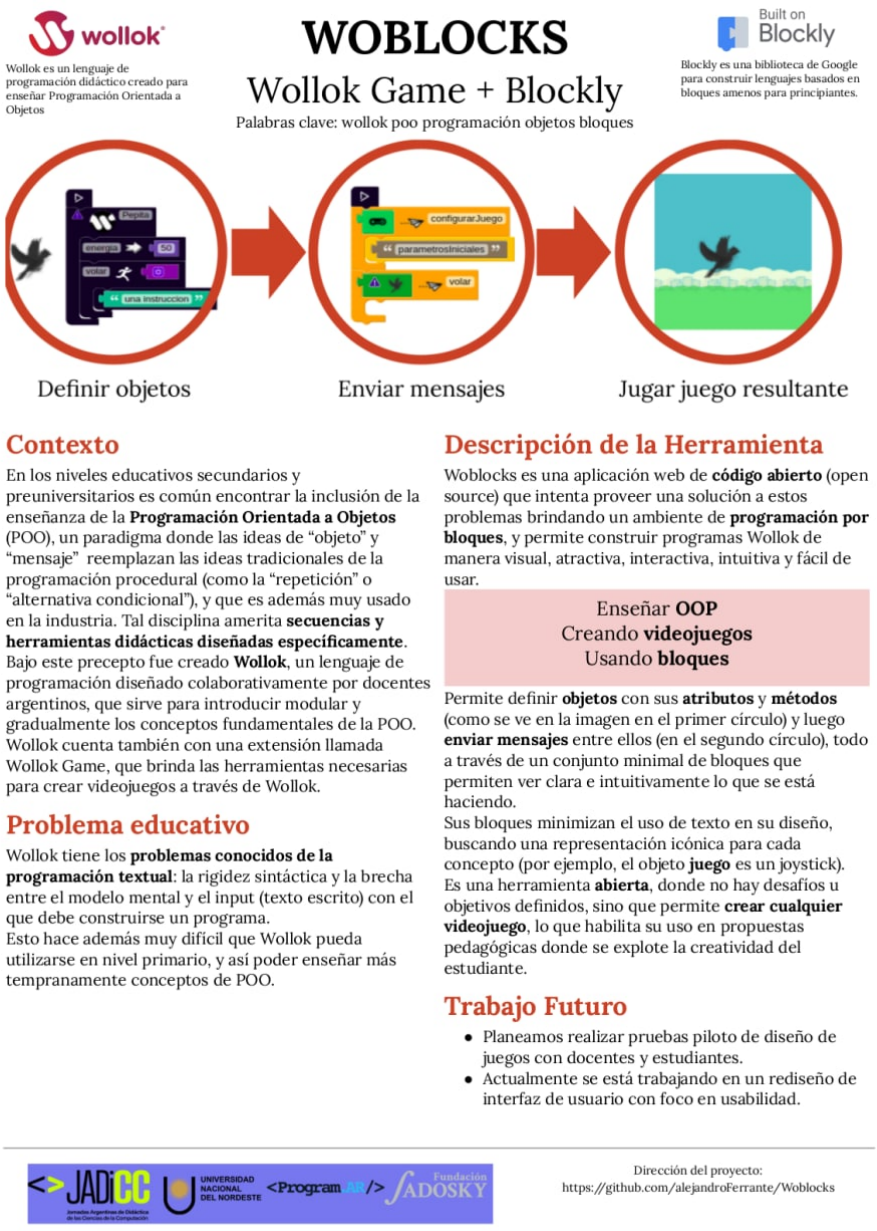


Fig. E.1: Poster presentado en JADICC 2022

Bibliografía

[1] Max Roser, Hannah Ritchie, and Esteban Ortiz-Ospina. Internet. *Our World in Data*, 2015. https://ourworldindata.org/internet.

[2] Ivan Mikovic, Teodora Vučković, Darko Stefanović, and Srdjan Sladojevic. Utilizing web and cloud-based technologies to support corporate business operations. In *XIII International May conference on strategic management IMKSM 2017 At: Bor, Serbia*, 05 2017.

[3] Ash Turner. How many smartphones are in the world?). `https://www.bankmycell.com/blog/how-many-phones-are-in-the-world`.

[4] comscore. Individuals using the internet in argentina). `https://www.comscore.com/Insights/Blog/US-Smartphone-Penetration-Surpassed-80-Percent-in-2016`, 2017.

[5] Felix Richter. Smartphones are taking over). `https://www.statista.com/chart/210/smartphones-are-taking-over/`, 2012.

[6] The World Bank. Individuals using the internet in argentina). `https://data.worldbank.org/indicator/IT.NET.USER.ZS?locations=AR`, 2019.

[7] Knud Lasse Lueth. State of the iot 2020: 12 billion iot connections, surpassing non-iot for the first time, 2020.

[8] Marc Prensky. Digital natives, digital immigrants. *From On the Horizon (MCB University Press, Vol. 9 No. 5)*, 2001.

