UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA INFORMÁTICA

Enseñanza de algoritmos de inteligencia artificial en una plataforma para aprendizaje en-línea de la programación

Teaching artificial intelligence algorithms in an online coding platform for kids

Realizado por
Miguel Mejía Jiménez

Tutorizado por
Francisco José Vico Vela

Departamento
Lenguajes y Ciencias de la Computación
Universidad de Málaga

MÁLAGA, SEPTIEMBRE, 2021

Abstract

Educating the population about Artificial Intelligence (AI), its possibilities, its risks and its real scope, is a 21st century need whose urgency grows every day. The contribution of the University of Malaga in this line of work is an online teaching platform developed by the Dept. of Languages and Computer Science: ToolboX.Academy.

The purpose of this work is to study the possibility of including AI algorithms among the contents taught in the platform. For this purpose, a formal analysis of the environment on which to define these algorithms is carried out, the classical AI problems with didactic potential that have been considered are listed and the selection of those that have been implemented is justified. The results of the work are then presented, both for the prototype version and for the final version of the teaching module dedicated to AI. These results include the code written for the platform libraries and the tasks that make up the teaching module. The conclusion discusses the difficulties that have been faced and takes them as a reference to point out the challenges that would appear in the implementation of new algorithms. Finally, some ways of approaching the extension of the AI module in the platform and the modifications that this would require in the platform are discussed.

Keywords: digital literacy,AI literacy, artificial intelligence, programming, teaching

Resumen

Educar a la población en torno a la Inteligencia Artificial (IA), sus posibilidades, sus riesgos y su alcance real, es una necesidad del siglo XXI cuya urgencia crece cada día. La aportación de la Universidad de Málaga en esta línea de trabajo es una plataforma de enseñanza en línea desarrollada por el Dpto. de Lenguajes y Ciencias de la Computación: ToolboX.Academy.

El propósito de este trabajo es estudiar la posibilidad de incluir algoritmos de IA entre los contenidos que se enseñan en la plataforma. Para ello, se realiza un análisis formal del entorno sobre el que definir estos algoritmos, se listan los problemas de IA clásica con potencial didáctico que han sido considerados y se justifica la selección de aquellos que se han implementado. A continuación, se exponen los resultados del trabajo, tanto los de la versión prototipo como los de la versión final del módulo de enseñanza dedicado a IA. Dichos resultados incluyen el código escrito para las librerías de la plataforma y las tareas que conforman el módulo de enseñanza. La conclusión del trabajo expone qué dificultades se han afrontado y las toma de referencia para señalar los retos que aparecerían en la implementación de nuevos algoritmos. Por último, se plantean algunas maneras de abordar la extensión del módulo de IA en la plataforma y las modificaciones que ésto exigiría en la misma.

Palabras clave: alfabetización digital, inteligencia artificial, enseñanza, programación

Índice

1. Introducción **1**

1.1. Motivación . . . 1

1.2. Objetivos . . . 3

1.3. Tecnologías usadas y metodología de trabajo . . . 3

1.4. Estructura del documento . . . 5

2. Desarrollo **7**

2.1. Introducción a ToolboX.Academy . . . 7

2.1.1. Interfaz . . . 7

2.1.2. Lenguajes de programación . . . 9

2.1.3. Entorno y narrativa . . . 9

2.2. Análisis formal del entorno y el agente . . . 10

2.2.1. Definiciones previas . . . 10

2.2.2. Acciones y percepción . . . 13

2.2.3. Naturaleza del entorno . . . 14

2.3. Estudio de posibles problemas de IA para implementar . . . 14

2.3.1. Búsqueda de caminos . . . 14

2.3.2. Optimización combinatoria . . . 15

2.3.3. Razonamiento . . . 16

2.3.4. Aprendizaje automático . . . 17

2.4. Elección de problemas . . . 18

2.4.1. Justificación . . . 18

2.4.2. Planteamiento formal . . . 19

2.5. Prototipo . . . 22

2.6. Implementación . . . 23

2.6.1. Módulos . . . 23

2.6.2. Funciones . . . 23

2.6.3. Tareas . . . 25

3. Conclusiones y Líneas Futuras **29**

3.1. Resultados excluidos 29

3.2. Extensión del módulo de IA 29

3.2.1. Mejora de los resultados existentes 29

3.2.2. Nuevos algoritmos 30

3.2.3. Experimentación sobre el entorno 32

3.3. Conclusión 33

1 Introducción

1.1. Motivación

La *alfabetización digital* se define en el Marco Global de Alfabetización Digital (DLGF) de la UNESCO (Law y col., 2018, p. 6) de la siguiente manera:

> *Digital literacy is the ability to access, manage, understand, integrate, communicate, evaluate and create information safely and appropriately through digital technologies for employment, decent jobs and entrepreneurship. It includes competences that are variously referred to as computer literacy, ICT literacy, information literacy and media literacy.*[1]

El término Inteligencia Artificial (*Artificial Intelligence*) no aparece en ningún momento en el DLGF, a pesar de que su presencia en la sociedad no hace sino crecer y de que sus riesgos y dilemas éticos (Dignum, 2018) son tan relevantes como otros que sí se han considerado. Por estos motivos, ya existe también un interés en educar a los ciudadanos en el concepto y las posibilidades de la IA; de realizar una *alfabetización en IA*. Long y Magerko (2020) realizan un estudio para su conceptualización y para establecer un marco de referencia tanto para desarrolladores como para educadores. La definición que proponen para la alfabetización en IA es:

> *A set of competences that enables individuals to critically evaluate AI technologies;*

[1] La alfabetización digital es la capacidad de acceder, gestionar, comprender, integrar, comunicar, evaluar y crear información de forma segura y adecuada a través de las tecnologías digitales para el empleo, el trabajo decente y el el espíritu empresarial. Incluye competencias que se denominan de diversas maneras: alfabetización informática, alfabetización en TIC alfabetización informacional y alfabetización mediática.Traducción realizada con la versión gratuita del traductor www.DeepL.com/Translator .

> *communicate and collaborate effectively with AI; and use AI as a tool online, at home, and in the workplace* [2].

Para responder a ambas necesidades (alfabetización digital y alfabetización en IA), en España existe el Instituto Nacional de Tecnologías Educativas y Formación del Profesorado (INTEF), la unidad del Ministerio de Educación que se encarga de la integración de las TIC en la educación preuniversitaria (INTEF, 2021b) y que incluye entre sus proyectos la Escuela de Pensamiento computacional e Inteligencia Artificial (INTEF, 2021a).

Una de las varias maneras de abordar la alfabetización en IA es a través de la enseñanza de la programación de ordenadores (en adelante sólo programación). En los últimos años, como parte de la alfabetización digital y con la creciente demanda de profesionales en informática, han aparecido a nivel internacional iniciativas para incluir la programación y el pensamiento computacional en el currículo escolar (Ottestad & Gudmundsdottir, 2018). En España esto se concreta en propuestas impulsadas por la Conferencia de Directores y Decanos de Ingeniería Informática (CODDII) y la Asociación de Enseñantes Universitarios en Informática (AENUI) (Llorens Largo y col., 2017), que se suman a las iniciativas que propiciarán la inclusión de los conceptos de IA en la enseñanza preuniversitaria.

Tras una búsqueda exhaustiva de herramientas para la enseñanza de programación para niños, queda claro que prácticamente la totalidad de las propuestas son plataformas en línea. Sólo algunas cuentan con contenido fuera de línea, que suele representar un pequeño porcentaje de su contenido total. Cabe mencionar algunas de las más relevantes, como *Scratch*, *CodeMonkey* y *Code.org*. La popularidad de las aplicaciones web como herramientas educativas se debe a que facilitan el acceso a recursos actualizados desde cualquier dispositivo con conexión a Internet. Además, en el caso concreto de la programación, evitan a los centros y a los alumnos tener que instalar software de desarrollo (compiladores, intérpretes, depuradores...) al ofrecerse como servicio SaaS (*Software as a Service*).

Por lo tanto, es lógico concluir que la alfabetización en IA se producirá en su mayor medida a través de este tipo de plataformas en línea y que para participar en ella es necesario competir en ese mismo medio.

