GENERACIÓN AUTOMÁTICA DE ESCENARIOS EN UN JUEGO EDUCATIVO CON DESAFÍO ADAPTADO A LAS NECESIDADES DE APRENDIZAJE

Trabajo de Fin de Grado

Presentado por

Adrián Alcántara Delgado

Dirigido por

Prof. Dr. D. Carlos León Aznar

Facultad de Informática, Universidad Complutense de Madrid

Grado en Desarrollo de Videojuegos

Madrid, 2021/2022

AUTOMATED SCENARIO GENERATION IN AN EDUCATIVE GAME WITH CHALLENGED ADAPTED TO LEARNING NEEDS

Bachelor Thesis

By

Adrián Alcántara Delgado

Supervised by

Prof. Dr. D. Carlos León Aznar

Computer Science Faculty, Universidad Complutense de Madrid

Bachelor Degree in Video Game Development

Madrid, 2021/2022

Resumen

Enseñar robótica a personas que cursan primaria o los primeros cursos de secundaria es una tarea que Smartick se propuso hace tiempo. El contenido de robótica del producto de Smartick se basa en un plan de estudios de mapas en los que hay que llevar un robot a una bandera usando una secuencia de instrucciones. Cada mapa tiene una solución diferente, que se ha creado manualmente basándose en unos criterios didácticos. Pensar las soluciones y crear el mapa a partir de las mismas es un proceso costoso y lento.

Para resolver el problema de tiempo y cantidad de contenido, se ha desarrollado un proyecto en el que, determinando las instrucciones disponibles, el tamaño deseado de la solución y las instrucciones obligatorias, se genera un número de mapas mucho mayor al de todo el plan de estudios de robótica del producto de Smartick para una unidad didáctica. El generador desarrollado usa las reglas didácticas necesarias para que solo se pueda trabajar con soluciones filtradas y que cumplan los requisitos de la unidad didáctica, del producto de robótica y también los parámetros que haya definido el equipo didáctico para la generación de cada unidad.

Las conclusiones del proyecto son las esperadas gracias a los resultados de las pruebas con los alumnos. Mantiene y mejora los resultados de los mapas creados desde cero y con soluciones inventadas. Facilita la creación de contenido, el equipo didáctico se ahorra la parte más abstracta de la generación de unidades didácticas, crear soluciones mezclando instrucciones. Los mapas que usan estas soluciones han dado buenos resultados, tanto por sí solos como mejorando los demás ejercicios posteriores. Todos han tenido buena superación y los alumnos los han resuelto en un número bajo de intentos. Con esta herramienta funcionando, se puede seguir mejorando para que sea más compleja y llegue a un estado de autosuficiencia en el que se generen mapas completos.

Palabras clave: Generador, didáctica, programación, robótica, Smartick

Abstract

Teaching robotics to people who are in primary or lower secondary school is a task that Smartick set out for a long time. The robotics content of Smartick's product is based on a map curriculum in which a robot must be brought to a flag using a sequence of instructions. Each map has a different solution, which has been created manually based on educational criteria. Thinking of solutions and creating the map from them is a costly and time-consuming process.

To solve the problem of time and amount of content, a project has been developed in which, by determining the available instructions, the desired size of the solution and the mandatory instructions, a much larger number of maps is generated than the entire plan. robotics studies of the Smartick product for a didactic unit. The developed generator uses the necessary didactic rules so that it is only possible to work with filtered solutions that meet the requirements of the didactic unit, the robotics product and also the parameters defined by the didactic team for the generation of each unit.

The conclusions of the project are as expected thanks to the results of the tests with the students. Maintains and improves the results of maps created from scratch and with invented solutions. It facilitates the creation of content, the didactic team saves the most abstract part of the generation of didactic units, creating solutions by mixing instructions. Maps using these solutions have given good results, both on their own and by enhancing the other exercises below. They have all passed well and the students have solved them in a low number of attempts. With this tool running, it can be further improved to become more complex and reach a self-sufficient state where complete maps are generated.

Keywords: Generator, didactics, programming, robotics, Smartick

Índice

1. Introducción

La educación se ha ido digitalizando en la última década en sus distintos niveles y con sus diferentes peculiaridades. En las universidades, bachillerato se han introducido y se han asentado campus virtuales. En algunos institutos y colegios se utilizan aplicaciones para reforzar el aprendizaje de diferentes materias. Esto se conoce como tecnología educativa.

La digitalización se ha ido acelerando tanto en las aulas como en actividades extraescolares debido al volumen de alumnos y las necesidades tan diferentes que pueden llegar a tener en el aula, además de estos últimos años por la pandemia. Esta necesidad la cubren aplicaciones online como Smartick, la empresa a la que va dirigido este proyecto.

Para cubrir las necesidades del alumno en el aula usando estas aplicaciones, tiene que haber una cantidad de contenido considerable para cubrir la velocidad de aprendizaje y una gran variedad en ese contenido para evitar un estancamiento en algunos conceptos que le puedan resultar más difíciles.

Para cubrir este contenido hace falta un equipo didáctico compuesto por humanos que cree un plan de estudios de la materia, una estrategia de aprendizaje para cubrir la mayor parte del espectro de los diferentes tipos de alumnos y una tecnología de creación de contenido.

Hay aplicaciones de aprendizaje de diversas materias que ofrecen una réplica del contenido de libros de texto que se usan en las aulas, otras tienen su contenido propio. Sin embargo, no todas las aplicaciones pueden ofrecer gran cantidad, variedad y adaptación de contenido enfocado al alumno para potenciar su aprendizaje y no solamente cubrir el currículum.

Smartick es un método online de aprendizaje con varios programas a elegir. Uno de ellos es el de matemáticas, combina programación, cálculo y lógica. La combinación se hace en sesiones repartidas diariamente. En orden de frecuencia diaria, cálculo es el tipo de sesión con mayor volumen, después va lógica y por último programación, o como lo nombran la mayoría de personas que conocen el producto, *Coding*.

Coding es el producto de programación en el que se introduce al alumno al proceso de resolución de problemas con código. Para enseñar esta introducción se utiliza un plan de estudios con conceptos repartidos en diferentes actividades. Parte de estos conceptos se enseñan en el plan de estudios de robótica. De forma muy breve, se tiene que llevar un robot (llamado *Robby*) a una bandera en el mapa que esté trabajando el alumno en ese momento.

No obstante, la enseñanza online tiene desventajas. Una de las desventajas de la tecnología educativa es que su implantación es desigual a causa de la brecha digital. Se crean iniciativas, estrategias de responsabilidad social corporativa, becas, pero el volumen de alumnos es tan inmenso que es difícil de paliar este problema. Otras desventajas no tan graves como la anterior es la deshumanización de la educación. Smartick tiene un equipo pedagógico para cubrir en cierto modo ese problema. Por norma general, en los alumnos que utilizan aprendizaje online con cualquier aplicación o servicio, tienen cierto grado de dependencia digital. Puede perjudicar en habilidades que se trabajan en el aula como la escritura, manualidades. Si las sesiones de aprendizaje se alargan puede haber distracciones, ya no solo con el ambiente si no también con otras aplicaciones instaladas. También puede crear problemas de interacción social porque normalmente el aprendizaje online suele ser extraescolar y en casa.

1.1. Motivación

Típicamente, el contenido de esta parte del producto se ha hecho de manera manual, bajo una supervisión didáctica para darle calidad a cada mapa que hacen los alumnos en su curva de aprendizaje.

Diseñar una herramienta que ayude al trabajo humano ahorra mucho trabajo al equipo didáctico y aporta gran variedad a cada unidad didáctica del plan de estudios. En el entorno de producción de *Coding*, hay alrededor de quinientos ejercicios de este tipo en el plan de estudios, el generador es capaz de generar más ejercicios por cada unidad didáctica.

Esta herramienta pretende quitar la mayor carga de trabajo posible del equipo didáctico en esta parte del plan de estudios para que así puedan dedicar su tiempo a la parte más compleja del producto. Si se requiere contenido nuevo de robótica en la propia sesión, si hay resultados adversos de alumnos en algunas unidades, si hay alumnos que después de su sesión rutinaria desean hacer más mapas para practicar, se pueden hacer con esta herramienta.

1.2. Hipótesis de partida

Este proyecto se inicia con la hipótesis de que si se utilizan los mapas generados con los mismos criterios que los creados manualmente, habrá mucha más variedad en el plan de estudios de robótica, no habrá dificultades innecesarias que no hayamos podido evitar con la generación manual y los alumnos podrán practicar más los conceptos que trabajen en ese momento.

En caso de que la hipótesis sea cierta, seguiría desarrollándose el proyecto con los objetivos que hay definidos a futuro. El siguiente hito sería generar mapas completos que cumplan exactamente los mismos requisitos que cumplen los creados manualmente.

1.3. Objetivos

El objetivo de este proyecto es crear una herramienta de generación de ejercicios didácticos del producto de robótica de Smartick para que pueda usarse en distintos ámbitos de la empresa. Estos ámbitos pueden ser para renovar o aumentar el contenido de sesión del alumno, también para el contenido post sesión, crear demos para la venta del producto, crear planes de estudios reducidos para campamentos. Aumentar el contenido diario de la academia presencial de Smartick.

Para conseguir este objetivo de negocio, los objetivos del proyecto han sido diseñar esta herramienta para que genere soluciones automáticamente tomando como referencia unas directrices didácticas que los puede determinar el equipo didáctico. Aparte, en la generación de las soluciones, se tienen que cumplir las reglas didácticas que ya cumplen los ejercicios que hay en el entorno de producción de *Coding*.

Como objetivo tecnológico, la generación debe ser eficiente y la experiencia de usuario debe ser intuitiva. Se diseñará un algoritmo lo más eficiente posible en tiempo de ejecución y en uso de memoria. También se diseñará una interfaz sencilla para poder usar el generador de soluciones.
Los objetivos de este trabajo son:

- Diseño de una herramienta que genere soluciones para mapas de robótica del producto de Smartick
- Automatización de la generación de las soluciones
- Parámetros didácticos sencillos de introducir para mejorar la usabilidad de la herramienta
- Todas las soluciones deben cumplir los criterios didácticos

- Generación eficiente de las soluciones
- Implementar modelo vista controlador para una comunicación de la futura base de datos con la interfaz de la herramienta
- Crear una interfaz gráfica para una usabilidad más sencilla
- Pruebas sobre alumnos reales que verifican la hipótesis de partida
- Planear los siguientes hitos a futuro a partir del resultado de las pruebas

1.4. Metodología

En esta sección se detallan los aspectos más importantes de la metodología usada para el desarrollo de este proyecto. Tanto las plataformas usadas para gestionar el proyecto, como las librerías necesarias.

1.4.1. Tecnologías y herramientas usadas

Se han usado apartados de la metodología de desarrollo ágil Scrum. El proyecto ha sido llevado a cabo por una sola persona, por lo que las directrices de esta metodología en las reuniones sólo se han aplicado en las reuniones con el equipo tutor. Pero sí que se ha usado el concepto de sprint, usando como periodo, dos semanas de desarrollo, salvo en los sprints de pruebas, que han sido de una semana. También se han hecho retrospectivas de los resultados de las pruebas locales para tomar decisiones en los siguientes pasos del desarrollo del proyecto.

Para crear y gestionar las tareas se ha utilizado la aplicación Notion[1], es un software para la gestión de proyectos muy parecido a Jira[2], de Atlassian. En esta aplicación se han creado las tareas principales del proyecto para poder tener una vista global en cualquier momento. Ha sido muy sencillo separar la parte principal del proyecto (todas las tareas relacionadas con el generador), de los aditivos como el generador de mapas y que el servicio sea visual y fácil de usar. De esta manera se ha podido priorizar las tareas por importancia y nivel.

Se ha utilizado GitHub[3] como sistema de control de versiones. Ha permitido mantener el proyecto constantemente actualizado para poder trabajar en cualquier dispositivo que se ha usado para el desarrollo y tener un repositorio accesible sin miedo a la pérdida del progreso del proyecto.

El proyecto está desarrollado en IntellIJ IDEA[4] que es un entorno de desarrollo muy potente y agradable para el programador. Aporta muchas facilidades para

[1] https://www.notion.so
[2] https://www.atlassian.com/jira-software/oficial
[3] https://github.com/
[4] https://www.jetbrains.com/idea/

poder usar Spring[5] como framework de Java y también da pequeños consejos para mejorar detalles del código y hacerlo más eficiente.

Para la interfaz del proyecto, algunas librerías han sido necesarias pero otras han sido por comodidad para desarrollar. Se ha utilizado Bootstrap para añadir estilos a la página en la que se seleccionan las opciones del generador y se muestra el listado de soluciones. Se ha utilizado para la comunicación con el servidor local y para la edición y lógica de la web, AJAX y jQuery. Para facilitar el trabajo de desarrollo de javascript se ha programado en ECMAscript 6, pero como esta versión no funciona en todos los navegadores, se transpilan los archivos javascript en ejecución, gracias a Gulp.js y Babel, que se han instalado en el proyecto usando Node.js

1.4.2. Plan de trabajo

El plan de trabajo que se ha seguido es el mismo que en el entorno laboral de Smartick, para mantener la familiaridad. Primero se ha utilizado un software de gestión de proyectos. En este caso se ha elegido Notion por la similitud que tiene con Jira y su licencia gratuita. Es un software muy intuitivo aunque con menos posibilidades. Por ejemplo, en Jira podría haberse creado épicas o historias de usuario, para crear y gestionar las tareas de alto nivel y para diferenciarlas entre tareas técnicas o tareas de cara al cliente del proyecto. Aún con estas limitaciones se ha podido organizar las tareas del proyecto y mantener una vista general del progreso, revisando las tareas que no estaban empezadas, las que estaban en curso y las finalizadas.