[9] Sukirman, Dias Aziz Pramudita, Aziz Afiyanto, and Utaminingsih. Block-based visual programming as a tool for learning the concepts of programming for novices. *IJIET 2022 Vol.12(5): 365-371 ISSN: 2010-3689*, 2022.

[10] Pooriya Balavi Hampus Gunnrup. Children learning object oriented programming: A design science study. *Department of Computer Science and Engineering, University of Gothenburg, Chalmers University of Technology, Gothenburg, Sweden*, 2017.

[11] Lucas Spigariol. Estrategias pedagógicas para la enseñanza de la programación. *Universidad Tecnológica Nacional, Facultad Regional Buenos Aires, Departamento de Ingeniería en Sistemas de Información*, 2016.

[12] Shorena Abesadze and David Nozadze. Make 21st century education: The importance of teaching programming in schools. *nternational Journal of Learning and Teaching, University of Georgia, Tbilisi, Georgia*, 2020.

[13] Leon Sterling. Coding in the curriculum: Fad or foundational? *Swinburne University of Technology, Victoria*, 2016.

[14] Sei Kwon and Katri Schroderus. Coding in schools: Comparing integration of programming into basic education curricula of finland and south korea. *Finnish Society on Media Education*, 2017.

[15] Fernando Schapachnik and María Belén Bonello. *Ciencias de la Computación en la escuela: Guía para enseñar mucho más que a programar*. Siglo XXI Editores, 2022.

[16] Soly Mathew Biju. Difficulties in understanding object oriented programming concepts. *University of Wollongong in Dubai*, 2013.

[17] Javier Blanco, Leticia Losano, Nazareno Aguirre, María Marta Novaira, Sonia Permigiani, and Gastón Scilingo. An introductory course on programming based on formal specification and program calculation. *Facultad de Matemática, Astronomía, Física y Computación, Universidad Nacional de Cordoba, Facultad de Ciencias Exactas, Físico-Químicas y Naturales Universidad Nacional de Río Cuarto* , 2009.

[18] Ronny Scherer, Fazilat Siddiq, and Bárbara Sánchez-Scherer. Some evidence on the cognitive benefits of learning to code. *Frontiers in Psychology*, 2021.

[19] Scherer Ronny, Siddiq Fazilatm, and Viveros Bárbara. The cognitive benefits of learning computer programming: A meta-analysis of transfer effects. *Journal of Educational Psychology*, 2018.

[20] Milena Vujosevic and Janicic Dušan Tošić. The role of programming paradigms in the first programming courses. *The Teaching of Mathematics*, 2008.

[21] Susan S. Brilliant and Timothy R. Wiseman. The first programming paradigm and language dilema. *Department of Mathematical Sciences, VIrginia Commonwealth University*, 1996.

[22] Abdullah Khanfor and Ye Yang. An overview of practical impacts of functional programming. In *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*, pages 50–54, 2017.

[23] Frank Pfenning, Thomas J. Cortina, and William Lovas. Teaching imperative programming with contracts at the freshmen level. *Department of Computer Science Carnegie Mellon University*, 2011.

[24] Noa Ragonis and Mordechai Ben-Ari. A long-term investigation of the comprehension of oop concepts by novices. *Computer Science Education Vol. 15, No. 3*, 2005.

[25] Mehmet C. Okur. Teaching object oriented programming at the introductory level. *Journal of Yasar University*, 2006.

[26] Laxmi P. Gewali and John T. Minor. Multi-paradigm approach for teaching programming. *School of Computer Science, University of Nevada, Las Vegas*, 2006.

[27] James Miller, Gerald Darroch, Murray Wood, Andrew Brooks, and Marc Roper. *Changing Programming Paradigm — An Empirical Investigation*, pages 62–65. Springer US, Boston, MA, 1995.

[28] Daniel Gayo Avello, Agustín Cernuda del Río, Juan Manuel Cueva Lovelle, Marián Díaz Fondón, María Pilar Almudena García Fuente, and José Manuel Redondo López. Reflexiones y experiencias sobre la enseñanza de poo como único paradigma. *Departamento de Informática,Universidad de Oviedo*, 2003.

[29] Charles Reis and Robert Cartwright. Taming a professional ide for the classroom. *ACM SIGCSE Bulletin Volume 36, Issue 1*, 2004.

[30] Zhixiong Chen and Delia Marx. Experiences with eclipse ide in programming courses. *Journal of Computing Sciences in Colleges Volume 21, Issue 2*, 2005.

[31] Madasari Bte Ansari. Incorportating visual and animation teaching tools in computer programming classes for effective teaching and learning. *Faculty of Information Technology and Multimedia Communications, Open University Malaysia*, 2011.

[32] Al-Linjawi Arwa and Al-Nuaim Hana. Using alice to teach novice programmers oop concepts. *Journal of King Abdulaziz University-Science*, 2010.