[2] Un conjunto de competencias que permite a las personas evaluar de forma crítica las tecnologías de la IA; comunicarse y colaborar de forma eficaz con la IA; y utilizar la IA como herramienta en línea, en casa y en el lugar de trabajo. Traducción realizada con la versión gratuita del traductor www.DeepL.com/Translator

1.2. Objetivos

Definir el dominio de aprendizaje

El objetivo de este trabajo es explorar las posibilidades de enseñanza de algoritmos de IA a estudiantes preuniversitarios. El primer reto es delimitar qué se pretende enseñar, porque la etiqueta IA abarca un conjunto difuso de técnicas, que cambia con el tiempo; en palabras de Hofstadter (1979), «IA es todo aquello que todavía no ha sido concretado».

Launchbury (2017) realiza una distinción en tres olas de IA: 1) una primera ola de técnicas de *conocimiento manufacturado*, también llamada en la bibliografía IA clásica, simbólica o tradicional, 2) una segunda ola de técnicas de *aprendizaje estadístico*, también llamada IA moderna -que incluye la IA bioinspirada-, y 3) una futura ola de técnicas de *adaptación contextual*. Será necesario realizar un análisis sobre los algoritmos pertenecientes a los dos primeros grupos, para seleccionar aquellos con mayor potencial didáctico. Esta selección no es intuitiva, porque a pesar de que las técnicas clásicas son una continuación natural del aprendizaje de la algoritmia necesario durante el aprendizaje de la programación, también pueden volverse más complejas y difíciles de ilustrar que algunas técnicas modernas.

Implementar e ilustrar los algoritmos en una plataforma real

Para enseñar un algoritmo, es conveniente no sólo describirlo, sino ilustrarlo. Por eso, incluimos entre los objetivos de este trabajo la implementación de los algoritmos que se quieran enseñar y de las utilidades necesarias para representar gráficamente el proceso de cómputo que los estudiantes deban comprender. La implementación se realizará en una plataforma existente para la enseñanza de la programación, de manera que, por lo menos, se pueda plantear la posibilidad de realizar pruebas orientadas a casos de uso reales.

1.3. Tecnologías usadas y metodología de trabajo

Las herramientas y la metodología usadas durante el desarrollo del trabajo se describen a continuación.

- La plataforma de enseñanza en línea elegida es ToolboX.Academy [3], desarrollada por el

[3] https://toolbox.academy

Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga (Vico y col., 2019). El *frontend*, donde se realizará la integración de los módulos de IA, es una aplicación web programada con *Vue.js* [4]. Por ello, la programación de los módulos de IA se realiza en Javascript. Aún así, el prototipo de los mismos se realiza en GNU/Octave.

- La interfaz gráfica del mundo de Roby (ver 2.1) está programada en *Phaser* [5]. La programación de las utilidades para ilustrar los algoritmos se hará usando este mismo *framework*.

- Las actividades o tareas de la plataforma están especificadas en ficheros JSON, almacenados en un repositorio aparte. Para su procesado automático, se utilizan *scripts* de Bash y la utilidad JQ [6].

- Para la grabación de los vídeos de demostración de las tareas de IA diseñadas, se utilizan scripts de Bash y la utilidad *xdotool* [7].

- El control de versiones se realiza con Git. Tanto el repositorio de la aplicación como el del contenido están alojados en *Bitbucket* [8].

- La colaboración con el resto del equipo se ha coordinado a través de *Trello* [9], utilizando una metodología ágil (*Kanban*) donde se realizan reuniones periódicas para comprobar el progreso del proyecto y se usan tarjetas para representar las tareas pendientes para completar los objetivos marcados.

[4] *Vue.js* es un *framework* basado en *Node.js* para construir interfaces de usuario. https://vuejs.org/

[5] *Phaser* es un *framework* para desarrollo de juegos en HTML5 para escritorio y dispositivos móviles. https://phaser.io/

[6] JQ es un procesador de ficheros en formato JSON para línea de comandos. https://stedolan.github.io/jq/

[7] *Xdotool* es una herramienta que usa librerías del sistema gestor de ventanas de algunos entornos de escritorios de Linux para simular entrada de ratón y teclado. https://www.semicomplete.com/projects/xdotool/

[8] *Bitbucket* es un servidor Git con una interfaz web y un sistema de usuarios y permisos sobre repositorios. https://bitbucket.org

[9] *Trello* es una aplicación web para organizar equipos mediante listas de tareas pendientes. https://trello.com

1.4. Estructura del documento

En la sección de Introducción (1) del trabajo se plantea la motivación y los objetivos del trabajo y se presentan las herramientas que se usan.

En la sección de Desarrollo (2), se hace una presentación de la plataforma ToolboX.Academy y un análisis formal en base al cual se deciden los problemas que se pueden representar en ella y qué modificaciones previas le hacen falta. A continuación, de entre los problemas estudiados, se seleccionan uno o varios para los que se implementan las soluciones de IA. Sin entrar en detalles del código, se explican las librerías escritas con las soluciones y las utilidades para ilustrarlas. Esta sección termina con una explicación de las tareas diseñadas para la enseñanza de estos algoritmos.

En la sección de Conclusiones y Líneas Futuras (3) se cierra el trabajo explicando las dificultades que se han afrontado durante el trabajo, repasando los problemas de IA que todavía se pueden implementar y proponiendo algunas maneras de continuar el desarrollo de la plataforma a partir de los resultados obtenidos.

2 Desarrollo

2.1. Introducción a ToolboX.Academy

La plataforma de ToolboX.Academy ha sido desarrollada de manera independiente a la realización de este trabajo. A continuación se describirán sus características y funcionamiento, con el objetivo de establecer el marco de desarrollo del módulo de enseñanza en IA.

Los cursos de ToolboX.Academy se componen de una serie de tareas propuestas para que el alumno las resuelva mediante código. Los programas que se introducen como soluciones para cada tarea son traducidos a una secuencia de movimientos por parte de un avatar dentro de un entorno gráfico con disposición de cuadrícula.

2.1.1. Interfaz

Como se muestra en la figura 1, la interfaz de ToolboX.Academy está dividida en tres partes principales: 1) arriba a la izquierda, la ventana del enunciado y los mensajes, 2) abajo a la izquierda, la interfaz gráfica del entorno - que en adelante llamaremos **mundo de Roby** - y 3) a la derecha, el editor de código. En la ventana del enunciado aparece la descripción de la tarea, típicamente la definición del objetivo del programa que debe escribir el alumno. El editor de código ofrece resaltado sintáctico y numeración de líneas. El botón *Run*, situado sobre la ventana del enunciado, inicia la ejecución el programa, haciendo que el avatar se mueva según las instrucciones del usuario e imprime la salida en la pestaña de mensajes, que adquiere el foco automáticamente, como se ve en las figuras 2 y 3. Los botones de *Tips* y *Wiki* abren ventanas emergentes con pistas e información útil relacionada con la tarea. En caso de que la tarea se resuelva con éxito, se muestra un mensaje de felicitación y un botón para pasar a la siguiente tarea (ver fig. 2), si la hay. En caso contrario, se muestra un mensaje de error y un botón para reiniciar el mundo de Roby (ver fig. 3).

Figura 1: Interfaz de ToolboX.Academy antes de ejecutar el código del alumno.

Figura 2: Interfaz de ToolboX.Academy después de ejecutar el código del alumno, si el resultado es correcto.

Figura 3: Interfaz de ToolboX.Academy después de ejecutar el código del alumno, si el resultado no es correcto o la ejecución ha fallado.

2.1.2. Lenguajes de programación

Actualmente ToolboX.Academy cuenta con soporte para dos lenguajes de programación en sus tareas: ToyScript, un lenguaje específico de la plataforma (Vico, 2020), y Javascript. El primero, al ser un lenguaje diseñado para la enseñanza de conceptos fundamentales de programación, tiene ciertas restricciones de diseño, como la ausencia de subrutinas y de tipos complejos, que obstaculizaría el desarrollo de algoritmos más avanzados. Por lo tanto, las tareas propuestas en este trabajo están pensadas para cursos en Javascript. Esto ha facilitado notablemente la integración de las librerías en la aplicación web.

2.1.3. Entorno y narrativa

El entorno gráfico dónde se mueve el avatar tiene la temática de una fábrica de robots animada, para resultar amigable y atractiva. El agente principal que ejecuta el código del alumno es Roby, un pequeño robot que cumple tareas que le asignan Tor y Eva, los otros robots de la fábrica. Los principales elementos que se encuentra Roby en el entorno son tuercas y tornillos que puede recoger, puertas etiquetadas, cajas y otros robots, que contienen información necesaria para la tarea, y muros que no puede traspasar. En la figura 4 se muestran estos elementos en la aplicación.