Una vez planificada la hoja de ruta del proyecto, se ha creado un repositorio en GitHub para que haya una integración continua del proyecto y poder salvar cualquier avance o prueba a lo largo del desarrollo, bien porque se haya terminado de desarrollar una funcionalidad o porque se ha tenido que continuar en otro dispositivo en mitad del desarrollo. Gracias a IntelliJ ha sido muy sencillo seguir esta metodología ágil porque tiene una integración de GitHub muy cómoda y segura.

Aquí se muestran las tareas a alto nivel ordenadas cronológicamente:

1. Estudio de la generación de soluciones con instrucciones disponibles
2. Añadir a la generación instrucciones obligatorias
3. Aplicación de combinatoria para generación de soluciones
4. Definir y generar las reglas didácticas
5. Seleccionar soluciones candidatas con las reglas didácticas
6. Comprobar el funcionamiento y eficiencia de la primera versión del generador

[5] https://spring.io/

7. Corregir fallas de la primera versión
8. Crear un multiconjunto para cubrir la diferencia de tamaño del set y el tamaño de la solución
9. Aplicación de programación dinámica a la generación de soluciones
10. Comprobar el funcionamiento de la nueva versión
11. Preparación de controladores y servicios para la página web del generador
12. Creación de la vista y estilos para la página web del generador
13. Maquetación de la vista
14. Pruebas visuales
15. Diseño del generador de mapas a partir de la solución
16. Reglas didácticas para la generación de mapas

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
O1										
O2										
O3										
O4										
O5										
O6										
O7										
O8										
O9										
O10										
O11										
O12										
O13										
O14										
O15										
O16										

O: Objetivo S: Sprint
Azul: Completado Naranja: A futuro

Tabla 1. Las tareas se han distribuido en sprints (S1…S10) que han ocupado dos semanas cada una. En el caso de las pruebas se han realizado en la mitad de un sprint.

1.5. Estructura del documento

En el capítulo dos se explican las referencias del proyecto. Las principales referencias son los generadores de ejercicios del producto de matemáticas de Smartick. Se explica y se pone contexto sobre el producto de programación de Smartick, Coding. Se mencionan y explican los proyectos que se habían tomado en un principio como referencia para generar mapas: AlphaGo y un generador de mapas del videojuego Sokoban. También se mencionan las referencias que sí se han tenido en cuenta para desarrollar el proyecto: combinatoria para la generación de las soluciones y el uso de programación dinámica para mejorar el rendimiento.

En el capítulo tres se expone todo el diseño del generador de soluciones. Se explican las instrucciones que tienen que usar los alumnos para resolver los ejercicios. Se argumentan las reglas didácticas que deben aplicar todas las soluciones en cada generación para que se puedan usar en un mapa.

En el capítulo cuatro se explica la parte de desarrollo de la tecnología. Se presenta la arquitectura que se usa y el motivo. Se explican también las implementaciones de las dos fases de implementación. Además, al final del capítulo se enseñan las soluciones utilizadas en las pruebas con alumnos y cómo ven los alumnos los mapas creados a partir de las distintas generaciones de las tres unidades didácticas.

En el capítulo cinco se detallan todos los parámetros que usan las tablas y los gráficos. También se argumenta cómo se han evaluado los datos. Se muestran las tablas y los gráficos necesarios para entender los resultados del generador. Todos estos datos han sido extraídos de Data Studio y se han extraído de sesiones de alumnos. Por último se analizan los resultados en busca de argumentos que apoyan el uso del generador y también de fallos para perfeccionar siguientes versiones de este.

En el capítulo seis se discuten los resultados del proyecto. Se ha dado importancia y se han destacado los peores resultados, que han sido pocos los ejercicios afectados, para proponer iniciativas a futuro y perfeccionar la herramienta.

En el capítulo siete se hace un planteamiento parecido al capítulo dos pero enfocado a futuro, con una nueva hipótesis de partida más ambiciosa, con una motivación hacia una herramienta mucho más elaborada y compleja, justificando esta escala con los buenos resultados de las pruebas. Además, se enumeran los grandes objetivos de las siguientes versiones de la herramienta hasta que se pueda considerar completa si se cumple el último objetivo.

En el capítulo diez se listan todas las referencias del proyecto. Las referencias para los generadores que se revisaron en el estado del arte, que servirán para las siguientes etapas del proyecto. Además también están las referencias para los planteamientos de combinatoria que se usan en el generador de soluciones.

2. Estado del arte

La idea de crear una herramienta que genere automáticamente soluciones y mapas de robótica surge de reducir o incluso eliminar la carga de trabajo que hay en el equipo didáctico de *Coding*, en esa parte del plan de estudios. Desde la creación del producto de programación de Smartick, el contenido didáctico de robótica se ha generado manualmente. Esto ha significado una gran limitación de tiempo y de cantidad de contenido. Actualmente hay alrededor de 500 ejercicios en los que se trabajan distintos conceptos de programación y de robótica. Con el proyecto se pretende poder generar más de esa cantidad de contenido por unidad didáctica.

2.1. Coding

Coding es un producto que enseña a alumnos de primaria y principios de secundaria, que tengan cierto nivel de lectura y hayan superado las unidades matemáticas de percepción espacial. El plan de estudios de *Coding* es una introducción a la programación y a la robótica. Los primeros conceptos se enseñan en un entorno de robótica en el que mediante una secuencia de instrucciones se mueve a un robot hasta una bandera. Este entorno es en el que trabaja el generador.

Los conceptos que se enseñan en el apartado de robótica del producto son variados entre pensamiento computacional, robótica y programación. Se trabaja la importancia del orden de ejecución en una secuencia de instrucciones. La orientación espacial es esencial en el plan de estudios de robótica de *Coding* porque los mapas se resuelven en vista isométrica. La lectura de código, para saber dónde estará la meta teniendo la posición inicial y las instrucciones ordenadas. Se trabaja la depuración pudiendo ejecutar la solución propuesta por el alumno paso a paso y de esta manera poder encontrar posibles errores en el código del alumno. También hay ejercicios con corrección de código erróneo, ya sea cambiando una instrucción incorrecta o borrando una sobrante. Se enseña el concepto de bucles y se enseña a encontrar patrones de instrucciones en los caminos a la bandera. Después de asentar el concepto de bucle se enseña a trabajar con bucles anidados. También se enseña a trabajar la resiliencia y tolerancia a la frustración si se quedan atascados en un ejercicio ya que de no superarlo no pueden avanzar al siguiente porque la dificultad del contenido del plan de estudios es creciente.

Tienen la opción de seguir intentándolo, repasar, ver el tutorial recomendado o después de fallar un número de veces, poder ver la solución del problema.

Con estos conceptos, el equipo didáctico de *Coding* definió una serie de unidades didácticas que formaban un plan de estudios de robótica. Cada unidad didáctica tiene una serie de instrucciones por defecto, que son las que pueden usar los ejercicios de la propia unidad. No es necesario usar todas las instrucciones por defecto. También puede ocurrir que en algunos ejercicios de una unidad se usen otras instrucciones además de las que están incluidas por defecto.

2.2. Smartick

Smartick es el método de aprendizaje al que pertenece *Coding*. Llevan años enseñando matemáticas online a alumnos con la misma franja de edad que utiliza *Coding*. En matemáticas se utilizan generadores de ejercicios en todo el plan de estudios. Los generadores funcionan mientras el alumno está haciendo la sesión, generando contenido adaptado a sus conocimientos y competencias.

Los generadores que utiliza Smartick para sus sesiones de matemáticas han sido la razón y la principal referencia para este proyecto. Salvando todas las diferencias que hay entre un producto y otro, tratando de los generadores.

La principal diferencia es que Smartick trabaja con enunciados y elementos de ayuda y explicación, la robótica en *Coding* se trabaja sobre una cuadrícula de 6x6 con obstáculos y objetos útiles e instrucciones. También hay otra gran diferencia que es la forma de evaluar. En matemáticas se valora la rapidez y se pueden corregir los ejercicios fallidos al final de la sesión. En *Coding* sin embargo no se avanza en el plan de estudios hasta que se consigue resolver el ejercicio que se está trabajando y se valora conseguir una cantidad considerable de ejercicios por unidad didáctica. También se valora como puntos extra la mejor solución posible.

2.3. Generadores relacionados

Al principio del proyecto el enfoque teórico que se le dió al proyecto era generar primero un mapa y después la solución. Pero fue un enfoque erróneo porque el contenido didáctico y el valor del ejercicio reside en la solución. El mapa es secundario, es casi decorativo y se puede modificar al antojo del diseñador, no tiene tanto impacto en el alumno a la hora de entender el contenido didáctico del ejercicio. El único mapa en la dificultad didáctica que pueda presentarse a un alumno, es la orientación espacial, por estar los mapas representados en perspectiva isométrica.

Con el enfoque de generar primero el mapa, se investigaron varios generadores para intentar conseguir un comienzo en el proyecto. Como se ha mencionado justo antes, el enfoque era erróneo. Las referencias que se van a explicar más abajo fueron las culpables del reenfoque, por así decirlo, del proyecto. Pero no es trabajo perdido, servirán de mucha utilidad para el futuro del proyecto. Con total seguridad, si se sigue trabajando en el proyecto en un futuro, el siguiente paso será crear los mapas tomando como referencia la información de los generadores que se explican en el siguiente apartado.

2.3.1. Árbol de búsqueda Monte Carlo

El árbol de búsqueda de Monte Carlo es una técnica de búsqueda muy usada en problemas de inteligencia artificial. Normalmente se usa en la planificación de movimientos en juegos como Tres en Raya, Conecta 4, 2048, póker o incluso Go. También se usa en videojuegos: en juegos de estrategia por turnos como en la IA de la saga Total War y para la generación de tableros como en Sokoban. (Ahmed, Umair et al.), (Dorit Dor y Uri Zwick)

Estudiando la capacidad computacional de resolver partidas de Go, teniendo un tablero de 19×19 y además su uso en generadores de mapas de Sokoban, crear una adaptación de este algoritmo es la solución para la generación de mapas. (Chaslot, Guillaume Maurice Jean-Bernard Chaslot)

2.3.1.1. Inteligencias artificiales para el juego GO

Investigar, utilizar el árbol de búsqueda de Monte Carlo pasa obligatoriamente por mencionar los programas de inteligencia artificial que juegan al juego de mesa Go. (Bernd Brugmann)
Go es un juego que en la inteligencia artificial se ha considerado un problema mayor que el ajedrez. Se ha intentado desarrollar durante décadas versiones para derrotar a los mejores jugadores. Este hito culminó en 2017 con AlphaGo.

El árbol de búsqueda Monte Carlo para juegos como Go, utiliza una “red de valor” y una “red de políticas”, ambas implementadas mediante una tecnología de redes neuronales de aprendizaje automático. Una cantidad limitada de detección de características pre-procesamiento-juego específico se utiliza para generar las entradas a las redes neuronales. Usa aprendizaje por refuerzo para mejorar su juego. (Cameron Browne)

2.3.1.2. Generadores de mapas de Sokoban

Los generadores de mapas del Sokoban que se han buscado para tomar referencia utilizan una combinación de técnicas de aprendizaje automático y árbol de búsqueda de Monte Carlo, combinadas con una amplia formación. (Ashlock, Daniel y Justin Schonfeld)

En algunos generadores de Sokoban se generan mapas proceduralmente, (Taylor, Joshua y Parberry, Ian). En otros, hay una serie de pasos para generar mapas. Principalmente se dividen en siete pasos en los que no todos se pueden reinterpretar para *Coding*:

1. Se construyen mapas vacíos escogiendo el ancho y el alto del tablero. En el caso de los mapas de *Coding* siempre es una cuadrícula de 6x6. El siguiente paso que toman es dividir el tablero vacío en mallas de 3x3. El generador tiene plantillas de 3x3 de paredes y suelos rodeados por bordes sin bloque, o pared, o suelo. En *Coding* en principio no se puede tener este sistema de plantillas por el set de instrucciones comparándolo con Sokoban.

2. Comprobaciones post-procesado: El nivel se comprueba por conectividad. Tiene que haber una sección de suelo contigua. En *Coding* sería equivalente a que Robby siempre tuviera un camino por el que Robby pudiera cruzar. En Sokoban el nivel debe tener suficiente suelo para el número definido de cajas, más el jugador y al menos una casilla vacía. En *Coding* el nivel debe tener suficientes casillas vacías para poder usar la solución generada, estas cuentan a Robby y la bandera. También habría que usar obstáculos alrededor del camino de la solución para no llegar a la bandera por un camino más corto.

3. Poner la meta en el mapa por fuerza bruta poniendo un tope de investigación por tiempo de ejecución. Este apartado es totalmente inútil para *Coding* ya que el punto de partida y de llegada ya se habrían definido en el paso anterior.