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

[34] Jens Bennedsen,Michael E. Caspersen. Teaching object-oriented programming – towards teaching a systematic programming process. *Eighth Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts. Affiliated with 18th European Conference on Object-Oriented Programming (ECOOP 2004) - Oslo, Norway*, 2004.

[35] Myungsook Klassen. Visual approach for teaching programming concepts. *Computer Science Department, California Lutheran University*, 2006.

[36] Andrew Rudder, Margaret Bernard, Shareeda Mohammed. Teaching programming using visualization. *The University of The West Indies St. Augustine Campus, Trinidad and Tobago*, 2007.

[37] Marcos J. Gomez, Marco Moresi, and Luciana Benotti. Text-based programming in elementary school: A comparative study of programming abilities in children with and without block-based experience. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, page 402–408, New York, NY, USA, 2019. Association for Computing Machinery.

[38] Carla Griggio, German Leiva, Guillermo Polito, Gisela Decuzzi, and Nicolas Passerini. A programming environment supporting a prototype-based introduction to oop. *Universidad Tecnologica Nacional Argentina, Universidad Nacional de Quilmes Argentina*, 2011.

[39] Osman Gazi Yildirim and Nesrin Ozdener. An action research study on the development of object-oriented programming course. *International Journal of Technology in Education and Science (IJTES), 5(4), 620-628.*, 2021.

[40] Luciana Benotti, Marcos J. Gómez, and Cecilia Martínez. Unc++duino: A kit for learning to program robots in python and c++ starting from blocks. In Munir Merdan, Wilfried Lepuschitz, Gottfried Koppensteiner, and Richard Balogh, editors, *Robotics in Education*, pages 181–192, Cham, 2017. Springer International Publishing.

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

[42] Nicolás Passerini, Carlos Lombardi, Javier Fernandes, Pablo Tesone, and Fernando Dodino. Wollok: Language + ide for a gentle and industry-aware introduction to oop. In *2017 Twelfth Latin American Conference on Learning Technologies (LACLO)*, pages 1–4, 2017.

[43] Carlos Lombardi and Nicolás Passerini. Wollok: reconciliando didáctica e industria en un lenguaje educativo para poo. In *XIV Congreso Nacional de Tecnología en Educación y Educación en Tecnología (TE&ET 2019),(Universidad Nacional de San Luis, 1 y 2 de julio de 2019).*, 2019.

[44] Nicolás Passerini and Carlos Lombardi. Postponing the concept of class when introducing oop. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, page 152–158, New York, NY, USA, 2020. Association for Computing Machinery.

[45] Bluej. `https://www.bluej.org/`.

[46] Leon Starr. *Executable UML How to Build Class Models.* Prentice Hall, 2001.

[47] Karel the robot. `https://mpru.github.io/karel/`.

[48] Robocode. `https://robocode.sourceforge.io/`.

[49] Alice. `http://www.alice.org/`.

[50] Code combat. `https://codecombat.com/`.

[51] Greenfoot. `https://www.greenfoot.org/door`.

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

[53] Wanda P. Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Steve Cooper. Mediated transfer: Alice 3 to java. In *Technical Symposium on Computer Science Education*, 2012.

[54] Neil Fraser. Blockly history. `https://neil.fraser.name/software/Blockly-History/`, 2019.

[55] Proyecto de wollok en github. `https://scratch.mit.edu/`.

[56] Fundación Uqbar. Fundación uqbar. `https://uqbar.org`, 2008.

[57] Nahuel Palumbo Lucas Spigariol, Nicolas Passerini. Enseñando a programar en la orientación a objetos. *Universidad Tecnológica Nacional, F.R. Buenos Aires. Universidad Nacional de Quilmes.*, 2013.

[58] Carla Griggio, Germán Leiva, Guillermo Polito, Gisela Decuzzi, and Nicolás Passerini. A programming environment supporting a prototype-based introduction to oop. In *Proceedings of the International Workshop on Smalltalk Technologies*, IWST '11, New York, NY, USA, 2011. Association for Computing Machinery.

[59] Lucas Spigariol, Nahuel Palumbo, and Nicolás Passerini. Competencias en programación orientada a objetos. *Electronic Journal of SADIO*, 20, 2021.

[60] Wollok lsp ide. `https://github.com/uqbar-project/wollok-lsp-ide`.

[61] Carolyn Snyder. *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. Morgan Kaufmann Publishers, 2003.

[62] Jadicc 2022. `https://jadicc2022.unne.edu.ar/`.

[63] Nasa task load index (tlx). `https://ntrs.nasa.gov/citations/20000021488`.

[64] Johan Huizinga. *Homo Ludens*. Beacon press, 1972.

[65] Raph Koster. *Theory of Fun for Game Design*. O'Reilly Media, Inc., 2nd edition, 2013.

[66] Wong Yoke Seng, Maizatul Hayati Mohamad Yatim, and Tan Wee Hoe. Learning object-oriented programming paradigm via game-based learning game – pilot study. *Sultan Idris Education University, Malaysia*, 2018.