Las reglas del mundo son las siguientes:

- Roby sólo puede moverse a casillas que estén dentro de la cuadrícula y en las que no haya un muro (ver fig. 5).
- Si Roby se mueve a la posición de una tuerca o un tornillo, ésta o éste desaparecen de la cuadrícula y se incrementa un contador interno de elementos recogidos (comparar fig. 1 y fig. 2).
- Roby sólo puede obtener información de un robot o una caja si se encuentra en su misma posición (ver fig. 6).

2.2. Análisis formal del entorno y el agente

El primer requisito para una propuesta de tareas de IA en ToolboX.Academy es hacer una definición formal del entorno, para que sirva de referencia en el estudio de los posibles problemas y soluciones de IA que tiene sentido implementar.

2.2.1. Definiciones previas

Definición 2.1 (Estado de una casilla)**.** Definimos el conjunto de estados de una casilla como

$$S = \{empty, occupied\,\langle obj, lab, val\rangle\,, takeable\,\langle obj, lab, val\rangle\}$$

donde obj y lab (objeto y etiqueta) son cadenas de texto y val (valor) un valor numérico.

Definición 2.2 (Entorno de una tarea)**.** Definimos el entorno de una tarea como una tupla (R, C, X) donde $R \times C$ son las dimensiones de la cuadrícula (filas por columnas) y X es el conjunto de casillas, definidas por su fila, su columna y su estado:

$$X = \{(i, j, s) \mid 0 < i \leq R,\, 0 < j \leq C, s \in S\}$$

Debe cumplir que $(i, j, s_1), (i, j, s_2) \in E \Rightarrow s_1 = s_2$; es decir, que una misma casilla no puede tener más de un estado.

Definición 2.3 (Posición bloqueada)**.** Dado un entorno (R, C, X), el conjunto de posiciones bloqueadas se define como:

$$\{(i, j) \mid 0 < i \leq R,\, 0 < j \leq C, \nexists s \in S : (i, j, s) \in E\}$$

Son posiciones válidas en la cuadrícula que no tienen un estado en el entorno.

Figura 4: De izquierda a derecha, de arriba a abajo: Roby, Tor, Eva, una tuerca, un tornillo, una caja, un muro y dos puertas.

Figura 5: Roby después de intentar moverse hacia un muro.

(a) (b)

Figura 6: Roby obteniendo información, en (a) de Tor y en (b) de una caja.

Definición 2.4 (Inventario)**.** Definimos un inventario como una función

$$I : Str \to \mathbb{N}_0$$

donde Str es, informalmente, el conjunto de las cadenas de texto.

Definición 2.5 (Estado del mundo)**.** Definimos el estado del mundo como una tupla (R_r, C_r, I, E) donde la posición (R_r, C_r) no está bloqueada e indica dónde se encuentra el agente en la cuadrícula, I es una función inventario y E es el entorno de la tarea..

Definición 2.6 (Estado del agente)**.** Definimos el estado del agente como el estado interno del programa del usuario.

Definición 2.7 (Función $take$)**.** La función $take$ devuelve un estado del mundo modificado, cambiando el estado de una casilla y actualizando la función inventario:

$$take(r, c, i_1, e_1) = \begin{cases} (r, c, i_2, e_2) & \text{si } \exists\, obj, lab, val \mid (r, c, takeable\, \langle obj, lab, val\rangle) \in X \\ (r, c, i_1, e_1) & \text{en otro caso} \end{cases}$$

donde

$$\begin{aligned} e_1 &= (R, C, X) \\ e_2 &= (R, C, (X - \{(r, c, takeable\, \langle obj, lab, val\rangle)\}) \cup \{(r, c, empty)\}) \\ i_2(x) &= \begin{cases} i_1(x) + val & \text{si } x = obj \\ i_1(x) & \text{en otro caso} \end{cases} \end{aligned}$$

Se trata de una función que cambia una casilla $takeable$ por una $empty$ - decimos que el objeto de la casilla ha sido recogido - y cambia la función de inventario de manera que devuelva la suma de los valores viejo y nuevo asociados a dicho objeto.

Definición 2.8 (Funcioń $move$)**.** La función $move$ devuelve un estado del mundo modificado, cambiando la posición del agente y modificando el entorno y el inventario en caso de que la nueva posición coincida con una casilla $takeable$.

$$move(\Delta r, \Delta c, r_1, c_1, i_1, e_1) = \begin{cases} (r_2, c_2, i_2, e_2) & \text{si } 0 < r_2 < R,\ 0 < c_2 < C,\ \exists\, s \mid (r_2, c_2, s) \in X \\ \bot & \text{en otro caso} \end{cases}$$

donde

$$e_1 = (R, C, X)$$
$$r_2 = r_1 + \Delta r$$
$$c_2 = c_1 + \Delta c$$
$$take(r_2, c_2, i_1, e_1) = (r_2, c_2, i_2, e_2)$$

Nótese que al ser una función que puede diverger, queda implícita la existencia de movimientos inválidos por parte del agente: aquellos que conduzcan a una posición bloqueada o fuera de la cuadrícula.

2.2.2. Acciones y percepción

Roby, el agente software, sólo puede modificar el estado del mundo mediante comandos de movimiento, que se traducen a cambios en su posición y en las casillas y el inventario al recoger objetos. Cada uno de los comandos se corresponde en Javascript con una función, parte de la librería base de ToolboX.Academy.

Usaremos la notación $\langle a, S_1 \rangle \rightarrow S_2$ para expresar que, si en el estado del mundo S_1, el agente lleva a cabo la acción a, entonces el nuevo estado del mundo será S_2. Nótese que los cambios en el estado del agente no son expresados.

Acción	**Transición**	**Función en Javascript**
up	$\langle up, (r, c, i, e) \rangle \rightarrow move(-1, 0, r, c, i, e)$	`up()`
left	$\langle left, (r, c, i, e) \rangle \rightarrow move(0, -1, r, c, i, e)$	`left()`
down	$\langle down, (r, c, i, e) \rangle \rightarrow move(1, 0, r, c, i, e)$	`down()`
right	$\langle right, (r, c, i, e) \rangle \rightarrow move(0, 1, r, c, i, e)$	`right()`

Además de las acciones para modificar el estado del mundo, Roby cuenta con una función de percepción, *info*, con la que potencialmente modificar su estado interno. Esta acción puede llevarse a cabo cuando el agente se encuentra en una casilla con estado $occupied\ \langle obj, lab, val \rangle$ para obtener el valor val. Si se usa en otro caso, la percepción resulta en un valor fijo de -1.

Sea (r, c, i, e) el estado actual de una tarea. Las acciones de percepción que el agente Roby puede llevar a cabo son:

Acción	**Información obtenida**	**Función en Javascript**
info	$\begin{cases} val & \text{si } \exists\, obj \mid (r, c, occupied\ \langle obj, lab, val \rangle) \in e \\ -1 & \text{en otro caso} \end{cases}$	`info()`

2.2.3. Naturaleza del entorno

Dentro las categorías descritas en Russell y Norvig (2010, capítulo 2.3), podemos clasificar el mundo de Roby como un entorno:

- *Parcialmente observable*. La percepción del agente, a través de la acción *info*, se limita a información parcial de la misma casilla en la que se encuentre.
- *Determinista*. Tras cada acción del agente, el siguiente estado del mundo (si lo hay) está bien definido. No hay componente de incertidumbre.
- *Secuencial*. Los cambios que el agente efectúa en el mundo pueden afectar a sus decisiones futuras, y por ello hay una sola ejecución del programa del usuario, que debe resolver el problema planteado. No hay episodios definidos de percepción-actuación.
- *Estático*. El entorno cambia si y solo si el agente actúa.
- *Discreto*. El conjunto de posibles estados de una tarea es finito.
- *Conocido*. Las reglas del entorno y el efecto de las acciones del agente se conocen y no varían en ningún momento.