4. Encontrar el estado más lejano desde el estado objetivo por cada posición de meta. Esto es para encontrar el estado que con el camino corto más largo llegue a la meta. Se evalúa el camino que resulta acorde con el número de turnos, no se cuentan turnos en los que se mueve la caja en la misma dirección que la anterior y el número de veces que la solución cambia cuál caja es empujada. En *Coding* se podría usar como métricas para descartar mapas, las soluciones que pasan más de una vez por la misma casilla y no se puede utilizar un teletransportador para evitar esa situación.

5. Búsqueda exhaustiva usando teoría de complejidad computacional (PSPACE). Los problemas completados son consumidores completos de memoria, se cambia el espacio por el tiempo y el algoritmo usa profundización iterativa doble, cambiando memoria por tiempo.

6. Se genera el set de niveles que está tan lejos del estado de la meta como sea posible. Desecha niveles que sean demasiado fáciles, o que sean muy similares a niveles que ya están en el set de niveles. Cuando se ha conseguido un número de niveles, se ordenan por dificultad. Todo este paso que se aplica en Sokoban, teóricamente se aplicaría en *Coding* sin que se utilicen los niveles que tienen la meta lo más lejos posible, eso no interesa.

7. El generador de Sokoban tiene una evaluación de métricas de dificultad, aP - bS + cL - dB + R:
    a. P es el número de empujes.
    b. S es el número de otros niveles que el generador ha encontrado a ese nivel de profundidad.
    c. L es el número de líneas.
    d. B es el número de cajas.
    e. R es un número aleatorio pequeño.

También se toman como métricas de dificultad las cajas atrapadas, tocando las paredes, tocando el jugador, tocando otras cajas, o un objetivo tocando otro objetivo.

En Coding las métricas serían número de elementos que tiene que accionar Robby (teletransportadores, trampolines, recoger martillo, romper piedra con el martillo), número de avances, número de giros, cambios de altura, perspectiva de trabajo para el alumno, número de instrucciones totales.

2.3.2. Generador de soluciones: combinatoria en otros problemas

Para el planteamiento de generación de soluciones se han ido buscando diferentes ejemplos que tras adaptarlos, se pudieran utilizar para la generación de soluciones. En primera instancia se creía suficiente la combinación con repetición («Combinaciones con repetición») de una selección de elementos. Pero en el planteamiento del algoritmo («Combinations with repetitions») y a lo largo de la implementación ha habido adaptaciones hasta el generador final.

En primera instancia se ha utilizado combinaciones con repetición para las soluciones que tienen menos instrucciones que el set de instrucciones por defecto junto con el set de instrucciones obligatorias. Este caso casi no se usa ya que el conjunto de los sets anteriores es muy pequeño y las soluciones que hay ya en el plan de estudios de robótica son más largas que la suma de estos dos conjuntos. Como referencias se han buscado ejercicios de combinatoria de combinaciones y permutaciones (Luis Javier López Arredondo) de caracteres y de números. («Permutar un Array»)

Para la parte avanzada y mejorada del generador, me ha servido de mucha ayuda el libro de mi profesor Alberto Verdejo, para aplicar programación dinámica a la parte más costosa del generador y así poder hacerlo más eficiente. Además de otras referencias como (perspolis)

3. Diseño del generador de mapas

En este capítulo se va a explicar toda la etapa de diseño del proyecto. Poniendo en situación el diseño paramétrico de la tecnología que se ha creado. Se expondrán las instrucciones que usan los alumnos para resolver los problemas de robótica, además de las reglas didácticas que tienen que cumplir todas las soluciones. Se explicarán los dos diseños que se han planteado para el generador de soluciones.

3.1. Diseño del generador de soluciones

Recordemos que la parte del plan de estudios de *Coding* que trabaja la robótica consiste en mover a un robot (Robby), mediante una secuencia de instrucciones, a una meta. En el plan de estudios de *Coding*, cada unidad didáctica tiene una serie de instrucciones por defecto que se pueden utilizar para resolver el ejercicio. No tienen por qué utilizarse todas las que se ofrecen.

Ese conjunto de instrucciones disponibles es pequeño, de media hay definidos por cada unidad del plan de estudios de robótica 3.8 instrucciones, de un total de seis. Estas seis instrucciones son las siguientes:

- Avance: Permite mover a Robby una casilla hacia delante. Es la primera instrucción que se presenta en el plan de estudios.
- Retroceder: Robby se puede mover una casilla hacia atrás.
- Salto: Robby puede saltar un nivel hacia arriba o saltar hacia abajo y caer más de un nivel.
- Giros: Permite girar a Robby en ángulo recto sin moverse de casilla. Con esta instrucción empiezan a trabajar la orientación espacial.
- Acción: Permite a Robby realizar diferentes acciones. Recoger herramientas (martillo o lupa) para romper ciertos obstáculos. También permite accionar botones vinculados con puentes o con elevadores.

La mayoría de soluciones del plan de estudios, excepto los tutoriales de cada instrucción usan más de esas 3.8 instrucciones. Este dato, va a ser muy importante entre las fases del diseño e incluso de la implementación del generador. A continuación se van a enumerar las reglas didácticas y en los siguientes apartados se explican las dos fases que he tenido el diseño del algoritmo.

3.1.1. Reglas didácticas

Para que una solución se considere válida para poder usarse en un mapa que se creará posteriormente, debe cumplir todas las reglas didácticas. Estas reglas las ha diseñado el equipo didáctico del producto.

Una vez generadas las soluciones, da igual cómo se hayan generado, se validan con estas reglas para discriminar las soluciones no candidatas. A continuación se explican las reglas que se han usado en el generador:

- Siempre tiene que haber un avance o un retroceder o un salto. Son los tres únicos movimientos que trasladan a Robby de la casilla en la que se encuentren y los únicos que permiten llegar a la bandera.

- No usar dos o más bloques de giro del mismo tipo seguidos, porque se hace un giro de 180 grados. Se podrían hacer ejercicios sin esta premisa, pero no aportan valor didáctico y en los datos que ya había de la creación de mapas manual, este tipo de secuencias provocaba bloqueos a los alumnos. Estos bloqueos son porque además de la secuencia de instrucciones tienen que pensar cómo estaría orientado Robby y en qué casilla del mapa se encontraría con las solución que el alumno esté construyendo. Por esta razón no es una buena práctica utilizar este tipo de subsecuencias en una solución.

- No usar por norma general, dos instrucciones contrarias seguidas, es decir: avanzar, retroceder y giro derecha, giro izquierda. Por norma general porque en ocasiones se utilizan elementos interactivos en el mapa que sí permitirían la secuencias contrarias de desplazamiento de casillas. En el caso del primer par se pueden hacer mapas con trampolines o teletransportadores en el mapa justo en la posición en la que Robby tenga que ejecutar esos movimientos. En esta norma sí que tiene cierto valor didáctico para el alumno, le aporta complejidad justificada al problema y ayuda a dividir el problema en subsecuencias. Estas subsecuencias existen para los alumnos porque hay envíos de soluciones que llegan hasta ese elemento interactivo pero no sigue la secuencia hasta la bandera. De esta manera pueden comprobar si están construyendo bien la solución.

- La última instrucción debe ser un desplazamiento, no puede ser un giro o el bloque de acción. Esto es debido a que en todos los mapas hay que llegar a la bandera y para ello debe de haber al menos un desplazamiento de casilla.

3.1.2. Primera fase del generador

3.1.2.1. Combinaciones con instrucciones disponibles

En prácticamente todos los ejercicios de robótica, hay menos bloques disponibles de los se van a tener que utilizar para generar una solución. Solamente no siguen esta regla los ejercicios que van después de los tutoriales y hay alrededor de quinientos ejercicios en producción. Los ejercicios que no siguen esa norma podrían representar un 0.02% del total de ejercicios de robótica. Pero, en el primer diseño del algoritmo, y en su primera implementación, no se planteó esta realidad. Eso fué un fallo que ocupó más tiempo del deseado.

La primera idea era crear conjuntos de combinaciones. Si se utilizaban combinaciones con repetición *n* elementos a partir de un conjunto de *m* elementos, se podrían generar las soluciones necesarias para crear una unidad didáctica. Para ello el conjunto m debe ser mayor que n.

Teniendo un set X = {a, b, c, d} se pueden crear las siguientes combinaciones de cuatro elementos disponibles tomados de tres en tres: `aaa`, `aab`, `aac`, `aad`, `abb`, `abc`, `abd`, `acc`, `acd`, `add`, `bbb`, `bbc`, `bbd`, `bcc`, `bcd`, `bdd`, `ccc`, `ccd`, `cdd`, `ddd`.

Cuando se detectó el problema en el diseño e implementación de este algoritmo, la primera idea fue hacer combinaciones del resultado de la primera combinación, pero no se llevó a cabo. Además no es una solución eficaz ni elegante.

Potencialmente puede ocurrir en cualquier intento de generación porque de media hay 3.8 instrucciones por cada unidad didáctica y las soluciones contienen más instrucciones. A partir de cierto nivel del plan de estudios incluso el doble de tamaño. Teniendo ahora esto como premisa, se intentaron resolver casos base como tener un conjunto de una sola instrucción y que se quiera repetir más de una vez, este caso es sencillo, se clona la instrucción hasta llegar al número necesario para crear la solución.

Cuando se definieron los casos base del problema, el algoritmo debe seleccionar cada instrucción por orden y repetirlo junto al resto de elementos para llegar al tamaño requerido de la solución. Ej: X = {a, b}, k = 4, las combinaciones serían: `aaab`, `aaba`, `abaa`, `baaa`, lo mismo con b y repitiendo las dos letras a la vez, sería: `aabb`, `abba`, `bbaa`, `abab`, `baba`.

3.1.3. Segunda fase del generador

3.1.3.1. Permutaciones

En este momento del proyecto se descubrió el grave problema y la ineficiencia que había supuesto esas últimas ideas del diseño del generador. Para arreglar este problema se siguió recurriendo a la combinatoria, pero esta vez usando permutaciones. Se usó este tipo de combinación porque en la última idea de la primera fase, básicamente se estaba permutando un conjunto de mayor tamaño creado a partir del conjunto base.

Para hacer el generador más completo, aprovechando que se iba a crear un nuevo conjunto con las instrucciones sobre las que iba a permutar, ese nuevo conjunto iba a tener instrucciones obligatorias con su número de repeticiones definido. Con este extra la persona que quiera generar una unidad didáctica puede hacer que obligatoriamente se use un conjunto de instrucciones, que no todas tienen que estar en el conjunto por defecto definido para esa unidad didáctica.

También se corrigió en esta fase, que no se trabajase sobre el conjunto de instrucciones por defecto de cada unidad, porque ese conjunto sólo define los tipos de instrucciones, no hay repetición.

Con las premisas anteriores se crea el nuevo conjunto con el que se permutará para crear todas las soluciones posibles. Para facilitar la generación, antes de permutar instrucciones repetidas lo que se permutan son los índices de esas instrucciones.

Después de generar todas las soluciones, se pasan por los filtros de las reglas didácticas y así tener el conjunto que sí se puede usar para la unidad didáctica.
Permutar escala en coste rápidamente según se añaden instrucciones a la cantidad total de la solución, en esta fase se ha usado programación dinámica para que sea más eficiente el generador. Para diseñar este algoritmo la principal referencia ha sido el libro de (Verdejo, A. et al.) en el que se explica en el cálculo de de los coeficientes del triángulo de Pascal de la forma combinatoria, bastante parecido al problema de permutación que se usa en el generador. Se usa una matriz para calcular el coeficiente de las combinaciones. Pero si se quiere mostrar únicamente la combinación mayor del triángulo de Pascal, no se necesita almacenar todas los subproblemas anteriores, solamente los de la fila anterior para calcular los de la fila actual.

4. Arquitectura e implementación del sistema de generación de soluciones

En este capítulo se explicará la parte técnica del proyecto. La arquitectura que se ha utilizado, la implementación de las distintas etapas del diseño del generador. También se mencionan las pruebas locales.

4.1. Arquitectura

El proyecto utiliza Spring, el framework de Java más famoso para seguir la arquitectura modelo vista controlador. Se ha creado un proyecto con Maven que usa Spring Web y Thymeleaf[6] como dependencias.
El modelo se va construyendo con las variables que se han definido en el controlador principal y único ya que no es necesario más de un controlador para mostrar en una página web un listado de soluciones. En un futuro puede que haya otros controladores si se hace una comunicación con una base de datos y se quieren mostrar las distintas generaciones de las unidades didácticas.
Como controlador frontal, Spring tiene por defecto *ServletDispatcher*, que gestiona las pocas peticiones que tiene el generador de soluciones. Estas peticiones filtradas por este servlet llegan al controlador de la vista html que se comunica con el servicio que utiliza la lógica del proyecto para llamar al generador de soluciones.
Además del controlador está el servicio, donde se implementa la lógica que comunica el controlador con la lógica del generador. En un futuro si se avanza en el proyecto y se crea una base de datos, también iría la comunicación con los repositorios de las entidades, que con las etiquetas de Spring y JpaRepository es muy fácil de implementar esta comunicación.
Por último, está la vista, que utiliza Thymeleaf como motor de plantilla para representar en una interfaz, el modelo de la arquitectura. Para aplicar estilos a la vista se ha utilizado Bootstrap y para las funciones javascript se ha usado jQuery.