2.3. Estudio de posibles problemas de IA para implementar

A continuación se enumeran los problemas que se han estudiado para implementarse en potenciales tareas de ToolboX.Academy. Las ilustraciones a continuación sólo son maquetas, porque el planteamiento se hace teniendo en cuenta que cada caso podría requerir de modificaciones en el entorno o en el agente, y la plataforma no está preparada aún para implementar algunas tareas que se proponen de ejemplo (para más detalles, ver 3.2.2).

2.3.1. Búsqueda de caminos

Camino entre dos celdas El agente debe encontrar un camino hasta una casilla objetivo, por ejemplo, para recoger una tuerca, evitando posiciones bloqueadas o marcadas como obstáculos.

Figura 7: Ejemplo de tarea para búsqueda de camino entre dos celdas.

Problema del viajante El agente debe encontrar el camino más corto para recoger varias tuercas.

Figura 8: Ejemplo de tarea sobre el problema del viajante.

2.3.2. Optimización combinatoria

Problema de las monedas Cada objeto en la cuadrícula tiene un valor asociado. Roby debe minimizar el número de objetos que debe recoger para acumular cierto valor, pero sin pasarse de un límite.

Problema de la mochila Cada objeto en la cuadrícula tiene un valor y un peso asociados. Roby debe maximizar el valor de los objetos recogidos, dado un límite de peso.

Figura 9: Ejemplo de tarea sobre optimización combinatoria, donde el enunciado proporcionaría una correspondencia de peso y valor para cada tuerca.

2.3.3. Razonamiento

Satisfacción de restricciones Roby debe modificar la posición de los objetos en la cuadrícula para que sigan una disposición concreta.

Figura 10: Ejemplo de tarea de satisfacción de restricciones, en la que el alumno debe reordenar las tuercas para que se encuentren junto a la puerta de su color.

Lógica Dada una base de conocimiento, Roby debe seleccionar un objeto de entre varios, según las proposiciones lógicas de primer orden que reciba durante la ejecución.

Figura 11: Ejemplo de tarea de lógica, en la que los agentes auxiliares proporcionarían a Roby predicados sobre los que razonar para decidir que objeto recoger.

Planificación Roby debe llegar a una casilla objetivo, pero primero debe desbloquear el camino.

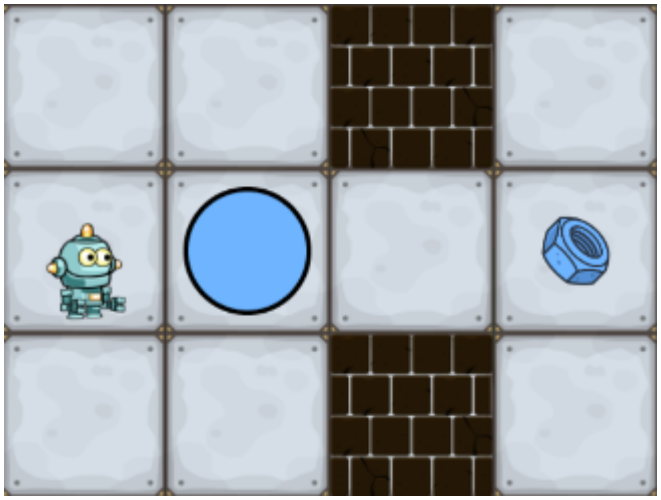


Figura 12: Ejemplo de tarea de planificación, en la que el orden de las acciones del agente es importante para llegar a su objetivo, con un nuevo tipo de objeto que en lugar de ser recogible, es empujado en la dirección de movimiento de Roby cuando éste entra en su casilla.

2.3.4. Aprendizaje automático

Clasificación Roby debe predecir la clase asociada a una o varias cajas según sus características, en base a un conjunto de datos de entrenamiento que cargue de un agente auxiliar.

Regresión Roby debe predecir el valor contenido en una o varias cajas según sus características, en base a un conjunto de datos de entrenamiento que cargue de un agente auxiliar.

Figura 13: Ejemplo de tarea de regresión, en la que Tor proporciona los datos de entrenamiento y Roby debe predecir el valor Y contenido en cada caja para cada valor X de las etiqueta correspondiente

2.4. Elección de problemas

Los problemas elegidos para implementarse son los de **búsqueda de caminos**. También se ha trabajado en los de optimización combinatoria, pero éstos se han excluido de esta sección por estar inconclusos (más detalles en 3.1).

2.4.1. Justificación

Los motivos de esta elección son los siguientes:

- Los algoritmos de búsqueda en grafos que se usan para solucionarlos son fáciles de ilustrar en el entorno. El resto de problemas no tiene el mismo componente espacial y los procesos de decisión subyacentes son más difíciles de representar visualmente.

  Esto es especialmente importante para introducir al alumno a distintos conceptos básicos de algoritmia como *estrategia de fuerza bruta*, *estrategia voraz* o *heurístico* de una manera más intuitiva.

- La búsqueda de caminos proporcionará al alumno funciones para programar agentes que abstraigan la navegación en la cuadrícula. Con ello será más fácil que el alumno se

centre en problemas más complejos sin preocuparse por los movimientos concretos del agente.

- La única modificación previa, necesaria para la implementación de los algoritmos, es añadir al agente nuevas acciones de percepción, puesto que la búsqueda de caminos requiere de un entorno totalmente observable. En un entorno parcialmente observable, sería necesario como mínimo un proceso previo de exploración para construir el mapa sobre el que realizar la búsqueda. De hecho, como la función actual de percepción está limitada a obtener información de la casilla en la que se encuentra el agente, ni siquiera es posible llevar a cabo dicha exploración, al no poder reconocer las posiciones boqueadas sin realizar movimientos no válidos.

  Otras tareas requerirían de modificaciones más significativas en las leyes del entorno o el funcionamiento de la plataforma.

2.4.2. Planteamiento formal

Búsqueda en grafos Hará falta una representación de la cuadrícula en estructura de grafo. Sobre éste aplicaremos búsqueda en anchura como ejemplo de fuerza bruta, el algoritmo de Dijkstra como alternativa voraz y A* como búsqueda informada por un heurístico.

Definición 2.9 (Grafo de la cuadrícula)**.** Dado el estado de una tarea (ver definición 2.2) $(R_r, C_r, I, (R, C, X))$, podemos definir el grafo de la cuadrícula como (N, E), donde N es el conjunto de nodos o vértices y E es el de aristas.

$$N = X \cup \{(R_r, R_c, occupied\,\langle Roby, \epsilon, -1\rangle)\}$$

$$E = \{(a, b) \mid \exists\, m \in M : \langle m, (r_a, c_a, i_1, e_1) \to (r_b, c_b, i_2, e_2)\rangle\}$$

donde M es el conjunto de acciones de movimiento del agente $M = \{up, left, down, right\}$ y a y b son nodos en el grafo: $a, b \in N, a = (r_a, c_a, s_a),\ b = (r_b, c_b, s_b)$.

En otras palabras, para el estado de una tarea, el grafo de la cuadrícula contiene la información de las casillas y el agente en los nodos, y las aristas conectan las posiciones que distan una acción de movimiento entre sí (ver figura 14).