[6] https://www.thymeleaf.org/

4.2. Implementación del generador de soluciones

Para crear el listado de soluciones se toman como parámetros de entrada:

- *defaultBlocks*: es una lista de las instrucciones definidas para cada unidad didáctica. Solamente hay una aparición por cada tipo de instrucción. Ej: `{"advance", "backwards", "turnRight", "turnLeft", "action"}`
- *r*: es el número de instrucciones que debe tener la solución.
- *mandatoryBlocks*: son los bloques que tiene que estar obligatoriamente en la solución con las repeticiones en las que aparezca cada uno. No tiene por qué usar todos las instrucciones disponibles y además se puede añadir instrucciones que no estén definidas para esa unidad. Ej: `{"advance", "advance", "advance", "turnRight", "jump", "backwards"}`

Con estos parámetros de entrada se crea el nuevo conjunto de instrucciones sobre el que se va a combinar para generar las soluciones.Ej sobre los parámetros anteriores: `{"advance", "backwards", "turnRight", "turnLeft", "action", "jump"}`. Se compara el tamaño de este nuevo conjunto con el tamaño requerido para las soluciones.

En los siguientes apartados se explican los pasos intermedios y la generación elegida dependiendo del tamaño de este conjunto. Después de estos pasos se discriminan las soluciones que no son candidatas.

4.2.1. Combinaciones

Si el tamaño del conjunto creado a partir de los parámetros es mayor que el requerido para la solución, se utiliza el algoritmo diseñado en la primera fase del generador. Este algoritmo consiste en combinar recursivamente. Se va eligiendo todos los elementos posibles de uno en uno en un array de índices para la combinación que se está creando. Como son combinaciones con repetición no se tiene en cuenta si ha sido elegido ya el pivote sobre el que se está combinando en la recurrencia. Cuando el índice por el que se va iterando se convierte en el número de instrucciones que tiene que tener la solución se almacena esa nueva combinación en el listado de soluciones.

4.2.2. Permutaciones

Cuando el tamaño del conjunto creado es igual o menor al del tamaño de la solución, se utiliza el algoritmo implementado en la segunda fase, permutaciones usando programación dinámica. Primero se crea un array que marca cuántas repeticiones por instrucción va a haber en la solución. Si el conjunto creado por

los parámetros es igual al del tamaño de la solución, habrá una repetición por cada instrucción del conjunto. Pero si el conjunto es menor, los huecos restantes se rellenan aleatoriamente de entre el conjunto creado. Entonces quedaría un nuevo conjunto que tiene mínimo el número de repeticiones que define *mandatoryBlocks* más aleatoriamente la diferencia entre *r* con el conjunto creado por los parámetros. Una vez determinadas las repeticiones, se despliega lo que sería ya de por sí una permutación y se crea un array de índices para hacer las permutaciones.

Para permutar por programación dinámica se necesita saber primero cual es el resultado del factorial del tamaño de la solución, para saber el número total de permutaciones. Después se crean dos listas vacías de permutaciones, una con la que tendrá la permutación anterior y otra que tendrá la permutación actual.

Se comienza añadiendo a la lista anterior la lista de índice de las instrucciones, que como se ha comentado antes, era la primera permutación. Después se comienza a iterar sobre el tamaño de la solución con el iterador como pivote para las nuevas permutaciones. Se vacía el listado de permutación actual y se rellena con las permutaciones que se calculen sobre la anterior.

Para calcular cada permutación y añadirla al listado de permutaciones se trabaja con tres iteradores, el principal de ellos sobre el tamaño total de la solución y de los otros, para que sea más fácil de entender: *i, j*; *j* itera hasta el iterador principal. Si coinciden en valor *j* y el iterador principal, el valor de esa posición de la permutación será el pivote, que es un parámetro de la función que se encarga de construir las permutaciones. Ese pivote es el iterador que se menciona en el párrafo superior. Si *j* y el iterador principal no son iguales, se va utilizando cada elemento de la permutación referente desde el principio iterando con *i* hasta que *j* llegue al iterador principal. Una vez rellenada la permutación, se añade al listado de las ya creadas y se vuelve a repetir el proceso.

4.3. Evaluación de la corrección de la implementación

Las pruebas del proyecto se han dividido en dos tipos según el entorno. Primero se han hecho pruebas en el entorno local del proyecto. Estas pruebas se han hecho al final de cada fase de desarrollo del generador. Las pruebas en local se hicieron en los sprints seis y diez. Además en los primeros sprints del proyecto se iba probando según se terminaba una parte del algoritmo de combinaciones. Como no terminaba de funcionar como se esperaba, se empezó a sospechar que el diseño podría ser incorrecto o la implementación.

Al final de la primera fase del generador se comenzaron las pruebas locales sobre el proyecto, no fue necesario probar muchas unidades del plan de estudios para observar resultados inesperados o incluso errores. Gracias a este hito se pudo rectificar a tiempo el diseño, la eficiencia y la consistencia del generador.

En la segunda fase, al centrarse mucho en hacer el diseño correcto y eficiente, la implementación fue más sencilla. No tuvo que haber pruebas intermedias con el planteamiento de las permutaciones y después mejorar el algoritmo con programación dinámica. Con las recomendaciones del tutor del proyecto se implementó el algoritmo usando programación dinámica directamente. Gracias a sus indicaciones, el repaso del mal planteamiento de la primera fase y el rediseño, las pruebas fueron mucho mejor. En las primeras generaciones podía llegar a haber hasta tres millones de soluciones, lo cuál no tenía sentido. No tenía sentido porque no se estaban aplicando las reglas didácticas, además en las siguientes generaciones seguía habiendo errores revisando las soluciones porque no se incluyeron algunas de las reglas que estaban definidas. Cuando se corrigieron estas carencias, el número de soluciones con los mismos parámetros cayó en picado, que era lo que se esperaba más o menos.

Después de estas pruebas en local se cambia el entorno a probar con alumnos. Se escogieron un total de diez soluciones aleatorias de tres unidades didácticas diferentes y se crearon mapas manualmente a partir de esas soluciones. Las tres unidades didácticas eran las que más tráfico de alumnos tenían en el momento en el que se subieron a producción estos ejercicios. No se guardaron los parámetros con los que se crearon las siguientes soluciones. Pero sí las soluciones porque no se crearon los mapas en el mismo momento ni en el mismo terminal. Puede que no coincidan los índices de las soluciones con las que tenían. Son los datos que se han conseguido recuperar, porque las pruebas no se hicieron pensando en la memoria del proyecto, como se puede observar en la falta de índice de la décima solución.

Tras estas pruebas sobre el generador se hicieron pruebas sobre la interfaz de usuario del proyecto. Se probó en un dispositivo móvil y en los terminales de trabajo del proyecto para comprobar que era un diseño adaptable. También se comprobó el funcionamiento de los elementos con los que se podía interactuar.

En las siguientes páginas se muestra cada mapa que corresponde a cada solución generada por la última versión implementada.

Solución 10799: ["jump", "jump", "jump", "advance", "turnRight", "advance", "turnLeft", "advance"]

Solución 19: ["advance", "advance", "jump", "turnLeft", "jump", "jump", "turnRight", "advance"]

5. Evaluación

El objetivo de estas pruebas es valorar que los ejercicios creados a partir de las soluciones del generador tengan al menos la misma calidad que los ejercicios creados manualmente que ya estaban en *Coding*.

La evaluación del generador de soluciones se ha lanzado sobre el entorno de producción de *Coding*. Para no contaminar las pruebas los alumnos están haciendo los ejercicios como en una sesión común de *Coding*. Estos ejercicios solo los han probado los alumnos que hayan estado trabajando en esas unidades.

Algunos de estos ejercicios tienen muy pocas soluciones enviadas entonces sería prematuro tomar decisiones sobre estos casos. Aún así se evalúan igual que el resto de ejercicios.

Para evaluar los datos, se ha utilizado la herramienta de Google para transformar datos en tablas y gráficas, Data Studio. Esta herramienta ya se usaba en Smartick para valorar todo tipo de datos, y por mi parte no he tenido que desarrollar nada de esta parte del proyecto, sólo evaluar los datos.

5.1. Esquema de evaluación

Para evaluar los datos se necesitan comprender algunos parámetros que se usan en las tablas y gráficas de Data Studio.

- Tabla de lecciones:
    - Superación: Es un porcentaje que determina la media de en qué envío de cada alumno ha sido la solución correcta. Cada alumno tiene un porcentaje de superación de 1 / su envío * 100. Por ejemplo, el primer ejercicio del generador que prueban los alumnos tiene una superación media del 34,95%. Este porcentaje se traduce en que el envío con la solución correcta es el envío 2,86.
    - Retroceso: Cuando un alumno ha realizado diez envíos de solución incorrectos, se le propone repasar o seguir intentando el ejercicio. Cuando un alumno acepta repasar se cuenta como retroceso. Por ejemplo, el primer ejercicio del generador tiene un 3,23% de retroceso. Eso significa que de los 426 alumnos que han enviado al

menos una solución, 14 ha aceptado repasar los ejercicios anteriores de la unidad didáctica dónde está el ejercicio.

    - Tiempo de respuesta: es la media de segundos que se tarda en enviar la solución desde que se le presenta el ejercicio. Este dato no suele ser muy determinante entre los ejercicios de robótica. Se usa más para predecir cuántos ejercicios por sesión podrían resolver a ese ritmo. Este parámetro sí que tendría valor si el generador se utilizara en la propia sesión del alumno. Por ejemplo si el alumno tarda de media 29 segundos en enviar una solución, como en el caso del primer ejercicio del generador, en una sesión podría ser capaz de hacer 31 ejercicios de la unidad que esté trabajando.

    - Alumnos: el número total de alumnos que ha enviado solución. En el ejemplo.

    - Versión: los resultados se pueden filtrar entre diferentes versiones para comparar los cambios que se han hecho para mejorar un mapa.

    - Unidad: se puede filtrar por unidad. Esto es esencial para ver la evolución de la dificultad en los ejercicios de una misma unidad. También se puede comparar los datos de todos los ejercicios en conjunto para compararlo con otras unidades y decidir si es necesario ubicarla en otra posición del plan de estudios.

- Tabla de soluciones: se repiten algunos parámetros y se añaden los siguientes:
    - Solución: Este parámetro almacena cada solución diferente enviada para un mismo ejercicio.
    - Correcta: es un booleano que marca si la solución es correcta o no.
    - Estrellas: es la calidad de la solución determinada por el número de intentos y el número de instrucciones usadas. Si se envía en el primer intento con el mismo tamaño que la solución ofrecida, son 3 estrellas. Si solamente se cumple que es correcta y una de las dos condiciones, 2 estrellas. Si solo es correcta 1 estrella.
    - num_respuestas: es el número de envíos de cada tipo de respuesta

- Tabla de soluciones por alumno: se enumeran todas las soluciones de todos los alumnos que han hecho un ejercicio. En el caso del primer generador, ha habido 426 envíos de soluciones.

    - Alumno: es el id del alumno. Por seguridad no se utilizará en esta memoria.

- Intentos: número de intentos del alumno para el ejercicio que ha enviado
- Solución: booleano para marcar si ha llegado al número de repeticiones del ejercicio en el que se muestra la pregunta de ver la solución del ejercicio.
- Solución aceptada: booleano que marca si han aceptado la pregunta de ver solución.
- Retroceso: booleano para marcar si ha llegado al número de repeticiones en el que aparece la pregunta de repasar ejercicios de la unidad que está trabajando.
- Retroceso aceptado: booleano para marcar si se ha aceptado la pregunta del dato anterior.

5.2. Metodología de evaluación

Para evaluar las soluciones del generador, se crearon diez ejercicios a partir de tres generadores de diferentes unidades. Las tres unidades a las que se añadieron estos ejercicios, eran las que más alumnos tenían la última semana antes de la subida a producción de estos ejercicios.

Primero se escogieron soluciones al azar de entre las generadas por cada unidad y después se crearon mapas a partir de esas soluciones. El equipo didáctico revisó los ejercicios para que se pudieran subir a producción y el 24 de agosto de 2022 los alumnos que estaban trabajando esas unidades pudieron hacer los ejercicios del generador como un ejercicio más de la sesión.

Para evaluar estos ejercicios se introdujeron en mitad de cada unidad para que ya tuvieran una introducción de los conceptos que se trabajan en esa unidad y se pudieran mezclar con los datos de otros ejercicios que ya estaban en producción y han sido creados manualmente.

El propio proceso de evaluación de los datos de los mapas está dividido en vistas desde una más general a una mucho más concreta. Además, como se indica en el planteamiento de la evaluación en los párrafos anteriores, también hay una parte de comparación con el resto de ejercicios de alrededor que ya estaban en producción. Se hará una comparación de los resultados de las unidades en la versión anterior, cuando no estaban los ejercicios del generador. También se valorarán los ejercicios del generador en los ejercicios posteriores de las propias unidades.

5.3. Resultados

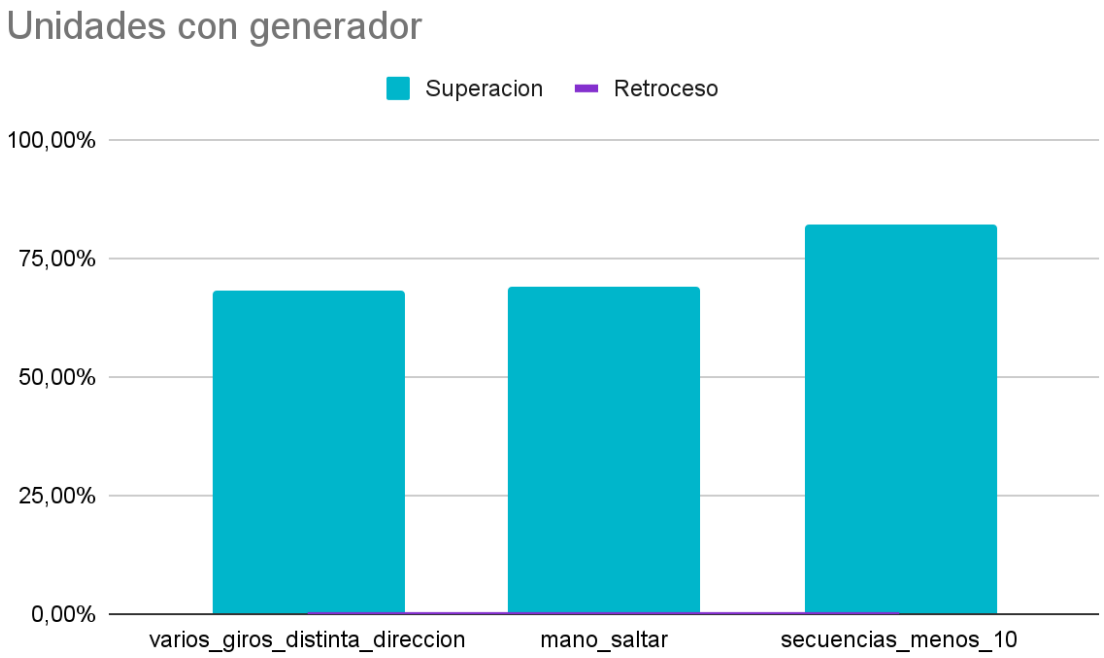


Gráfico 1. Superación y retroceso de las unidades de los ejercicios del generador.
La media de superación de los ejercicios del generador está por encima del 65%

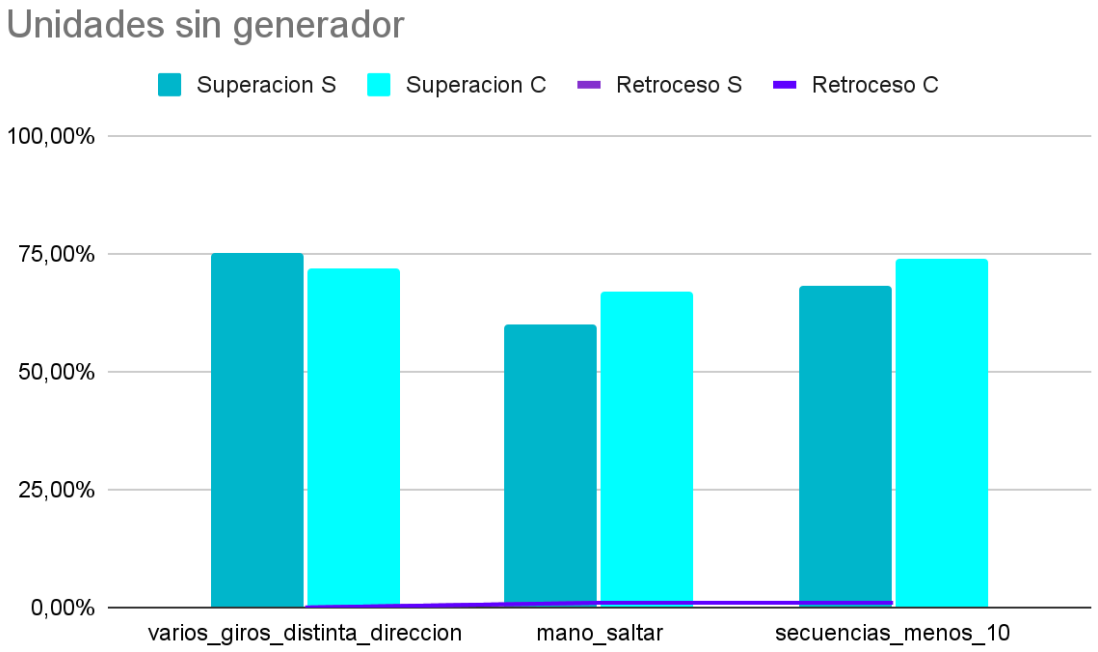


Gráfico 2. Superación y retroceso de las unidades de los ejercicios sin generador y con generador
En la primera unidad empeora un 3% pero mejora un 7% en las otras dos

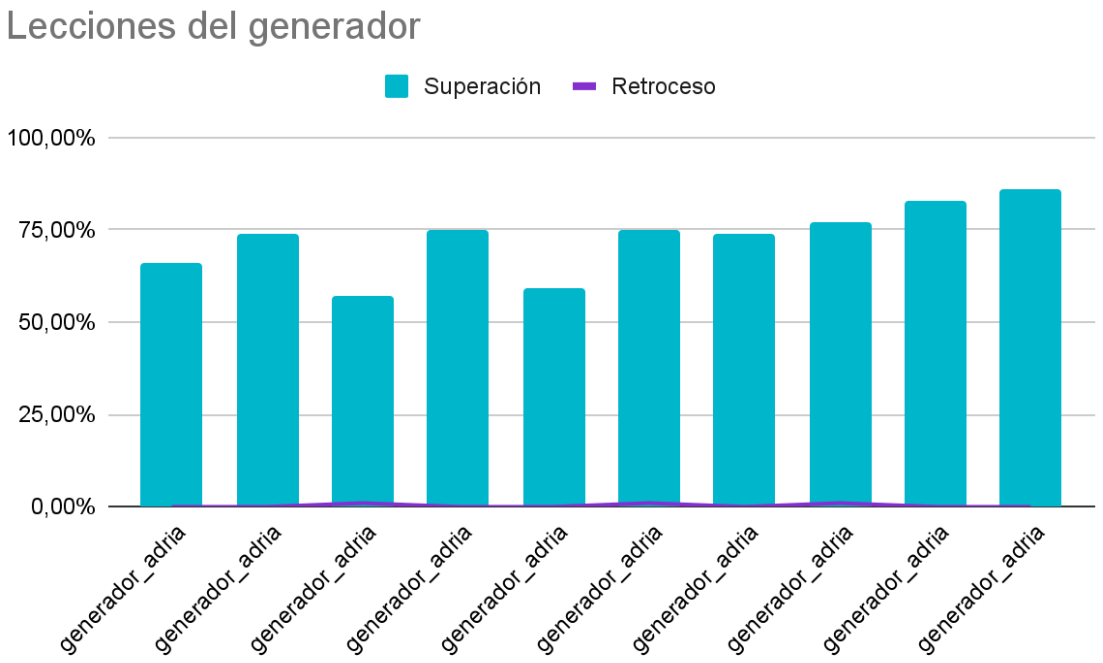


Gráfico 3. Superación y retroceso de los ejercicios con generador
El 70% de los ejercicios tienen una superación mayor al 70%, el resto superior a 55%

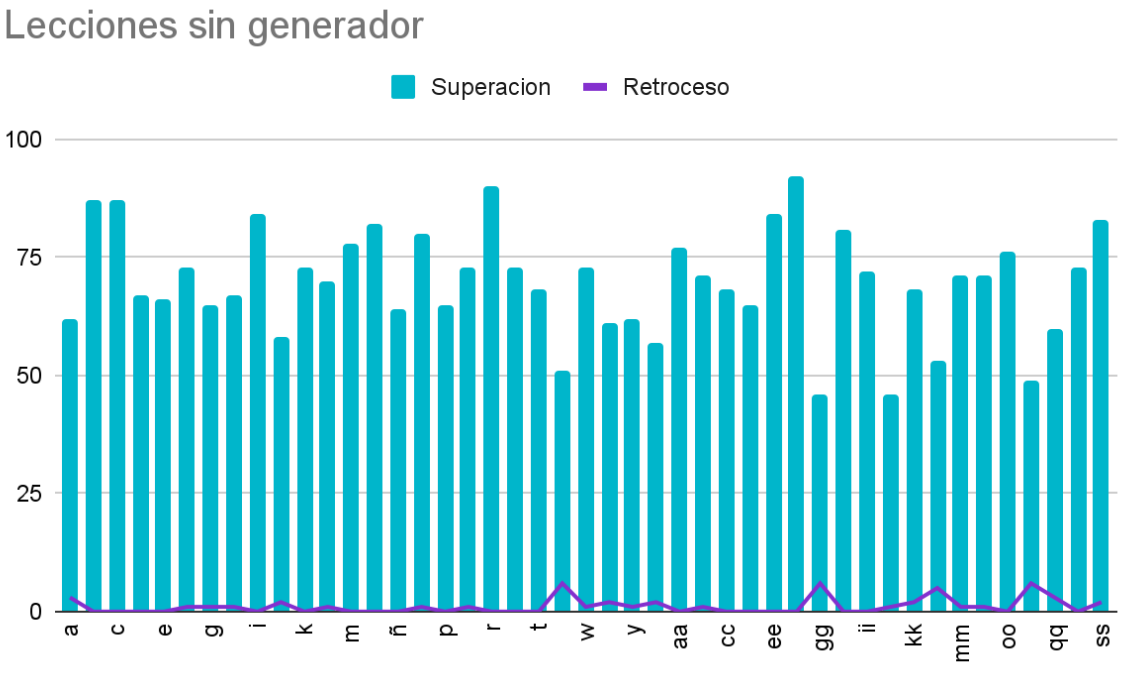


Gráfico 4. Superación y retroceso de los ejercicios de alrededor sin generador
La media de superación de estos ejercicios es del 70%

Unidad	I3	P1	P2	P3	P4	P5	P6	S1	S2
varios_giros_distinta_direccion	-	55	55	63	68	67	74	76	73
mano_saltar	61	66	52	65	65	69	76	79	81
secuencias_menos_10	-	80	61	77	82	87	86	89	87

P: Primaria S: Secundaria

Tabla 1. Superación por unidad y curso
Generalmente la superación mejora por curso y por unidad

Unidad	exercise_name	I3	P1	P2	P3	P4	P5	P6	S1	S2
varios_giros_distinta_direccion	generador_adrian_1	-	47	51	61	66	63	75	78	72
	generador_adrian_2	-	76	62	70	73	75	78	84	75
	generador_adrian_3	-	40	48	52	53	59	61	65	66
	generador_adrian_4	-	60	61	70	80	70	81	80	78
mano_saltar	generador_adrian_5	50	54	41	53	53	56	68	72	81
	generador adrian 6	100	86	58	74	68	75	79	88	74
	generador adrian 7	33	58	59	68	72	76	81	77	86
secuencias_menos_10	generador adrian 8	-	79	48	69	76	84	84	80	86
	generador adrian 9	-	83	60	75	79	89	90	92	94
	generador adrian 10	-	77	78	85	89	87	84	95	81

P: Primaria S: Secundaria

Tabla 1. Superación por ejercicio y curso
Vista más detallada de la tabla anterior

exercise_name	% Superación	% Retroceso	Tiempo	Alumnos
generador_adrian_1	66,07	0,41	00:00:24	490
generador_adrian_2	74,17	0,21	00:00:26	485
generador_adrian_3	56,83	0,81	00:00:28	492
generador_adrian_4	74,63	0,43	00:00:25	462
generador_adrian_5	58,64	0,25	00:00:27	396
generador_adrian_6	75,01	0,76	00:00:28	397
generador_adrian_7	74,03	0,48	00:00:27	418
generador_adrian_8	76,61	0,67	00:00:31	298
generador_adrian_9	83,09	0	00:00:31	294
generador_adrian_10	86,09	0	00:00:20	302

Tabla 2. Superación, retroceso, tiempo de envío y número de alumnos por ejercicio
El superación es buena, el retroceso bajo, el tiempo es parecido en todos los casos y el número de alumnos es lo suficientemente alto como para analizar con seguridad los resultados

solution	correcta	stars	num_respuestas
[jump action turnRight advance jump action advance turnLeft advance]	1	1	418
[jump turnRight action advance jump action advance turnLeft advance]	1	1	11
[jump action turnRight advance jump action turnLeft advance]	0	0	11
[jump action turnRight advance jump action advance turnRight advance]	0	0	44
[jump action turnRight advance jump action jump turnLeft advance]	0	0	43
[jump action turnRight advance jump action turnRight advance]	0	0	15
[jump action turnRight advance jump action advance turnLeft]	0	0	10
[jump action turnRight advance jump jump action turnLeft advance]	0	0	10

Verde: correcto Naranja: falta una instrucción para ser correcto Rojo: instrucción equivocada

Tabla 3. Soluciones de generador_adrian_3

El 80% de las instrucciones son correctas en las soluciones incorrectas

alumno_smartick	Fecha	intentos	Solución	Solución aceptada	Retroceso	Retroceso aceptado
1	30 ago 2022	18	1	1	2	2
2	30 ago 2022	13	0	0	2	1
3	9 sept 2022	11	0	0	1	1
4	25 ago 2022	9	0	0	0	0
5	2 sept 2022	8	0	0	0	0
6	2 sept 2022	8	0	0	0	0
7	8 sept 2022	8	0	0	0	0
8	7 sept 2022	8	0	0	0	0
9	4 sept 2022	8	0	0	0	0

Tabla 4. Soluciones de alumnos generador_adrian_5

La superación baja del ejercicio 5 se puede atribuir a casos aislados con muchas repeticiones

5.4. Análisis de los resultados

En este apartado se van a explicar, justificar y entender los datos representados en cada una de las tablas. Como se ha comentado en apartados anteriores. Las Tablas están distribuidas desde información más general, a nivel de unidad o curso, a nivel más específico, de ejercicio, soluciones más enviadas o alumnos concretos con muchos intentos para un mismo ejercicio. En general los resultados de las pruebas verifican el buen funcionamiento del generador de soluciones y de la aplicación de normas didácticas.