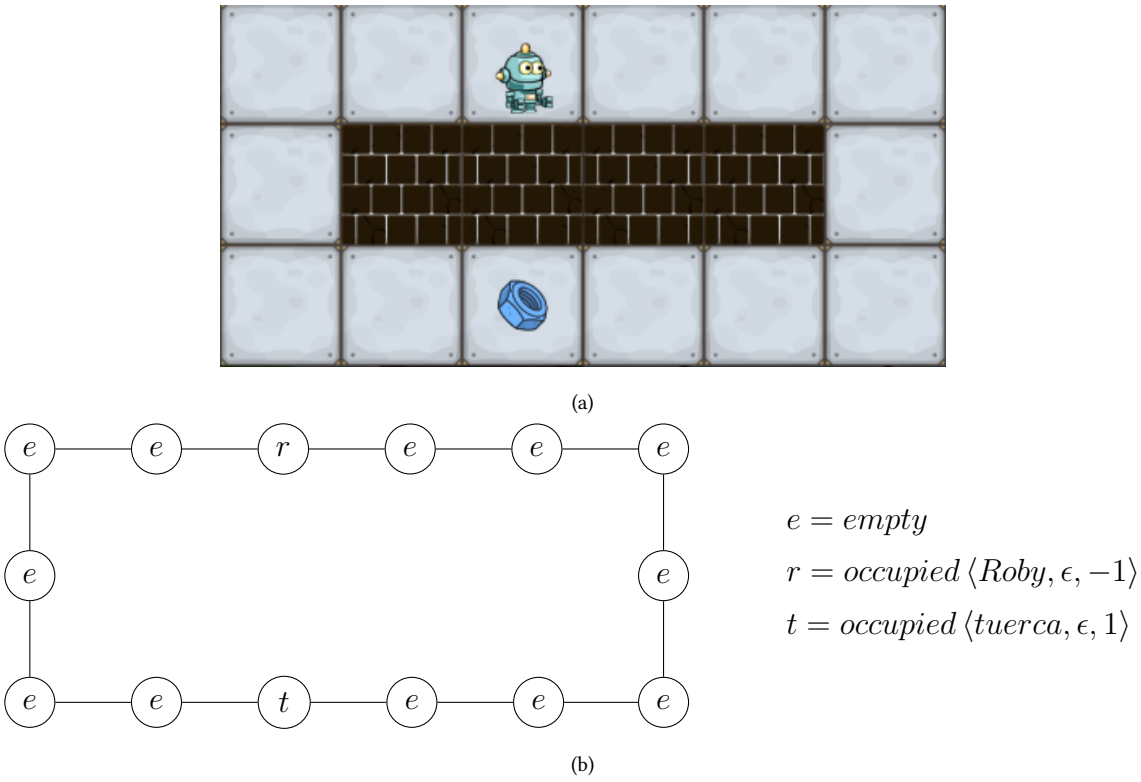


Figura 14: Ejemplo de la representación visual (a) y en estructura de grafo (b) de la cuadrícula.

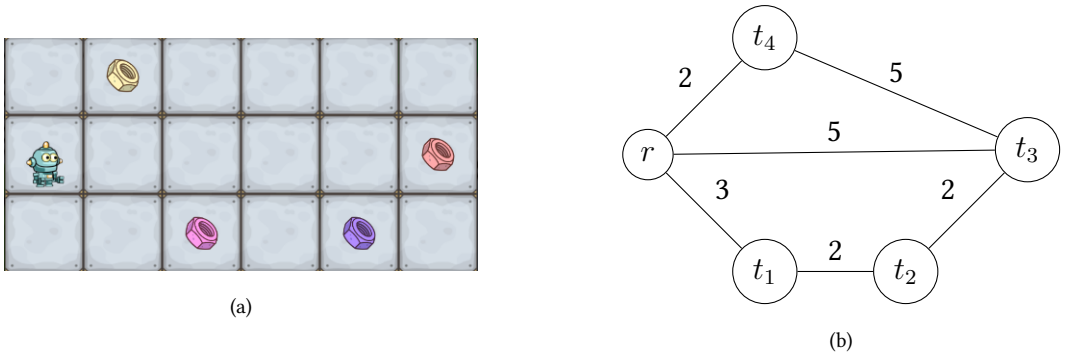
Definición 2.10 (Función *searchpath*)**.** Dado el grafo de una cuadrícula, casilla inicial y una objetivo, la función *searchpath* es aquella que devuelve el vector de nodos que conforman el camino más corto entre las dos casillas.

$$searchPath((N, E), n_a, n_b) = P$$

$$n_a, n_b \in N, P \in N^*$$

Problema del viajante Una vez que la navegación en el tablero ha sido abstraída gracias a la búsqueda de caminos, es posible plantear el problema de encontrar la ruta óptima que pase por ciertos puntos clave. Para buscarle solución, necesitamos una nueva representación en grafo de los objetos que queremos recoger. Sobre éste, aplicaremos búsqueda por fuerza bruta, una solución voraz y un algoritmo evolutivo.

Definición 2.11 (Mapa de carreteras)**.** Dado el grafo de una cuadrícula (N, E), definimos un mapa de carreteras como un una tupla (C, R, W) en el que $C \in N$ es el conjunto de ciudades o vértices del grafo, R es el conjunto de carreteras o aristas y W es la función de peso, qué



(a) (b)

Figura 15: Ejemplo de la representación del mapa de carreteras donde las ciudades son las casillas ocupadas.

asigna a cada carretera un peso correspondiente a la longitud del camino mínimo entre las ciudades que conecta.

$$W(e) = |\, searchpath((N, E), a, b)\,|$$

donde $a, b \in C, e = (a, b) \in R$

En nuestra definición de mapa de carreteras, vamos a excluir carreteras redundantes, imponiendo la siguiente restricción: si existen dos caminos distintos de la misma longitud entre dos ciudades y uno pasa por una tercera ciudad, entonces el conjunto de carreteras no incluye aquella directa entre las dos primeras ciudades. Como ejemplo, en la figura 15, no hay una carretera directa entre t_1 y t_3 porque una de la misma longitud existe pasando por t_2.

Definición 2.12 (Función *salesman*)**.** Llamaremos función *salesman* a aquella que, dado un mapa de carreteras, devuelve un vector con las ciudades en el orden en que deben visitarse para realizar un recorrido óptimo.

$$salesman(C, R, W) = P$$

donde $P \in C^{|C|}$

Nuevas funciones de percepción El agente va a necesitar: 1) una acción de percepción para encontrar la posición de un objeto único en la cuadrícula, 2) una acción de percepción para reconocer el grafo de la cuadrícula en el estado actual de la tarea y 3) una acción de percepción para construir un mapa de carreteras en el estado actual de la tarea. Las estructuras que estas acciones le permitirán almacenar en su estado interno serán necesarias para poder aplicar sobre ellas los algoritmos que solucionen los problemas planteados.

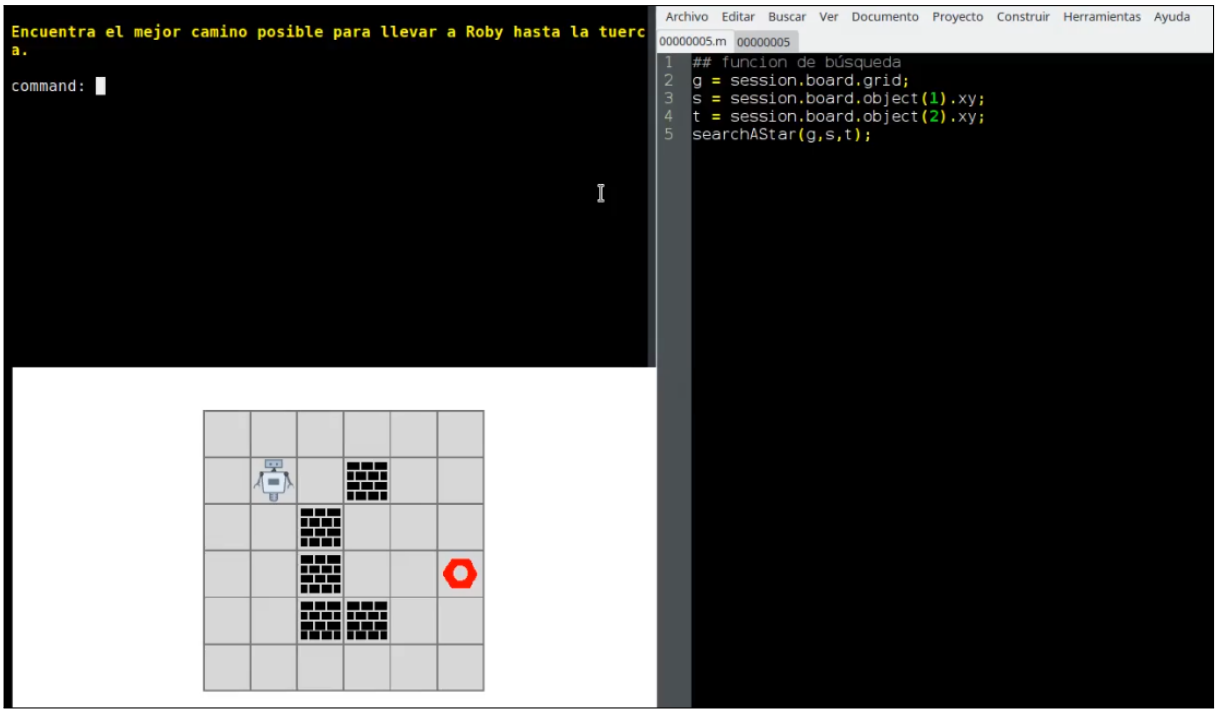
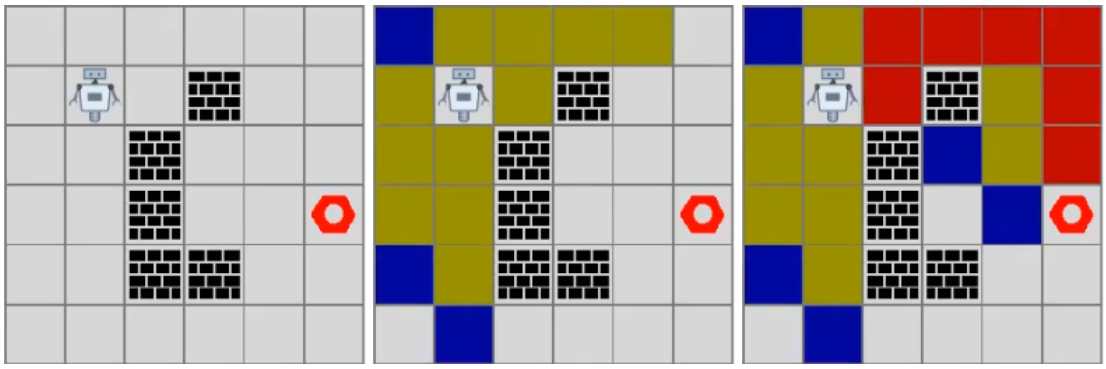