5.4.1. Análisis de las tablas por unidades

Las tablas por unidades, que son la Gráfico 1 y Gráfico 2, muestran la superación y retroceso de las unidades que contienen los ejercicios del generador. En la Gráfico 1 aparecen los datos de la versión anterior a la que se añadiera la versión y en la Gráfico 2 los datos de los ejercicios del generador.

Los resultados de los ejercicios del generador a nivel de unidad están relativamente bien, recordar que un porcentaje de superación por encima del 50% significa que de media se ha superado entre el primer y segundo intento. También es cierto, que un porcentaje alto de superación no tiene por qué ser del todo bueno, puede significar que es un ejercicio que no aporta nada didáctico al alumno.

Con más detalle, en la primera unidad los ejercicios del generador tienen de media un 68% de superación. En la versión anterior sin los ejercicios del generador la unidad tuvo un 75% de superación y ahora un 72%. Sabiendo que no ha habido más cambios en esa unidad que los nuevos ejercicios del generador, se ha empeorado los resultados un 3%. El porcentaje de retroceso apenas ha variado, por lo que se puede suponer que los alumnos necesitan algún intento más para conseguir el ejercicio pero sin llegar a tantas repeticiones como para repasar.

Separar estos datos en dos tablas ha servido para saber cuál ha sido el problema, además de que los resultados del resto de ejercicios de la unidad apenas ha variado de una versión a otra. La colocación de los ejercicios del generador ha sido manual, el número de instrucciones es ligeramente mayor a la media de instrucciones de esa unidad. Como tienen más instrucciones que el resto, una solución sería que fueran al final de la unidad. También se pueden utilizar menos instrucciones en el generador para ver cómo afectaría a la superación de la unidad.

Las otras dos unidades se han comportado de una manera similar entre ellas. Han mejorado los resultados de la unidad alrededor de un 7%. Esa mejora se observa en los ejercicios que aparecen después de los ejercicios del generador. Además de que los ejercicios del generador tienen buena superación.

5.4.2. Análisis de las tablas por lecciones

En la Gráfico 4 podemos observar todos los ejercicios de las unidades a las que pertenecen los ejercicios del generador. Están ordenados según aparecen en el plan de estudios. En la Gráfico 3 solo aparecen los ejercicios del generador.

La media de superación de los ejercicios que no son del generador es de 69.8%, que comparándola con los del generador es 2,8% peor. Esto verifica que los ejercicios del generador son ligeramente mejores en la misma situación, porque recordar que las pruebas no se han hecho en un entorno especial, es una sesión para el alumno igual que cualquier otra que hayan hecho hasta ahora.

Aunque la media es mejor, los ejercicios del generador de las dos primeras unidades podrían tener mejor resultado. El conjunto de esos ejercicios, del primero al séptimo, tienen una media de superación del 68,5% que es menor al de los ejercicios que no son del generador. Se revisarán las posiciones de los ejercicios del generador en sus correspondientes unidades y se estudiarán las soluciones propuestas por los alumnos. Como se hará en el apartado 4.4.5.

5.4.3. Análisis de las tablas por cursos

La Tabla 1 y la Tabla 2 muestran la misma información, pero a diferentes niveles. Esto se hace para poder filtrar qué contenido es el que hay que mejorar y no tener que revisar todas las superaciones de los ejercicios para cada curso, en una situación habitual fuera del proyecto.

En ambas tablas se puede observar que los resultados mejoran, como normal general, según se avanza de curso. Sorprendentemente hay datos de tercero de infantil, que no es nada habitual que tengan Coding activado debido a los requisitos que pone Smartick matemáticas. También se puede observar comparando los datos con las tablas anteriores, que los ejercicios con menos superación se comportan muy parecido en el rango de cursos, los resultados pueden variar ±15%.

Estas tablas no están para valorar el resultado del proyecto en sí. Están para justificar que en un futuro, se debería aplicar al generador filtros por curso, para que el contenido sea más personalizado y se potencien las habilidades

respectivas de cada curso. Con este proyecto a futuro se aseguraría un aprendizaje equitativo para todos los cursos, de manera automática.

5.4.4. Análisis de la Tabla 2

En la Tabla 2 se puede estudiar mejor los resultados de la Gráfico 3. Revisando las columnas de izquierda a derecha podemos ver que tan solo un 30% de los ejercicios del generador están por debajo del 70% pero están por encima del 55%. Son datos alentadores para justificar el buen funcionamiento del generador. Además, el retroceso no supera en ningún caso el 1%, habiendo ejercicios en el plan de estudios de robótica que tienen un 5% de retroceso o en algunos casos extremos un 9%. El generador podría mejorar el retroceso de esos ejercicios, escogiendo mejor el orden de las instrucciones o reduciendo el número de instrucciones para la solución.

El tiempo de respuesta es bajo, como el resto de ejercicios de esas unidades. Se puede inferir que son igual de fáciles de entender. Para el generador de soluciones sigue justificando su rendimiento igual que el resto de datos. Este dato servirá mejor para las ampliaciones a futuro cuando se implemente un generador de mapas.

Por último se puede observar el número de alumnos que ha enviado intentos para cada ejercicio, justifica el valor de los datos de todas las tablas.

5.4.5. Análisis de las tablas por soluciones

En las últimas dos tablas (Tabla 3 y Tabla 4) se han escogido los ejercicios del generador con peores resultados. Con los resultados de estas tablas se puede analizar con precisión cuál ha sido el problema y si es localizado en algunos alumnos u ocurre generalmente en las sesiones.

Primero, en la Tabla 3, se revisan las soluciones correctas, en caso de que haya algunas con menos instrucciones de las propuestas por la solución modelo. En un futuro, con el generador de mapas, en principio no debería haber ninguna mejor que la solución modelo, a menos que tenga un valor didáctico de encontrar caminos ocultos que sean mejor que el evidente.

Recordar que la superación de este ejercicio es de 56,83%, la peor de los ejercicios del generador, entonces habría que observar los resultados enviados por los alumnos. En la Tabla 3, podemos observar que la mayoría de soluciones son casi correctas. Solo tienen un 20% de la solución incorrecta, en la mayoría usan una instrucción incorrecta, esto puede ser problema de la orientación espacial del mapa, que está en vista isométrica.

En la Tabla 4 se podrá estudiar si la superación baja y el porcentaje de retroceso es algo que ocurre generalmente o es en casos especiales. Se ha escogido el ejercicio con peores datos por alumno, y la superación baja no es de forma general, es porque un número pequeño de alumnos ha tenido que repetir el ejercicio muchas veces. Incluso hay tres que han llegado a aceptar el repaso de la unidad y uno de ellos incluso ha aceptado ver la solución. En este caso se revisará la trayectoria de ese alumno para ver qué hacer y mejorar su experiencia y resultados.

6. Discusión

Revisando los resultados de los análisis de los datos obtenidos en las pruebas con los alumnos, se puede sacar una conclusión general positiva. Se ha mejorado el rendimiento de estas unidades gracias a los ejercicios del generador, recordar que esas unidades no se han modificado anteriormente por varias versiones por lo que los datos no están contaminados con otros cambios fuera de este proyecto. Aunque, está claro que hay un 30% de ejercicios que no han dado tan buenos resultados, incluso en alguno de ellos se puede decir que ha habido malos resultados. Estos ejercicios deben ser estudiados con todos los datos que hay disponibles para hacer una siguiente versión del generador de soluciones, ya sea creando reglas didácticas nuevas, revisando los parámetros de generación, eligiendo mejor las mejores soluciones de cada generación, de momento manualmente.

Aunque en el mejor de los casos, se ha mejorado un 7% la superación de dos unidades didácticas, como es la primera versión funcional del generador de soluciones, hay margen de mejora tanto en la generación como en la gestión de las soluciones. Se puede añadir el estudio de datos al generador para puntuar las soluciones y que el generador tenga cierta inteligencia artificial y clasifique y estime, la superación de nuevas generaciones basándose en datos de generaciones anteriores.

Estudiando los resultados de la primera unidad de la Gráfico 2, junto con la Gráfico 3 y la Gráfico 4, claramente el fallo ha sido la pronta colocación de los ejercicios del generador en la unidad didáctica. Se deben repartir los ejercicios por la parte final de la unidad, eso mejoraría los datos de estos ejercicios y no bajaría la superación de los que ya estaban en producción las versiones anteriores.

Se puede discutir también cómo mejorar los ejercicios problemáticos revisando los datos de la Tabla 3. Se puede suponer y hacer la hipótesis, de que la elección de esa solución generada, es correcta. Entonces el problema pasaría al siguiente nivel, la creación del mapa, de momento, manual. Primero, se observa que al menos, el 55% de la solución desde la primera instrucción es una elección correcta, han entendido el camino que hay que seguir hacia la bandera. Después de ese 55% puede haber entre 12% y 22% de fallo, ese 12% ocurre en el 85% de las soluciones incorrectas. Sería muy acertado pensar que ajustando el mapa, ya sea marcando esa parte del camino a la bandera, si el problema de esos alumnos que han fallado ha sido confundir la

altura de un tile o que falte un avance. Otro ajuste, que es el más usado en todo el plan de estudios, es modificar la orientación del mapa, recordar que usa una vista isométrica, esto es importante para que los alumnos tengan más claro, qué orientación tendría Robby cuando tenga que hacer el último giro para llegar a la bandera. Con estos ajustes, se podría suponer que un 29% de esos envíos de soluciones erróneas, o bien se podrían evitar directamente en un caso idílico o reducir con los cambios propuestos.

Una vez repasados los fallos del generador, que sorprendentemente han sido pocos y a priori, fáciles de corregir, hay que centrarse en las faltas del generador. Con las hipótesis de partida de la generación de mapas y el estudio de la generación de mapas por el método de Monte Carlo, los resultados de las pruebas confirman que hay una parte de la creación de ejercicios. Por los datos, una parte casi tan importante como la solución y las reglas didácticas, en concreto puede ser de un 22% como se menciona en el apartado anterior, es la creación del mapa. La creación del mapa es necesaria incluirla después de la generación de soluciones, con la hipótesis de partida y las hipótesis de este apartado del proyecto, es prácticamente un acierto implementar una generación de mapas para mejorar los resultados de los alumnos en el plan de estudios de didáctica. Es cierto, que este desarrollo es mucho más ambicioso, mucho más complicado que el generador de soluciones y es mucho más sencillo que en las primeras versiones de este nuevo generador, las pruebas den malos resultados por la mezcla de datos de soluciones y mapas.

7. Conclusiones y trabajo a futuro

Este proyecto ha cumplido las expectativas en cuanto a la importancia que tiene la solución por sí sola. Hasta la creación de esta tecnología se creaba una solución manualmente que tenía que ser ideada por una persona del equipo didáctico. Si esa solución no funcionaba en las pruebas del equipo se tenía que pensar una nueva. Este proceso puede ser muy lento pero si se tiene a disposición del equipo didáctico una gran cantidad de opciones para una unidad didáctica que además puede tener las instrucciones exactas que se necesitan, el trabajo es mucho más fiable y rápido. Tener en unos segundos 31679 soluciones diferentes para una misma unidad, facilita mucho el trabajo al equipo didáctico.

Los resultados de las pruebas con alumnos también han dado pistas de la distribución de importancia que se reparten la solución y el mapa. La solución siempre es lo primero, pero si no hay un mapa bien estructurado y que sea fácil de entender, puede generar una complicación innecesaria y hasta frustración en un alumno. Por con los malos resultados que también ha tenido el generador de soluciones, es fácil justificar que el siguiente paso a futuro es crear un generador de mapas a partir de las soluciones creadas por esta tecnología. Además, cuando se comenzó a diseñar el proyecto, el enfoque era hacia este generador, entonces ya hay definiciones y referencias que servirán de gran ayuda.

Con los resultados por curso, también se puede hacer una nueva hipótesis de partida a futuro. Esta nueva hipótesis consiste en personalizar los ejercicios según el curso en el que esté el alumno. Así se puede enseñar los mismos conceptos a todos los alumnos con distintos escalones de dificultad que se adapten a las capacidades de cada curso.

También se puede observar la importancia del momento en el que el alumno trabaja cada ejercicio. Los ejercicios del generador que tienen malos resultados podrían tener mejores si se avanzan de posición para que lo trabajasen más tarde. Esto puede ser otra hipótesis para una versión avanzada del generador de soluciones. En una versión avanzada el generador podría utilizar resultados de generaciones anteriores para presentar el orden didáctico de cada ejercicio de una unidad didáctica. Los parámetros tendrán que ser más complejos para que ocurra esto, es ineficiente que todos los ejercicios de una unidad tengan la misma cantidad de instrucciones y un set de instrucciones prácticamente idéntico.

Como se explica al final del párrafo anterior, es ineficiente que todos los ejercicios de una unidad tengan la misma cantidad de instrucciones y un set de instrucciones prácticamente idéntico. En el estado actual de la tecnología habría que ejecutar varias generaciones por tamaño. En hitos a futuro se deberían poner rangos de instrucciones para que la generación sea más variada y más completa.

La conclusión más importante, después de presentar todas las anteriores, es que gracias a esta tecnología y a los resultados tan favorables, se puede mejorar mucho la calidad de aprendizaje de un alumno utilizando todos los criterios didácticos en distintos generadores para presentar mayor cantidad de contenido, con mucha más variedad y lo más importante, que ese contenido pueda ser personalizado a las necesidades y capacidades de cada alumno.

7.1. Trabajo a futuro

Ahora que se ha automatizado parte del proceso de creación de un ejercicio e incluso de una unidad didáctica, la motivación ha evolucionado a automatizar por completo la creación de un mapa, tanto solución como tablero.

Cuando se consiga esa generación, el siguiente paso es dotar de inteligencia a estos generadores. Que usen datos de generaciones anteriores para adaptarse a los resultados de los datos que dan los alumnos y de esta manera mejorar mucho la calidad de contenido que recibe cada alumno en todas sus sesiones de robótica.

7.1.1. Hipótesis para el trabajo futuro

La nueva hipótesis ya no es igualar la calidad de la generación manual en cuanto a superación de cada ejercicios si no de superar la calidad y poder corregir en una cantidad muy inferior de tiempo, un mapa problemático en el plan de estudios de robótica.
Si la hipótesis se cumple en las siguientes etapas de desarrollo con sus correspondientes pruebas con alumnos, se plantearía una versión mejorada que genera contenido adaptado al alumno en cada sesión y a partir de esa versión mejorada se mantendría esta tecnología.

7.1.2. Objetivos para el trabajo futuro

Los objetivos para futuro son en orden cronológico:

- Generar mapas con las tecnologías estudiadas en el apartado de estado del arte.
- Procesar los resultados de las generaciones con minería de datos para crear perfiles de alumnos basados en superación y curso en el que están.
- Utilizar los perfiles creados por el apartado anterior para mejorar la calidad de las generaciones y dotar de inteligencia artificial a los generadores.
- Utilizar los generadores para que trabajen bajo demanda de cada alumno en su sesión según el contenido que están trabajando para que así los mapas estén personalizados hacia las capacidades de cada alumno.

8. Introduction

Education has been digitalized in the last decade at its different levels and with its different peculiarities. In universities, baccalaureate degrees have been introduced and virtual campuses have been established. In some institutes and schools, applications are used to reinforce the learning of different subjects. This is known as educational technology.

Digitization has been accelerating both in the classroom and in extracurricular activities due to the volume of students and the very different needs that they may have in the classroom, in addition to recent years due to the pandemic. This need is covered by online applications such as Smartick, the company that this project is aimed at.

To meet the needs of the student in the classroom using these applications, there has to be a considerable amount of content to cover the speed of learning and a great variety in that content to avoid a stagnation in some concepts that may be more difficult for them.

Covering this content obviously requires a teaching team to create a subject syllabus, a learning strategy to cover most of the spectrum of different types of learners, and content creation technology.

There are learning applications for various subjects that offer a replica of the content of textbooks used in the classroom, others have their own content. However, not all applications can offer a large quantity, variety and adaptation of content focused on the student to enhance their learning and not only cover the curriculum.

Smartick is an online learning method with several programs to choose from. One of them is mathematics, it combines programming, calculation and logic. The combination is done in sessions distributed daily. In order of daily frequency, calculation is the type of session with the highest volume, followed by logic and finally programming, or as most people who know the product name it, *Coding*.

Coding is the programming product in which the student is introduced to the process of solving problems with code. To teach this introduction, a curriculum with concepts spread over different activities is used. Some of these concepts are taught in the robotics curriculum. Very briefly, you have to take a robot (Robby) to a flag on the map that the student is working on at that moment.

But, online teaching has disadvantages. One of the disadvantages of educational technology is that its implementation is uneven due to the digital divide. Initiatives, corporate social responsibility strategies, scholarships are created, but the volume of students is so immense that it is difficult to alleviate this problem. Other disadvantages not as serious as the previous one is the dehumanization of education. Smartick has a pedagogical team to cover this problem in a certain way. But as a general rule, in students who use online learning with any application or service, they have a certain degree of digital dependency. It can harm skills that are worked in the classroom such as writing, crafts. If the learning sessions are extended there may be distractions, not only with the environment but also with other installed applications. It can also create problems of social interaction because online learning usually takes place outside of school and at home.

8.1. Motivation

For a few years the content of this part of the product has been done manually, under didactic supervision to give quality to each map that the students make in their learning curve.

This is where this tool comes in, it saves a lot of work for the didactic team and brings great variety to each didactic unit of the curriculum. As data, in the Coding production environment, there are around five hundred exercises of this type in the study plan, the generator is capable of generating more than the number of exercises for each didactic unit.

This tool aims to remove as much workload as possible from the teaching team in this part of the curriculum so that they can dedicate their time to the most complex part of the product. If new robotics content is required in the session itself, if there are adverse student results in some units, if there are students who after their routine session want to make more maps to practice, they can be done with this tool.

8.2. Hypothesis

This project starts with the hypothesis that if the generated maps are used with the same criteria as those created manually, there will be much more variety in the robotics curriculum, there will be no unnecessary difficulties that we have

not been able to avoid with the manual generation and students will be able to practice more the concepts they are working on at that moment.

If the hypothesis is true, the project would continue to be developed with the objectives defined for the future. The next milestone would be to generate complete maps that meet exactly the same requirements as those created manually.

8.3. Objectives

The objective of this project is to create a tool for generating didactic exercises for the Smartick robotics product so that it can be used in different areas of the company. These areas can be to renew or increase the content of the student's session, also for the post session content, create demos for the sale of the product, create reduced curricula for camps. Increase the daily content of the Smartick face-to-face academy.
To achieve this business objective, the objectives of the project have been to design this tool so that it generates solutions automatically, taking as reference some didactic guidelines that can be determined by the didactic team. Besides, in the generation of the solutions, the didactic rules that already comply with the exercises that exist in the *Coding* production environment must be complied with.
As a technological objective, the generation must be efficient and the user experience must be intuitive. An algorithm will be designed as efficiently as possible in execution time and memory use. A simple interface will also be designed to be able to use the solution generator.
The objectives of this work are:

- Design of a tool that generates solutions for robotics maps of the Smartick product
- Automation of solution generation
- Easy-to-enter didactic parameters to improve the usability of the tool
- All solutions must meet the didactical criteria
- Efficient generation of solutions
- Implement model view controller for future database communication with tool interface
- Create a graphical interface for easier usability
- Tests on real students that verify the starting hypothesis
- Plan the following milestones in the future based on the results of the tests

8.4. Methodology

8.4.1. Technologies and tools used

Sections of the scrum agile development methodology have been used. The project has been done by one person, so the guidelines of this methodology in the meetings have not been used. But the sprint concept has been used, using two weeks of development as a period, except in the test sprints, which have been one week. Retrospectives of the results of the local tests have also been made to make decisions in the next steps of the development of the project.

To create and manage tasks, the Notion application has been used, it is a project management software very similar to Jira, from Atlassian. In this application the main tasks of the project have been created in order to have a global view at any time. It has been very easy to separate the main part of the project (all the tasks related to the generator), from the additives such as the map generator and to make the service visual and easy to use. In this way it has been possible to prioritize tasks by importance and level.

GitHub has been used as a version control system. It has allowed us to keep the project constantly updated to be able to work on any device that has been used for development and to have an accessible repository without fear of losing the progress of the project.

The project is developed in IntelliJ IDEA which is a very powerful and pleasant development environment for the programmer. It provides many facilities to be able to use Spring as a Java framework and also gives small tips to improve details of the code and make it more efficient.

For the project interface, some libraries have been necessary but others have been for convenience to develop. Bootstrap has been used to add styles to the page where the generator options are selected and the list of solutions is displayed. It has been used for communication with the local server and for web editing and logic, AJAX and jQuery. To facilitate the work of javascript development, it has been programmed in ECMAscript 6, but since this version does not work in all browsers, the running javascript files are transpiled, thanks to Gulp.js and Babel, which have been installed in the project using Node.js

8.4.2. WorkPlan

The work plan that has been followed is the same as in the Smartick work environment, to maintain familiarity. Project management software has been used first. In this case, Notion has been chosen because of its similarity to Jira and its free license. It is a very intuitive software although with fewer possibilities. For example, in Jira you could have created epics or user stories, to create and manage the high-level tasks and to differentiate between technical tasks or project customer-facing tasks. Even with these limitations, it has been possible to organize the tasks of the project and maintain an overview of the progress, reviewing the tasks that were not started, those that were in progress and those that were completed.

Once the project roadmap has been planned, a repository has been created on GitHub so that there is a continuous integration of the project and to be able to save any progress or test throughout the development, either because a functionality has been developed or because had to continue on another device in the middle of development. Thanks to IntelliJ it has been very easy to follow this agile methodology because it has a very comfortable and secure GitHub integration.

Here are the high-level tasks in chronological order:

1. Study of the generation of solutions with instructions available
2. Add mandatory instructions to generation
3. Application of combinatorics to generate solutions
4. Define and generate the didactic rules
5. Select candidate solutions with the didactical rules
6. Check the operation and efficiency of the first version of the generator
7. Fix bugs in the first version
8. Create a multiset to cover the difference in size of the set and the size of the solution
9. Application of dynamic programming to the generation of solutions
10. Check the operation of the new version
11. Preparing controllers and services for the generator web page
12. Creating the view and styles for the generator web page
13. Layout of the view
14. visual tests
15. Map generator design from solution
16. Didactic rules for generating maps

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
O1										
O2										
O3										
O4										
O5										
O6										
O7										
O8										
O9										
O10										
O11										
O12										
O13										
O14										
O15										
O16										

O: Goal S: Sprint
Blue: Completed Orange: Future

Table 1. The tasks have been distributed in sprints (S1…S10) that have occupied two weeks each. In the case of the tests, they have been carried out in the middle of a sprint.

8.5. Document structure

In chapter two the references of the project are explained. The main references are the Smartick math product exercise generators. Smartick's programming product, Coding, is explained and put in context. The projects that had initially been taken as a reference to generate maps are mentioned and explained: AlphaGo and a map generator from the Sokoban video game. The references that have been taken into account to develop the project are also mentioned: combinatorics for the generation of solutions and the use of dynamic programming to improve performance.

In chapter three the entire design of the solution generator is exposed. The instructions that the students have to use to solve the exercises are explained. The didactic rules that all the solutions must apply in each generation so that they can be used on a map are argued.

Chapter four explains the technology development part. The architecture used and the reasons are presented. The implementations of the two implementation phases are also explained. In addition, at the end of the chapter the solutions used in the tests with students and how the students see the maps created from the different generations of the three didactic units are taught.

Chapter five details all the parameters used by the tables and graphs. It also discusses how the data has been evaluated. The tables and graphs necessary to understand the results of the generator are displayed. All this data has been extracted from Data Studio and has been extracted from student sessions. Finally, the results are analyzed in search of arguments that support the use of the generator and also of failures to improve future versions of it.

In chapter six the results of the project are discussed. Importance has been given and the worst results have been highlighted, that few exercises have been affected, to propose future initiatives and improve the tool.

In chapter seven, an approach similar to chapter two is made but focused on the future, with a new more ambitious starting hypothesis, with a motivation towards a much more elaborate and complex tool, justifying this scale with the good results of the tests. In addition, the major objectives of the following versions of the tool are listed until it can be considered complete if the last objective is met.

In chapter ten all the references of the project are listed. The references for the generators that were reviewed in the state of the art, which will serve for the following stages of the project. In addition there are also the references for the combinatorics approaches that are used in the solution generator.

9. Conclusions and future work

This project has fulfilled expectations in terms of the importance of the solution alone. Until the creation of this technology, a solution was created manually that had to be devised by a person from the teaching team. If that solution didn't work in the team's tests, a new one had to be thought of. This process can be very slow, but if a large number of options are available to the teaching team for a teaching unit that can also have the exact instructions that are needed, the work is much more reliable and faster. Having 31679 different solutions for the same unit in a few seconds greatly facilitates the work of the teaching team.