Figura 16: Interfaz de ToolboX, predecesor de ToolboX.Academy.

2.5. Prototipo

Al comienzo de este trabajo, el propio ToolboX.Academy se encontraba en una etapa incipiente con el nombre de ToolboX, implementado en GNU Octave, y la primera versión del módulo de IA se hizo en este prototipo. La única función que se implementó fue `searchAStar`. El entorno era totalmente observable, puesto que ToyScript aún no había sido definido y todo el entorno era observable para el agente a través del estado del programa de Octave. En la figura 16 se muestra el aspecto original de ToolboX y en la figura 17, capturas del proceso ilustrado de búsqueda de caminos mediante el algoritmo A*.

Al comienzo de ToolboX.Academy, el código del nuevo módulo de IA mantuvo una estructura monolítica, igual que en el prototipo, hasta que se implementó un sistema de carga de librerías. Entonces, el código se separó en diferentes módulos, que se explicarán a continuación.



(a) (b) (c)

Figura 17: Proceso de búsqueda de caminos. En azul, casillas correspondientes a nodos abiertos. En amarillo, nodos cerrados. En rojo, nodos parte de la solución. En (a) se muestra el estado inicial, en (b) cómo se extiende el árbol de búsqueda y en (c) cómo se colorea el camino encontrado.

2.6. Implementación

2.6.1. Módulos

`Graph.js` : Implementación de una clase grafo genérica y los algoritmos de búsqueda.

`Board.js` : Funciones relacionadas con el tablero: colorear casillas, pintar conexiones, obtener posiciones de casillas que se ajusten a cierto criterio, funciones de distancia entre casillas...

`Evolutive.js` : Implementación genérica de un algoritmo evolutivo.

`Pathfinding.js` : Funciones para construir el grafo de la cuadrícula, mapas de carreteras, y aplicar búsqueda de caminos o solucionar el problema del viajante en el mundo de Roby.

2.6.2. Funciones

Métodos para renderizar las soluciones El principal criterio didáctico en ToolboX.Academy es que las soluciones deben ser lo más visualmente intuitivas posible. ToolboX.Academy no contaba con una función para dibujar nada en el tablero, así que fue necesario programar una función `Board.drawConnection` que renderizase una línea entre dos casillas y un nodo en el centro de cada una. Los colores de cada nodo y el de la conexión se pasan como parámetros de la función. En la figura 18 se muestra el resultado de llamar a esta función sobre dos casillas contiguas.

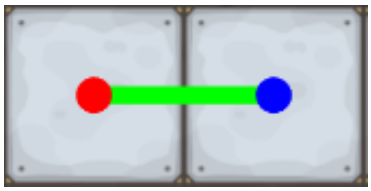


Figura 18: Conexión dibujada con nodo origen en rojo, arista en verde y nodo destino en azul.

A partir de esta función se construyen las siguientes:

`Pathfinding.paintPath` : Dibuja en orden las conexiones que definen un camino.

`Pathfinding.paintSearch` : Dibuja la evolución del árbol de búsqueda a lo largo del proceso.

`Pathfinding.paintRoads` : Dibuja los caminos correspondientes a las carreteras del mapa de carreteras, empezando por las que parten de la posición de Roby.

Nuevas funciones de percepción Como se ha dicho previamente, el agente necesitaría nuevas acciones de percepción para hacer el entorno totalmente observable. Las funciones que satisfacen esta necesidad son:

`Board.filterCells` : Devuelve una lista con la información de las casillas que contienen cierto objeto, valor o etiqueta.

`Board.findCell` : Atajo para obtener el primer elemento de la lista devuelta por `Board.filterCells`. Implementa la primera función de percepción especificada en 2.4.2.

`Pathfinding.makeBoardGraph` : Inicializa una representación interna del grafo de la cuadrícula para el estado actual. Implementa la segunda función de percepción especificada en 2.4.2.

`Pathfinding.RoadMap` : Constructor de un objeto con la información de un mapa de carreteras, para ciertas ciudades. Implementa la tercera función de percepción especificada en 2.4.2.

Algoritmos para las soluciones

`Pathfinding.searchPath` : Calcula un camino entre dos posiciones. Pueden especificarse, como parámetros opcionales, qué casillas se consideran obstáculos para que Roby las evite y qué estrategia de búsqueda debe utilizarse - por anchura, por Dijkstra o A*, que utiliza la distancia manhattan como heurístico. Implementa la función *searchpath* (ver definición 2.10).

`Pathfinding.RoadMap.salesmanBrute` : Calcula el camino óptimo probando todas las permutaciones de las ciudades del mapa. Implementa la función *salesman* (ver definición 2.12).

`Pathfinding.RoadMap.salesmanGreedy` : Calcula un camino no necesariamente óptimo dirigiéndose siempre a la ciudad más cercana. El único motivo por el que no consideramos que implemente *salesman* (ver definición 2.12) es porque no devuelve necesariamente una solución óptima.

`Pathfinding.RoadMap.salesmanEvolutive` : Calcula un camino no necesariamente óptimo aplicando un proceso de evolución sobre una generación inicial aleatoria. Al igual que en el caso anterior, no consideramos que implemente *salesman* (ver definición 2.12) porque puede no devolver una solución óptima.

2.6.3. Tareas

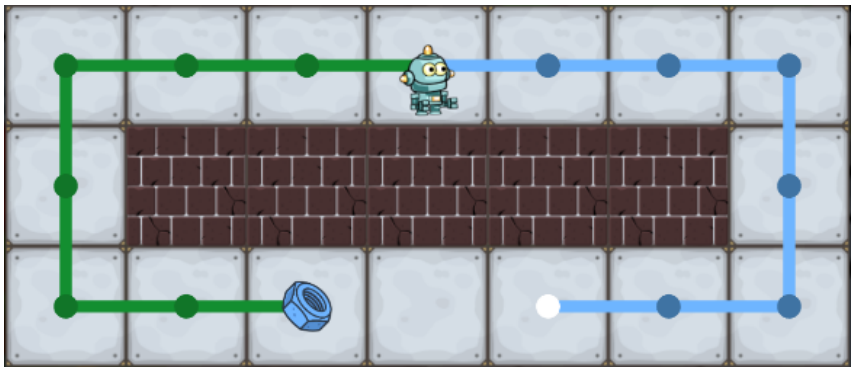
Podemos agrupar las tareas de la versión demo del módulo por los conceptos que ilustran:

La misma solución resuelve varios problemas Los módulos de los cursos existentes enseñan soluciones únicas para una sola configuración de tarea. En cambio, con los algoritmos de IA, el mismo código puede resolver muchas tareas con distintas configuraciones (ver fig. 19).