The results of the tests with students have also given clues to the distribution of importance between the solution and the map. The solution always comes first, but if there is not a well-structured map that is easy to understand, it can generate unnecessary complication and even frustration for a student. Due to the poor results that the solution generator has also had, it is easy to justify that the next step in the future is to create a map generator from the solutions created by this technology. Also, when the project began to be designed, the focus was on this generator, so there are already definitions and references that will be of great help.

With the results by course, a new starting hypothesis can also be made in the future. This new hypothesis consists of personalizing the exercises according to the course in which the student is. In this way, the same concepts can be taught to all students with different levels of difficulty that adapt to the abilities of each course.

The importance of the moment in which the student works each exercise can also be observed. Generator exercises that have poor results could have better results if they were moved up in position to be worked on later. This may be another hypothesis for an advanced version of the solution generator. In an advanced version, the generator could use results from previous generations to present the didactic order of each exercise of a didactic unit. The parameters will have to be more complex for this to happen, it is inefficient for all exercises in a unit to have the same number of instructions and a practically identical set of instructions.

As explained at the end of the previous paragraph, it is inefficient for all exercises in a unit to have the same number of instructions and a practically identical set of instructions. In the current state of technology you would have to run multiple generations per size. In future milestones, ranges of instructions should be included so that the generation is more varied and more complete.

The most important conclusion, after presenting all the previous ones, is that thanks to this technology and the very favorable results, the quality of a student's learning can be greatly improved by using all the didactic criteria in different generators to present a greater amount of content, with much more variety and most importantly, that content can be customized to the needs and abilities of each student.

9.1. Future work

Now that part of the process of creating an exercise and even a teaching unit has been automated, the motivation has evolved to fully automate the creation of a map, both solution and board.

When that generation is achieved, the next step is to equip these generators with intelligence. That they use data from previous generations to adapt to the results of the data that the students give and in this way greatly improve the quality of content that each student receives in all their robotics sessions.

9.1.1. Hypothesis for future work

The new hypothesis is no longer to match the quality of manual generation in terms of overcoming each exercise, but rather to exceed the quality and be able to correct in a much shorter amount of time, a problematic map in the robotics curriculum.
If the hypothesis is fulfilled in the following stages of development with their corresponding tests with students, an improved version would be proposed that generates content adapted to the student in each session and from that improved version this technology would be maintained.

9.1.2. Objectives for future work

Future goals are in chronological order:

1. Generate maps with the technologies studied in the state of the art section.
2. Process the results of the generations with data mining to create student profiles based on achievement and the course in which they are.
3. Use the profiles created by the previous section to improve the quality of the generations and provide artificial intelligence to the generators.
4. Use the generators so that they work on demand for each student in their session according to the content they are working on so that the maps are personalized towards the capacities of each student.

10. Referencias

Ahmed, Umair, et al. *Automatic generation of alternative starting positions for traditional board games.* https://research-explorer.app.ist.ac.at/record/5410.

Ashlock, Daniel y Justin Schonfeld. *Evolution for automatic assessment of the difficulty of sokoban boards.* https://ieeexplore.ieee.org/abstract/document/5586239.

Bernd Brugmann. *Monte Carlo Go*. http://www.ideanest.com/vegos/MonteCarloGo.pdf.

Cameron Browne. *Monte Carlo Tree Search*. https://web.archive.org/web/20160308043415/http://mcts.ai:80/index.html.

Chaslot, Guillaume Maurice Jean-Bernard Chaslot. *Monte-carlo tree search*. https://www.researchgate.net/profile/Mohamed-Mourad-Lafifi/post/Does_anybody_has_exprience_about_combining_monte_carlo_tree_search_and_evolutionary_algorithm/attachment/59d63ad279197b8077997e40/AS%3A407582138224641%401474186563237/download/Monte-Carlo+Tree+Search+Chaslot_thesis.pdf.

«Combinaciones con repetición». *wikipedia*, https://es.wikipedia.org/wiki/Combinaciones_con_repetición.

«Combinations with repetitions». *geeksforgeeks*, https://www.geeksforgeeks.org/combinations-with-repetitions/.

Dorit Dor y Uri Zwick. *SOKOBAN and other motion planning problems*. https://reader.elsevier.com/reader/sd/pii/S0925772199000176?token=D3CD5B71818B2195C714FF21CBFFCD220474F0445500838879AAD318F6F75F05CF9F3E8FC6DDF4D25C4257E3721A8C31&originRegion=eu-west-1&originCreation=20220915170633.

Luis Javier López Arredondo. «Permutaciones con repetición». *geocities*, 18 de octubre de 2006, https://www.geocities.ws/luisja80/esp/permutacionesrep.html.

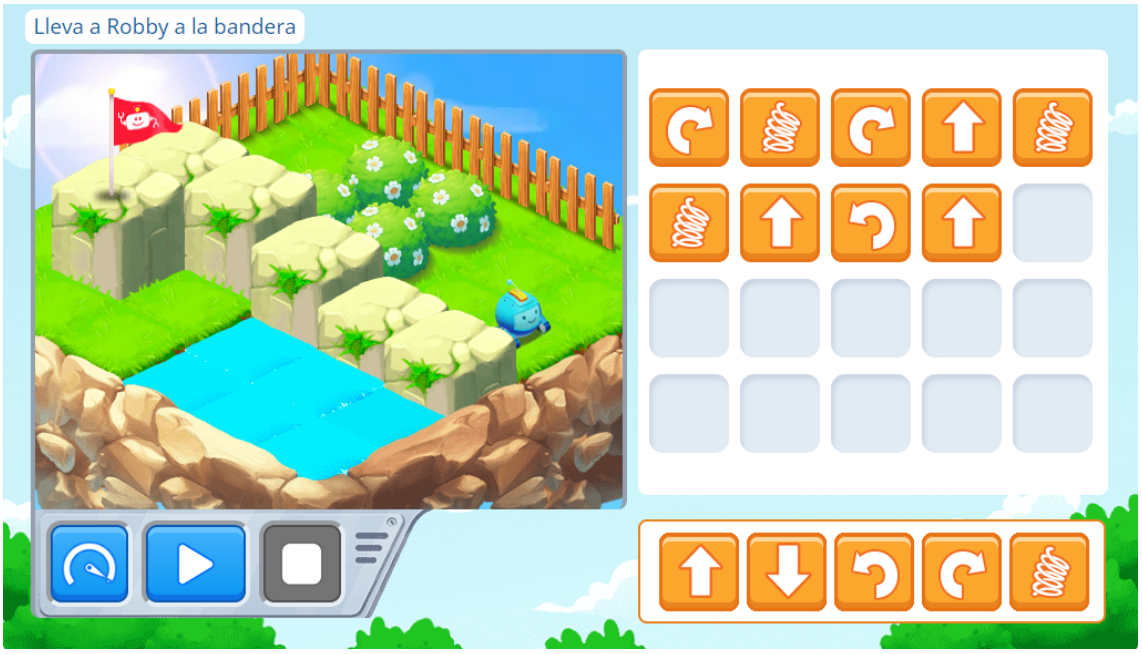
«Permutar un Array». *wextensible*, https://www.wextensible.com/temas/recursivos/permutar.html.

perspolis. «Calculating Permutation Using Dynamic Programming». *codeproject*, https://www.codeproject.com/Tips/891811/Calculating-Permutation-Using-Dynamic-Programming.

Taylor, Joshua y Parberry, Ian. *Procedural generation of sokoban levels*. https://ianparberry.com/techreports/LARC-2011-01.pdf.

Verdejo, A., et al. *Estructuras de datos y métodos algorítmicos: 213 ejercicios resueltos.* 2ª Edición, Garceta Grupo Editorial, 2013.

I. Apéndice

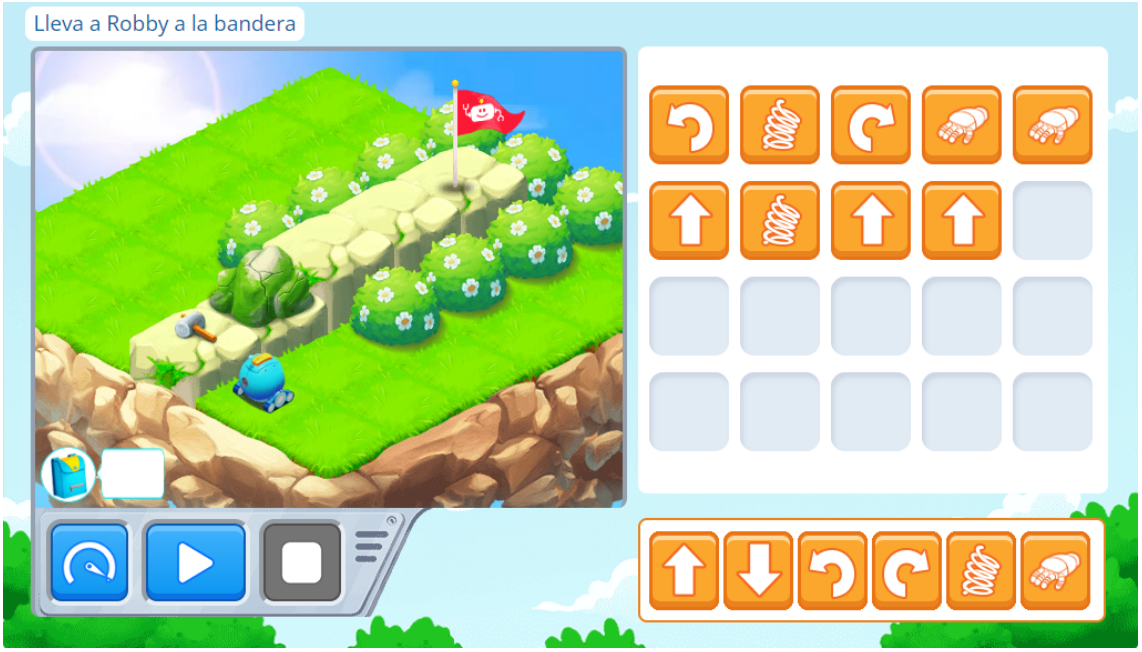


Solución 5760: ["turnRight", "jump", "turnRight", "advance", "jump", "jump", "advance", "turnLeft", "advance"]

Solución 8455: ["turnLeft", "jump", "turnRight", "advance", "advance", "jump", "jump", "advance"]

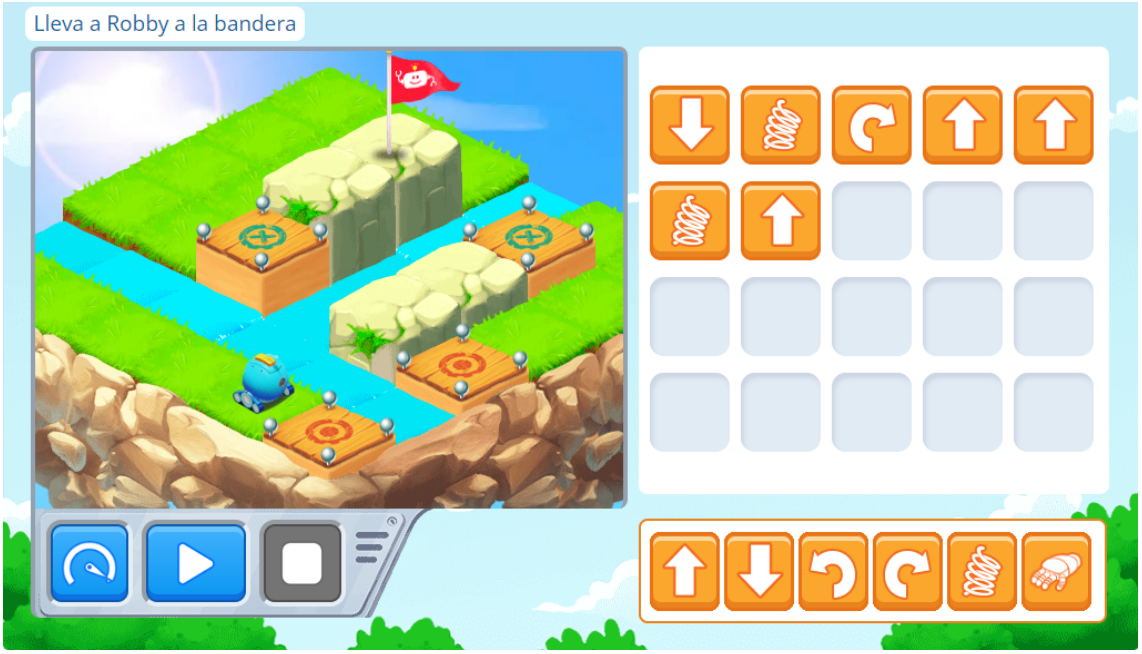
Solución 42500: ["jump", "action", "turnRight", "advance", "jump", "action", "advance", "turnLeft", "advance"]

Solución 42774: ["turnLeft", "jump", "turnRight", "advance", "action", "jump", "action", "advance", "advance"]



Solución 29: ["turnLeft", "jump", "turnRight", "action", "action", "advance", "jump", "advance", "advance"]

Solución 260: ["advance", "advance", "jump", "turnRight", "backwards", "jump", "advance"]



Solución 589: ["backwards", "jump", "turnRight", "advance", "advance", "jump", "advance"]

Solución X: ["advance", "jump", "advance", "advance", "jump", "turnRight", "backwards"]