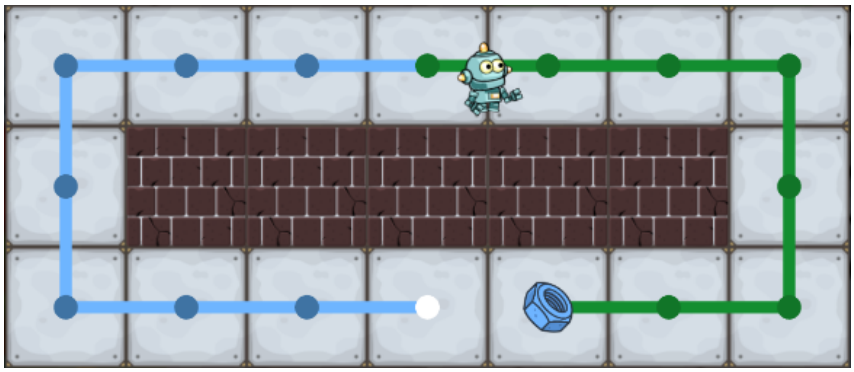
```
import Board
import Pathfinding

// Define starting point and goal
```

```
start = Board.findCell('Roby')
goal = Board.findCell('tuerca')

// Perform the search
result = Pathfinding.searchPath(start, goal)

// Display the process
Pathfinding.paintSearch(result)
Pathfinding.paintPath(result, 'successful')

// Make Roby follow the path
Pathfinding.runPath(result.path)
```

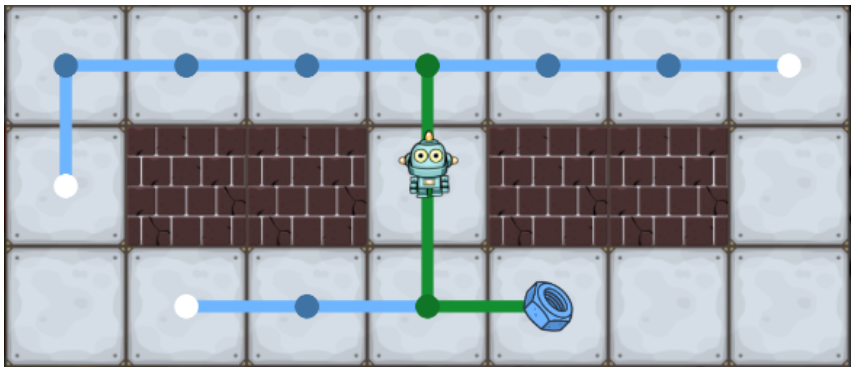
Calidad de la solución y coste de ejecución El segundo concepto que se enseña en el módulo es que existen distintos enfoques y que cada uno tiene sus ventajas y desventajas. Para ilustrarlo, se hace una comparación entre los resultados de una solución por fuerza bruta y una voraz en un problema del viajante. Se presentan dos tareas con exactamente la misma configuración, resueltas con sendos enfoques. Al finalizar se imprime en la pestaña de mensajes la longitud del camino recorrido y el número de iteraciones en el proceso de decisión, quedando claro que el voraz es mucho más rápido pero no da necesariamente la mejor solución, mientras que el de fuerza bruta es muy lento pero encuentra la solución óptima (ver fig. 20).



(a)

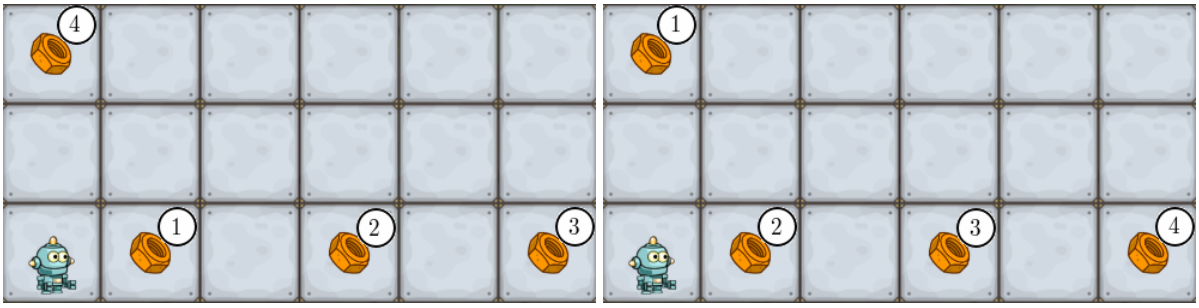


(b)



(c)

Figura 19: Tres configuraciones distintas resueltas por el mismo código.



(a) (b)

Figura 20: Comparación de resultados entre enfoque voraz (a), que efectúa 5 iteraciones en la búsqueda y consigue un camino de longitud 13, y el enfoque de fuerza bruta (b), que necesita 32 iteraciones pero consigue un camino de longitud 10.

3

Conclusiones y Líneas Futuras

3.1. Resultados excluidos

La experiencia de este trabajo deja claro que el verdadero reto no está en la parte informática, puesto que escribir una librería o incluso integrar una existente no presenta una dificultad considerable. El verdadero desafío está en el diseño didáctico; en ilustrar los procesos subyacentes de manera visual e intuitiva. Un buen ejemplo de cómo esta dificultad obstaculiza la integración de la IA en los módulos de aprendizaje son los algoritmos que se incluyeron en las librerías para resolver el problema de la mochila. A pesar de haberse programado y probado en tareas reales, éstas no se han incluido en ninguna demostración porque no se ha terminado de decidir la mejor manera de comunicarle visualmente al estudiante qué sucede durante la ejecución del código.

El resultado que no se incluyó es el módulo `Combinatorial.js`, que implementa varias soluciones al problema de la moneda y el problema de la mochila; una versión de fuerza bruta, una voraz y una genética, parametrizando el algoritmo evolutivo programado en `Evolutive.js` (ver 2.6.1). Parte del resultado incluye una función de renderizado para cambiar el color de fondo de una casilla, que se utilizó para intentar ilustrar el proceso por el que Roby selecciona los elementos de la solución. Sin embargo, al ejecutarlo (ver fig. 21), no quedaba expresadas las diferencias entre cada estrategia ni los criterios de selección de los elementos.

3.2. Extensión del módulo de IA

3.2.1. Mejora de los resultados existentes

A continuación se describen algunas maneras de mejorar o completar el trabajo realizado.

(a) (b) (c)

Figura 21: Ejemplo de ejecución de la tarea sobre el problema de la mochila, excluida del módulo final. Las casillas resaltadas indican qué elementos están incluidos en la solución.

- Es necesario un esfuerzo para diseñar más y mejores métodos de visualización de algoritmos. Ciertas cosas que podrían visualizarse son: 1) los costes de los caminos en la búsqueda por Dijkstra, 2) diferentes heurísticos en la búsqueda por A* y 3) el proceso de selección de ciudades en el problema del viajante y no solo el camino final. También hace falta decidir una representación didáctica para las soluciones de los problemas de combinatoria, para que puedan incluirse en el módulo de enseñanza.

- Añadir más tareas al módulo. La versión existente es la mínima para demostrar el potencial didáctico de la plataforma, pero todavía no ofrece un recorrido adaptado a los usuarios a los que está destinado. Son necesarias más tareas intermedias que expliquen de manera gradual cada concepto implicado y que reiteren sobre los más importantes.

3.2.2. Nuevos algoritmos

De la misma manera que se justificó en 2.4.1 la necesidad de extender la capacidad de percepción del agente para implementar la búsqueda de caminos, hace falta definir las exigencias formales de cada problema de interés. Una vez hecho, habrá que traducir cada solución al lenguaje visual de la plataforma, como se hizo con los métodos de renderizado descritos en 2.6.2. Concluimos que la inclusión de cada nuevo algoritmo tendrá dos fases: 1) estudiar sus requisitos formales y modificar la plataforma para satisfacerlos, y 2) decidir cómo extender las utilidades gráficas para representarlo visualmente.

Para varios problemas de IA propuestos en el análisis hecho en 2.3, podemos prever algunas

dificultades potenciales.

Razonamiento (2.3.3) El reto principal de esta categoría compromete la misma filosofía de enseñanza de la plataforma. De mantener el mismo paradigma de programación, el enfoque de estos problemas, requeriría escribir librerías para trabajar con predicados de lógica de primer orden, satisfacción de restricciones, sistemas de verificación, etc. El trabajo necesario no sería pequeño. La alternativa sería utilizar un motor de inferencias existente, como *Tau Prolog* [10]. Sin embargo, esto significaría dar un giro para incluir la enseñanza de la programación lógica. Por una parte, esta decisión tendría más sentido, teniendo en cuenta que este paradigma existe precisamente para abstraer los algoritmos de razonamiento automático y centrarse en el funcionamiento y la definición de las bases de conocimiento. Por otra parte, un cambio de este calibre no debería tomarse a la ligera y exige justificar su integración con sentido, tanto en las librerías de la plataforma como en el módulo de enseñanza. Una posible solución para este dilema es la construcción de un sistema híbrido, que ofrezca una interfaz de programación imperativa del estilo propio de la plataforma, pero que por debajo funcione con el motor de inferencias propuesto.

Aún habiendo resuelto esta cuestión, la representación visual del uso de una base de conocimiento o de un proceso de satisfacción de restricciones, no es algo que sea fácil diseñar. Aún más, si llegara a realizarse la integración con un motor lógico de terceros, podría ser necesario realizar toda una interfaz entre el mismo y los nuevos métodos de representación visual.

Aprendizaje automático (2.3.4) Los algoritmos basados en el entrenamiento sobre conjuntos de datos, como el perceptrón o los árboles de decisión, requieren de bases de datos con cierto tamaño para justificar su uso. Si el sistema de entrada de datos se mantiene tal y como existe en el mundo de Roby, a través de la acción *info*, cada entrada del conjunto de datos equivaldría a una o varias casillas ocupadas en la cuadrícula. Teóricamente, esto no supone un problema para la plataforma, pero el proceso de adquisición de los datos por parte de Roby sería demasiado lento y tedioso, y el tamaño de la cuadrícula podría empeorar la visualización de la aplicación en pantallas de poco tamaño. Una alternativa es introducir el conjunto de datos con una sola acción *info*, aunque esto haría opaca para el alumno la naturaleza de los datos

[10] *Tau Prolog* es un intérprete en JavaScript de Prolog, un lenguaje de programación lógica. http://tau-prolog.org

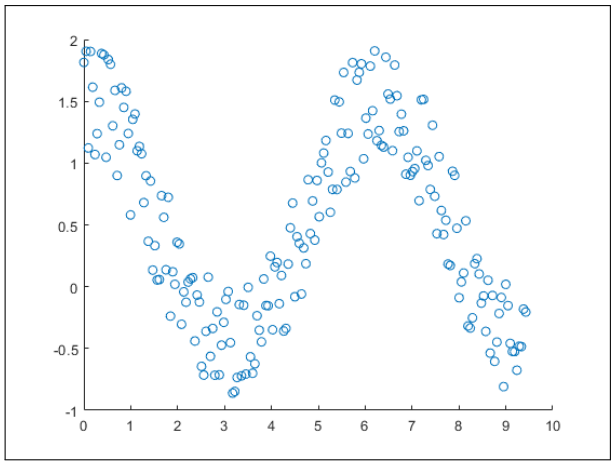


Figura 22: Ejemplo de una gráfica renderizada por una función de Matlab. Fuente: https://www.mathworks.com/help/matlab/ref/scatter.html.

introducidos, si no se diseñase una solución para visualizarlos. En este respecto, la plataforma podría beneficiarse de una nueva pestaña en su interfaz, similar a la de *Enunciado* y *Mensajes*, que sirviera de lienzo para la visualización de conjuntos de datos y gráficas, de manera similar a las ventanas gráficas que se utilizan en Matlab y GNU/Octave (ver fig. 22).

3.2.3. Experimentación sobre el entorno

Un enfoque alternativo para proponer problemas de IA sería modificar las propiedades el entorno (descritos en 2.2.3) para explorar las posibilidades que ofrece. A continuación se proponen algunas.

Problemas de navegación Necesitan un entorno observable sólo en un rango de distancia. Si las casillas lejanas al agente no fueran visibles para el alumno y las funciones de percepción estuvieran limitadas a obtener información de las casillas en ese rango, podría plantearse la tarea de implementar un *crawler*; es decir, de programar al agente para que explorase el entorno evitando obstáculos hasta llegar a un objetivo definido.

IA para juegos Serían viables en un entorno episódico y dinámico o multi-agente. En lugar de programar una secuencia que se ejecuta una sola vez por tarea, el agente podría ejecutar el mismo código, con una estructura de percepción-actuación, hasta que el entorno cumpliese

cierta condición - un funcionamiento similar al de *Robocode* [11]. Esto estaría justificado si el entorno se modificase independientemente de las acciones del agente. Tener un agente contrincante o elementos que actúen por cuenta propia daría pie a tareas que requiriesen implementar algoritmos de toma de decisiones como *minimax*.

Lógica difusa y razonamiento bayesiano Necesitan un entorno estocástico. Si las reglas del mundo tuvieran asociadas probabilidades y el resultado de una tarea no fuera un valor binario (solución correcta o incorrecta), sería posible plantear tareas que precisaran implementar operadores sobre conjuntos difusos. El resultado de este tipo de tarea, además, tendría que evaluarse también de manera difusa y definir un umbral para considerarla superada.

3.3. Conclusión

En el marco que ofrece ToolboX.Academy, se ha obtenido una versión básica de lo que sería un módulo de enseñanza en IA, que si bien sólo contiene una pequeña porción de lo que se puede enseñar, sirve de referencia para versiones futuras. Para la plataforma, aporta el estudio formal realizado sobre la ella y un primer paso en el desarrollo de código y contenidos; fuera de la plataforma, ofrece la experiencia de adaptación de la tecnología a unos criterios didácticos, y un análisis sobre los requisitos particulares de distintos tipos de algoritmos de IA para ser enseñados en un entorno dirigido a estudiantes en etapas preuniversitarias.

El reto a superar para llevar a cabo la alfabetización en IA no es tecnológico, sino comunicativo; se trata de una labor que debe realizarse de manera conjunta entre expertos en IA y profesionales de la educación. Por eso, la continuación de este trabajo no será sólo extender el módulo de enseñanza de IA en ToolboX.Academy, sino aplicar la retroalimentación de los alumnos y los docentes que le den uso en casos reales.

[11] *Robocode* es un juego donde se programa el comportamiento de un tanque para ganar un combate con otros tanques programados. https://robocode.sourceforge.io/

Referencias

Dignum, V. (2018). Ethics in artificial intelligence: introduction to the special issue. https://doi.org/10.1007/s10676-018-9450-z

Hofstadter, D. R. (1979). *Godel, Escher, Bach: An Eternal Golden Braid*. Basic Books, Inc.

INTEF. (2021a). *Escuela de Pensamiento computacional e Inteligencia Artificial - INTEF*. https://intef.es/tecnologia-educativa/pensamiento-computacional/

INTEF. (2021b). *Quiénes somos - INTEF*. https://intef.es/quienes-somos/

Launchbury, J. (2017). *A DARPA Perspective on Artificial Intelligence*. YouTube. https://www.youtube.com/watch?v=-O01G3tSYpU

Law, N., Woo, D., de la Torre, J. & Wong, G. (2018). A global framework of reference on digital literacy skills for indicator 4.4. 2. *UNESCO Institute for Statistics*. http://hdl.voced.edu.au/10707/548017

Llorens Largo, F., García-Peñalvo, F. J., Molero Prieto, X., Vendrell Vidal, E. y col. (2017). La enseñanza de la informática, la programación y el pensamiento computacional en los estudios preuniversitarios. https://doi.org/10.14201/eks2017182717

Long, D. & Magerko, B. (2020). What is AI literacy? Competencies and design considerations. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1-16. https://doi.org/10.1145/3313831.3376727

Ottestad, G. & Gudmundsdottir, G. B. (2018). Information and communication technology policy in primary and secondary education in Europe. *Second handbook of information technology in primary and secondary education*, 1-21. https://doi.org/10.1007/978-3-319-53803-7_92-1

Russell, S. & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach. Third Edit* (Third Edit). Prentice Hall.

Vico, F. (2020). *ToyScript: A computer language for teaching coding in the K-12 system* (Technical report). Universidad de Málaga.

Vico, F., Masa, J. & Garcia, R. (2019). ToolboX. Academy: Coding & Artificial Intelligence made easy for kids, Big Data for educators. *Proceedings of the 11th annual International*

Conference on Education and New Learning Technologies (Edulearn19). http://lib.uib.kz/edulearn19/files/papers/1279.pdf

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga