



Universidad de Murcia

Facultad de Informática

Grado en Ingeniería en Informática

Trabajo Fin de Grado

Integración de un modulo de simulación de robot en el proyecto Descubre

Autor

Alberto López Sánchez

Dirigido por

Juan Antonio Sánchez Laguna

Febrero 2016

Resumen

En la última década se lleva experimentando un aumento en la aceptación y uso de la informática y tecnología en la sociedad. Servicios como Google, Twitter o Facebook son usados en masa por un gran número de personas. Compañías como Apple o Microsoft son conocidas por la mayoría de los integrantes de la sociedad actual. La informática está presente en cada aspecto de la vida de las personas. Desde investigaciones en el cuidado de la salud, pasando por las relaciones sociales que se establecen y hasta la simple acción de realizar la compra están apoyadas por la informática. La informática tiene un gran papel en nuestra sociedad que cada vez cobra más protagonismo.

Pero la informática y sus bases siempre han sido algo desconocido para la gran mayoría de personas. Es común encontrar gente que a pesar de usar diariamente en su trabajo un ordenador, desconoce su funcionamiento. La informática se creó como una herramienta para resolver problemas. Pero también es importante que se entienda su funcionamiento, para poder resolver cada vez problemas más complejos.

Desde los primeros años de la informática se ha buscado extender este conocimiento entre los miembros de la sociedad. Especialmente entre los más pequeños, que formarán las nuevas generaciones de informáticos, pero también de médicos, ingenieros y profesores.

Es imprescindible que se extienda el *pensamiento computacional* entre los miembros más jóvenes de la sociedad. Se ha demostrado en varios estudios que aprender programación en edades tempranas mejora la capacidad de concentración y abstracción, se consigue mayor autonomía y un notable aumento en la creatividad.

Proyectos como Robomind o Scratch buscan justo esto, entrenar el *pensamiento computacional* entre alumnos de educación Primaria y Secundaria. También existen proyectos como Moway o Lego Mindstorm, que utilizan la robótica para atraer a los más pequeños a aprender jugando.

Descubre la programación es una plataforma online cuyo objetivo es enseñar a programar a los alumnos de Educación primaria y Secundaria de la Región de Murcia. Descubre proporcionando un editor online, retos y una red social para compartir los programas que se crean y aprender de los compañeros. También tiene la finalidad de que se use la plataforma para extender la programación en los colegios,

proporcionando diferentes herramientas para seguir el progreso del alumno.

En este proyecto se intenta aglutinar todas las ideas en una herramienta para aprender a programar. Se ha creado un simulador de un robot que se integrará en la plataforma Descubre la programación.

De esta manera, se une la idea de enseñar a programar en un entorno online a la de usar la robótica como punto de captación y entretenimiento. Aprender a programar jugando.

, con acceso de cualquiera, desde cualquier

Extended Abstract

Índice general

Resumen	III
Extended Abstract	V
Índice	VIII
Lista de figuras	X
Lista de tablas	XI
1. Introducción	1
1.1. Situación actual	2
1.2. La robótica como herramienta de aprendizaje	3
1.3. Proyecto Descubre la programación	4
1.4. Motivación y enfoque de este proyecto	5
2. Estado del arte	9
2.1. Logo y el robot Turtle	9
2.1.1. Turtle de Logo	10
2.2. Scratch	11
2.3. Aprendiendo con robots	12
2.3.1. Lego Mindstorm NXT/EV3	12
2.3.2. Moway	13
2.3.3. Karel the Robot	14
2.3.4. Robomind	15
2.4. Aprendizaje Online	16
2.4.1. CodeHS	17
2.4.2. Code.org	17
3. Análisis de objetivos	19
3.1. Tecnologías usadas	20
3.2. Herramientas utilizadas	21

4. Diseño y resolución del trabajo realizado	23
4.1. Modelo de programación del simulador	24
4.1.1. Modificación del lenguaje iJava	27
4.2. Integración en Descubre del simulador	28
4.2.1. Modificación de la plataforma Descubre	30
4.3. Motor físico	31
4.3.1. Definición del Mundo	33
4.3.2. Construcción del robot	35
4.3.3. Moviendo a Robode	41
4.3.4. Detectando colisiones	42
4.3.5. Detectando lineas	44
4.3.6. Animando a Robode	48
5. Conclusiones y vías futuras	53
A. Edad escolar en diferentes Sistemas Educativos	57
B. Lenguaje Logo	59
C. Programando a Turtle de Logo	63
D. Programación con Robomind Academy	65
E. Programando con Scratch	67
F. Modificación de iJava para el simulador	69
G. Creación de elementos en Box2D	75
H. Construcción de circuitos	79
Bibliografía	85

Índice de figuras

1.1.	Mapa de visualización de eventos de llevados a cabo por el proyecto <i>Hora del código</i> alrededor del mundo. Actualmente 198,473 en todo el mundo, 1,839 en España. Obtenido de [14].	3
1.2.	Sección <i>Crea</i> del proyecto Descubre la programación. Obtenido de [28].	6
1.3.	Salida del programa escrito en iJava ejecutado por el código 1.2.	6
2.1.	Gráfico de edad de los usuarios de Scratch. Obtenido de https://scratch.mit.edu/statistics/	11
2.2.	Entorno de programación en Robomind Academy. Obtenido de [47]. . .	16
4.1.	Diagrama de clases de la plataforma Descubre: punto de partida.	29
4.2.	Diagrama de comunicación de la función <code>run()</code>	30
4.3.	Diagrama de comunicación que se produce cuando se ejecuta un código Java después de integral el simulador con Descubre.	31
4.4.	Diagrama de clases de la aplicación después de integrar el simulador con Descubre.	32
4.5.	Representación de lo que ocurre cada vez que se realiza la llamada a la función <code>Step()</code> en Box2D. Obtenido de [9].	35
4.6.	Esqueleto del robot que muestra los diferentes <i>bodies</i> y <i>joints</i> que lo forman.	36
4.7.	Ejemplo de curva de Catmull-Rom con 4 puntos que definen su forma. Obtenido de [20].	45
4.8.	Ejemplo de curvas de Bezier cuadrática. Obtenido de http://pomax.github.io/bezierjs	45
4.9.	Ejemplo de curvas de Bezier cúbica. Obtenido de http://pomax.github.io/bezierjs	45

- A.1. Tabla que muestra los típicos patrones de progresión en el Sistema Educativo Americano. Fuente: U.S. Department of Education, National Center for Education Statistics, Annual Reports Program. Obtenido de http://nces.ed.gov/programs/digest/d11/figures/fig_01.asp 58
- E.1. Ejemplo de la interfaz de usuario y el editor de código que ofrece Scratch. 68

Índice de cuadros

2.1. Tabla que muestra los resultados obtenidos en el ITBS por los alumnos que aprendieron a programar (denominado <i>computer</i>) y el grupo de control. Obtenido de [23, p.251].	10
A.1. Comparativa de edades de escolarización en diferentes Sistemas Educativos con respecto a las etapas del Sistema Educativo Español.	57
B.1. Resumen de las operaciones elementales que ofrece el lenguaje Logo.	60
C.1. Resumen de órdenes en Turtle usando el lenguaje Logo.	64

Capítulo 1

Introducción

Como dice M. Lemaire[30], en una sociedad que está incrementando su dependencia a las nuevas tecnologías¹, es imprescindible que las nuevas generaciones desarrollen la habilidad de pensar de manera crítica sobre tecnología.

El *Computational Thinking* (pensamiento computacional)[57], es el arte de pensar como un ordenador. El pensamiento computacional, como se refiere A. Bundy en [11], afecta a investigaciones de casi todas las disciplinas, tanto de ciencias como de humanidades. [...] La informática no solo ha permitido que los investigadores puedan hacer nuevas preguntas, sino también ha permitido aceptar nuevos tipos de respuesta. Por ejemplo, preguntas que requieren el procesamiento de una gran cantidad de datos.

Es necesario comprender la informática y desarrollar el *pensamiento computacional*. Se necesitan desarrollar nuevas tecnologías, nuevo Hardware y Software que automatice tareas largas, complejas y con una alta cantidad de cómputo. Tareas que no siempre los humanos podemos resolver de manera directa. Para conseguir esto, muchas veces es necesario programar.

Pero aprender a programar es mencionado como uno de los 7 grandes retos de la educación informática [36] y diversos estudios [44] muestran que las principales dificultades para un alumno cuando está en el proceso de aprender a programar son (a) como empezar un programa; (b) comprensión de la sintaxis específica del lenguaje de programación; (c) comprensión de la lógica² y (d) problemas a la hora de depurar el código escrito.

Aunque a primera vista aprender a programar pueda parecer una tarea ardua y compleja, tiene sus ventajas. En el estudio realizado por J. Siegmund y otros en [52], podemos ver como los participantes mientras comprendían, analizaban y buscaban errores en pequeños trozos de código, daban claras muestras de estar desarrollando

¹Aquí, con nuevas tecnologías me refiero a productos y proyectos derivados de la Informática y las Tecnologías de la Información y la Comunicaciones (TIC) como puede ser los proyectos que detallamos en la sección 2.

²En este caso me refiero a lógica booleana, o también llamada Álgebra de Boole.

actividad cerebral en regiones del cerebro relacionadas con el procesado del lenguaje, la atención y la memoria de trabajo.

De la misma manera, estudios realizados en niños [13] de entre 6 y 8 años, muestran que estos demostraron mayor capacidad de atención, más autonomía y un mayor placer por el descubrimiento de nuevos conceptos. En la misma linea, un estudio en niños de infantil [19] que utilizaban el Lenguaje Logo demostró que los mismos obtuvieron mejores resultados en pruebas de razonamiento, matemáticas o resolución de problemas. Otro estudio más reciente [31] demuestra que aprender a programar (independientemente del lenguaje) a una corta edad, potencia la creatividad y la habilidad de aprendizaje.

1.1. Situación actual

Actualmente existen muchos proyectos con el fin de introducir la programación como asignatura obligatoria en Educación Primaria y Secundaria³. Estos proyectos vienen respaldados por importantes cambios en los planes de estudios de todo el mundo para enseñar programación[18, 35, 2, 24, 6], marcando una clara tendencia social de incluir la programación en los currículos académicos.

Uno de los proyectos más importantes y con más repercusión a nivel global es Code.org[15]. Propone una serie de herramientas y juegos diseñados especialmente para que los niños aprendan programación mientras juegan con sus personajes favoritos de películas y videojuegos. Todo ello acompañado de una infraestructura para que profesores puedan incluir estos proyectos en las aulas, y padres en sus casas. Su proyecto *Hora del código*[14] consigue reunir a niños de todas las edades una hora al día para que la inviertan en programar. Code.org cuenta con decenas de millones de estudiantes de más de 180 países, disponible en más de 30 idiomas. De manera gratuita, cualquiera puede aprender a programar en eventos que se realizan por todo el mundo (figura 1.1). Code.org está apoyado por grandes compañías y personalidades a nivel global como puede ser Microsoft, Google, el Presidente Barack Obama, Mark Zuckerberg, Bill Gates o Walt Disney Company. En la sección 2.4.2 se abordará más detalladamente las diferentes formas que tiene Code.org de enseñar a programar.

A nivel europeo, la Comisión Europea[16] ha promovido durante el año 2015 la *EU Code Week*[56] como parte de su Estrategia para la Educación y la Formación 2020. Este proyecto consistió en eventos de una semana de duración en la que se

³En este documento se hablará a menudo de alumnos de Educación Primaria, Secundaria o Bachillerato pero no siempre referido al sistema educativo español. Cada país tiene un sistema educativo diferente que varía la edad de escolarización de los alumnos. Por generalizar, se tomará la edad y cursos escolares como referencia cuando se hable de Educación Primaria, Secundaria o Bachillerato. En el Apéndice A se profundiza más en la diferencia de edad en los principales Sistemas Educativos que pueden ser referenciados, directa o indirectamente, en el presente documento.



Figura 1.1: Mapa de visualización de eventos de llevados a cabo por el proyecto *Hora del código* alrededor del mundo. Actualmente 198,473 en todo el mundo, 1,839 en España. Obtenido de [14].

enseñaba informática y programación en lugares de toda Europa⁴.

1.2. La robótica como herramienta de aprendizaje

Es bastante común que la introducción en las aulas del *Computational Thinking* sea a través de la robótica y electrónica. Muchos proyectos de robótica animan al alumno a formarse en dos grandes materias: electrónica y programación. Cuando un estudiante tiene que hacer que un robot se mueva, tendrá que aprender a programar su comportamiento, pero también deberá conocer y en algunos casos incluso crear nuevas piezas para su correcto funcionamiento y personalización.

Existen proyectos como Arduino[7] y Raspberry Pi[26] que proporcionan los elementos y materiales básicos para poder crear nuevos proyectos. Arduino es una placa base con un microprocesador que es capaz de analizar entradas de información (un sensor, un botón o incluso una notificación de Twitter (www.twitter.com) y convertirlo en una respuesta como el activar un robot o encender un led. Todo esto acompañado de un software fácil de programar. El proyecto es completamente *Open Source*. Raspberry Pi es un ordenador del tamaño de una tarjeta de crédito. También se provee por defecto un Sistema Operativo *Open Source* pero el sistema está completamente abierto a cambios y mejoras por parte de la comunidad.

En concreto, una de las metas originales de Raspberry Pi es conseguir que los niños aprendan a programar y conozcan cómo funciona realmente un ordenador. Tanto Arduino como Raspberry Pi pueden adquirirse a un bajo precio, lo que facilita que

⁴En España, se realizaron eventos dentro del marco del proyecto *EU Code Week* en Madrid, Sevilla, Murcia, Asturias, Canarias, Cantabria, Zamora, Cataluña, Ceuta, Badajoz, La Rioja, País Vasco y Valencia.

las instituciones de enseñanza introduzcan estas tecnologías en las aulas como parte de su currículo.

Otro proyecto que también está muy extendido en las aulas es Lego Mindstorm EV3 y NXT[37]. Dos kits de iniciación a la robótica con Lego. Permite al estudiante crear con piezas de lego un robot y programar su comportamiento. Los robots suelen tomar forma de brazo robótico o de vehículo capaz de seguir líneas pintadas en el suelo haciendo uso de sensores. Toda este material robótico viene acompañado de un software para programar el comportamiento de los diferentes componentes del robot y un estilo de programación con bloques[38], similar al usado en Scratch.

En la sección 2.3 se hablará más del uso de la electrónica y la robótica como elementos de enseñanza.

1.3. Proyecto Descubre la programación

Descubre la programación[28] es un proyecto diseñado en la Facultad de Informática de la Universidad de Murcia y tiene como objetivo ayudar a los alumnos de Secundaria y Bachillerato a que desarrollen sus capacidades descubriendo lo que es la informática y aprendiendo a programar. Así como fomentar la inclusión del aprendizaje de la programación en secundaria y bachillerato.

Para ello, en un mismo sitio web, se integra (a) un conjunto de tutoriales de programación; (b) una herramienta que permite programar (figura 1.2) y realizar ejercicios o retos propuestos y (c) una pequeña red social que permite publicar y compartir con el resto de compañeros los programas realizados. Adicionalmente se puede consultar las estadísticas de aprendizaje y tiempo dedicado en la plataforma, tanto por el alumno como por el profesor. De esta manera se permite que los profesores puedan utilizar Descubre en las aulas y realizar un seguimiento de la dedicación y progreso del alumno.

En cuanto a la herramienta para desarrollar programas, el lenguaje utilizado es iJava[49] y ha sido desarrollado por J. A. Sánchez Laguna. iJava es un lenguaje imperativo basado en Java que comparte su sintaxis. Ofrece un enfoque procedural para programar con la Orientación a Objetos como algo opcional. También incorpora un conjunto reducido de funciones de librería clasificadas en los tres grupos siguientes: numéricas, entrada/salida y gráficas⁵.

A modo de ilustración, podemos ver el código 1.1, un programa que imprime por pantalla la cadena "Hello World". En el código 1.2 podemos ver un programa un poco más complejo que dibuja círculos de colores en la pantalla según la posición del ratón. En la figura 1.3 se puede ver la salida de este último programa.

```
1 void main() {
```

⁵Si el lector está interesado en aprender más sobre este lenguaje, puede consultar [49] o [29].

```
1   print("Hello World");
3 }
```

Listado de código 1.1: Programa básico en iJava imprimiendo la cadena "Hello World".

```
1 void main() {
2     //repetimos en bucle la funcion 'draw'
3     animate(draw);
4 }
5
6 void draw() {
7     //coloreamos el circulo segun la posicion del raton
8     fill(mouseX, mouseY, 0); //valores RGB
9     //dibujamos una ellipse con radio 50
10    ellipse(mouseX, mouseY, 50,50);
11 }
```

Listado de código 1.2: Programa en iJava que dibuja un circulo de un color diferente según la posición en la pantalla en la que se encuentra el ratón.

La sintaxis de iJava se parece a muchos de los lenguajes modernos y más utilizados (al ser un subconjunto de la sintaxis de Java), lo cual simplifica el aprendizaje cuando se intenta aprender un nuevo lenguaje. También, gracias a la librería gráfica y matemática, los programas se simplifican mucho en cuanto a complejidad y longitud. De esta manera se consigue aligerar la carga de trabajo que tiene que realizar el alumno para conseguir hacer un programa vistoso, haciendo que la actividad de programar sea más atractiva.

1.4. Motivación y enfoque de este proyecto

Las nuevas tecnologías están haciendo que cada vez más gente de todas las edades se interese por la informática, y más concretamente por la programación. Poco a poco la informática deja de ser cosa de un grupo selecto de gente que entiende su funcionamiento. Como bien argumenta J. Wing en [57] y [58], se debería extender el pensamiento computacional en ambientes pre-universitarios y exponer a los alumnos al pensamiento y a los métodos computacionales.

Como ya hemos comentado anteriormente, existe un movimiento que pretende introducir la informática y la programación en las aulas puesto que se han demostrado los beneficios de enseñar a programar en una edad escolar temprana.

Es en este momento cuando se hace imprescindible tomar decisiones acertadas. Aprender a programar en edades pre-universitaria debe ser algo accesible a cualquier

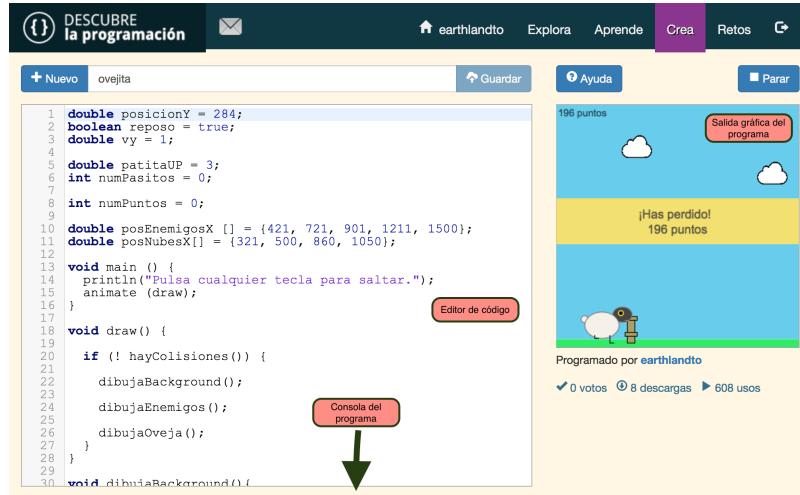


Figura 1.2: Sección *Crea* del proyecto Descubre la programación. Obtenido de [28].

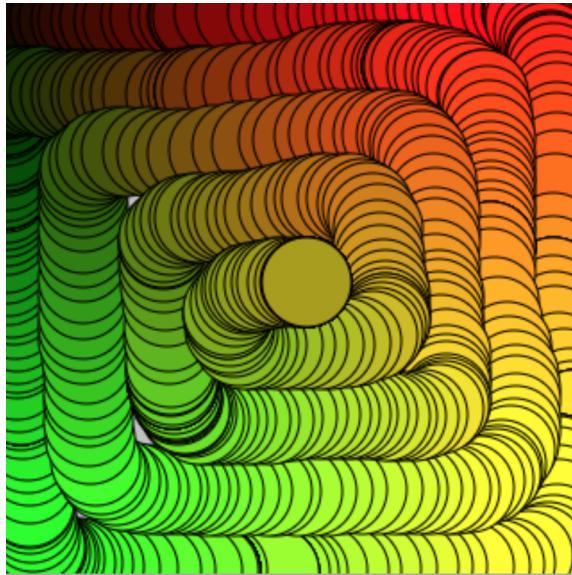


Figura 1.3: Salida del programa escrito en iJava ejecutado por el código 1.2.

estudiante, independiente de su condición o los antecedentes del mismo. Igualmente, los conceptos de programación deben ser presentados de manera incremental, empezando por los más simples para luego ampliar a conceptos más complejos, como defiende L. Fernandez y otros en [22].

Este proyecto se enfocará desarrollando un simulador de robot de dos ruedas en el que los alumnos puedan programar su comportamiento. Se ofrece una librería de funciones para simplificar la interacción con el mismo y conseguir cierta funcionalidad

extra que simplifique la tarea de comprensión y usabilidad del simulador.

El simulador se integrará en la plataforma Descubre la programación, mencionada en la sección 1.3. Esto supone que el estudiante programará el comportamiento del robot simulado en iJava. El robot estará dentro de un circuito con distintos elementos con los que podrá interactuar. Asimismo, el robot dispondrá de una serie de sensores para poder recibir información del mundo que le rodea.

Al incluir el simulador como un módulo de Descubre, se pretende reforzar el esfuerzo por parte de sus creadores de hacer llegar la programación al mayor número de estudiantes pre-universitarios posible, proponiendo una alternativa atractiva y que añade una componente más de entretenimiento a la actividad de programar. Por otra parte, permite que el alumno asimile conceptos de robótica de manera transparente. Al utilizar una plataforma web y de libre acceso, también se busca eliminar la barrera que puede suponer realizar una inversión en material electrónico como puede ocurrir con Arduino, Raspberry Pi o Lego.

Igualmente, los estudiantes que ya están usando la plataforma Descubre, podrán trabajar los conceptos de programación (bucles, condiciones, variables y funciones, entre otros) en un entorno diferente, renovando así su interés en otra actividad y aprender jugando.

En el capítulo 2 se analizaran cuales son las alternativas que existen actualmente que trabajan en esta linea. En el capítulo 4 se verá como se ha desarrollado la idea principal y que resultados se han obtenido. Por último, en el capítulo 5 se analizarán los objetivos conseguidos y que ofrece mi propuesta en comparación a los proyectos actuales.

Capítulo 2

Estado del arte

A nivel global y desde hace varias décadas, existe una gran cantidad de proyectos con la única intención de enseñar, a alumnos de Educación Primaria, Secundaria y Bachillerato, diferentes aspectos de la informática como lo es la programación [50, 15, 4], la robótica [47, 21] e incluso electrónica (con Arduino[7], Raspberry Pi[26] o Lego Mindstorm[38]). La mayoría de estos proyectos promueven una enseñanza independiente y autodidacta bajo un entorno on-line y gratuito. De esta manera, el alumno puede aprender a su propio ritmo y desde cualquier parte del mundo.

En las siguientes secciones se hablará del Lenguaje Logo[34] y el proyecto Turtle[1], un proyecto que lleva décadas activo, con la principal finalidad de enseñar programación y matemáticas a niños. También se estudiarán los diferentes proyectos que existen actualmente y que promueven una enseñanza a niños en entornos web. Por último, se analizarán los diferentes proyectos que enseñan programación jugando y que utilizan robots y/o simuladores.

2.1. Logo y el robot Turtle

El lenguaje Logo, desarrollado a finales de la década de los 60 y basado en Lisp, fue diseñado como una herramienta de aprendizaje a niños. Todas sus características -interactividad, modularidad, extensibilidad, flexibilidad en los tipos de datos- persiguen esta meta. El lenguaje Logo fue uno de los primeros proyectos en emprender la difícil tarea de enseñar a programar a niños de Primaria.

Como se explica en la página oficial del proyecto Logo[34], durante la década de los 70, en el Massachusetts Institute of Technology (MIT) y diferentes centros de investigación europeos, se llevaron a cabo investigaciones sobre el uso del Lenguaje Logo en pequeños grupos de alumnos de Educación Primaria.

A pesar de los diversos estudios que se han realizado a lo largo de los años, no se han conseguido obtener unos resultados claros sobre la posible ventaja de enseñar

a programar a niños de Primaria y Secundaria con Logo. En [23], Feurzeig y otros consiguen una leve mejoría en la nota obtenida en el Test de Habilidades Básicas de Iowa (Iowa Test of Basic Skills, o ITBS)¹. En la tabla 2.1 se puede ver como la nota global del grupo que aprende a programar (denominado *computer*) es de 114 puntos más que el curso anterior, mientras que la obtención del grupo de control es solo de 6 puntos. Aún así, se puede ver como la nota del grupo de control sigue siendo mayor que la del grupo *computer*. Ante éste hecho, Feurzeig concluye que el grupo de control no representaba muy bien al grupo *computer* y que, por tanto, la comparación perdía validez. Más tarde, Pea y Kurland en [42] concluyen que, tras el estudio en dos cursos separados de alumnos, aprender a programar no conseguía mostrar ningún beneficio claro en el rendimiento de los alumnos en comparación al grupo de control. Poco más tarde, Moss [39] obtiene evidencias de la relación en el aprendizaje de Logo con el desarrollo de conceptos primitivos que más adelante se enlazarían con la álgebra básica.

Number of Correct Answers in ITBS			
		Range for <u>Individual Students</u>	<u>Grand Total</u>
7th Grade (1968)	(Computer)	166 - 298	2896
	(Control)	214 - 371	3174
8th Grade (1969)	(Computer)	144 - 305	3010
	(Control)	209 - 382	3180

Tabla 2.1: Tabla que muestra los resultados obtenidos en el ITBS por los alumnos que aprendieron a programar (denominado *computer*) y el grupo de control. Obtenido de [23, p.251].

En el Apéndice B se hace un breve estudio del lenguaje Logo y las principales características que posee el lenguaje.

2.1.1. Turtle de Logo

El proyecto *Turtle* es el proyecto más popular del lenguaje Logo. Nació como una criatura robótica, arreglada para que pareciera una tortuga, que se movía por el suelo con un rotulador atado que pintaba el suelo. Más tarde evolucionó a una

¹El Iowa Test of Basic Skills (ITBS), es un test que se realiza anualmente siguiendo una serie de estándares a nivel de estado para medir el rendimiento académico de los alumnos en materias como: vocabulario, ortografía, álgebra, conceptos aritméticos y comprensión de tablas y gráficos, entre otros.

imagen de una tortuga que se movía por la pantalla pintando según se desplazaba. Turtle se utiliza ampliamente desde hace décadas para enseñar a programar a niños de Primaria y Secundaria conceptos de programación, pero también de matemáticas y geometría[3, 10].

A pesar de su larga vida, el proyecto Turtle ha sabido mantenerse vivo y se ha modernizado en proyectos más recientes como Turtle Academy [5] o Curly Logo [32].

En el apéndice anexo:logo-turtle-lenguaje se puede consultar información sobre cual es el funcionamiento de Turtle.

2.2. Scratch

Scratch[51] es un proyecto creado en el seno del Lifelong Kindergarten Group en el MIT Media Lab, Massachusetts Institute of Technology (MIT). Permite programar juegos interactivos, animaciones y programas de todo tipo en un entorno web. Todo esto apoyado por una gran comunidad de usuarios y profesores que forman una enorme red social de más de 9.5 millones de usuarios registrados y casi 13 millones de proyectos compartidos actualmente².

Scratch está orientado a enseñar a programar a niños de Primaria y Secundaria y como se puede apreciar en la figura 2.1 la mayoría de usuarios de Scratch cumplen dicho perfil, la edad está en torno a los 12 y 15 años.

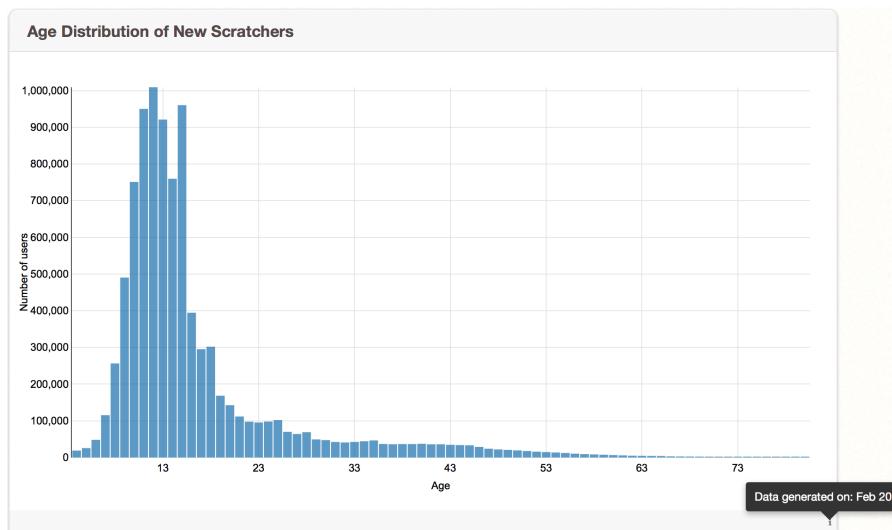


Figura 2.1: Gráfico de edad de los usuarios de Scratch. Obtenido de <https://scratch.mit.edu/statistics/>.

²En la página oficial de Scratch se pueden obtener las estadísticas de uso y participación (<https://scratch.mit.edu/statistics/>).

La idea básica para programar con Scratch es la de crear el flujo del programa mediante la unión de bloques que representan las diferentes funcionalidades del lenguaje (crear o modificar variables, bucles, condiciones, funciones de librería, etc). Existen dependencias entre la forma de los bloques, de esta manera, solo unos bloques *encajaran* con los que tengan una forma compatible. De esta manera, se consigue una relación visual y directa por parte del programador entre los bloques que puedes unir, y sobretodo, de lo que no puedes hacer.

Para saber más sobre el funcionamiento de Scratch y como se programa con esta herramienta se puede consultar el apéndice E.

Como conclusión, Scratch es un entorno completo y autosuficiente con una gran y compleja funcionalidad que permite a niños y mayores realizar aplicaciones sin necesidad de saber programar pero aprendiendo conceptos básicos como la repetición, el control de instrucciones y como llevar a cabo una animación.

2.3. Aprendiendo con robots

En secciones anteriores se han analizado dos formas de aprender a programar. Logo, junto a Turtle, fue el primer lenguaje de programación que se creó para enseñar a programar a niños de Primaria y Secundaria. Por otra parte, Scratch es actualmente un gran promotor del *Computational Thinking* con una cantidad considerable de usuarios activos.

No obstante, existen otros enfoques para enseñar programación y desarrollar el pensamiento computacional. La robótica es una opción muy extendida y que atrae a partes iguales a alumnos y docentes. En las siguientes secciones se estudiarán diferentes alternativas que utilizar tanto simuladores como robots reales.

2.3.1. Lego Mindstorm NXT/EV3

Como se ha mencionado en la sección 1.2, el proyecto Lego Mindstorm[37] está orientado a enseñar robótica a estudiantes de Primaria y Secundaria. Lego proporciona dos kits para crear tu propio robot: NXT y EV3. Lego Mindstorm NXT³ fue lanzado en 2006, y su siguiente versión, Lego Mindstorm EV3, salió al público en 2013. Tanto la versión NXT como EV3 son ampliamente utilizadas y existen una gran cantidad de competiciones que fomentan su uso.

Cada kit está compuesto por elementos Software para poder programar el comportamiento del robot y elementos Hardware como sensores, accesorios, piezas de lego y el *Brick* (ladrillo). El *Brick* es un aparato con el que se pueden programar comportamientos básicos a las piezas conectadas al mismo, pero principalmente es

³Actualmente, Lego Mindstorm NXT tiene soporte por parte de Lego hasta finales del año 2015.

el cerebro de todo el robot. Las piezas son mayormente compatibles entre los dos sistemas, la diferencia radica en el *Brick*.

En concreto, EV3 viene acompañado del *Lego Mindstorm EV3 Home Edition Software* y su *brick*, el *Brick P EV3*. De esta manera se puede programar instrucciones al robot con bloques (o iconos) que recuerdan a Scratch. Los bloques representan diferentes funciones del lenguaje y el programador debe conectar dichos bloques para crear el flujo de ejecución del programa. Lego también ofrece *EV3 Programmer*, una versión gratuita y simplificada (con menos funcionalidad y bloques) para programar los robots de Lego.

Los bloques se dividen en: bloques de acción, bloques de flujo, bloques de sensores, bloques de operación de datos y bloques avanzados. Los bloques de acción permiten controlar la rotación de los motores y las luces del *Brick P EV3*. Los bloques de flujo definen cual será el flujo del programa, cuyo inicio se define con el *bloque de inicio*. Los bloques de sensores, entre otras funciones, controlan los sensores táctiles y de color conectados al *brick*. Los sensores de operación de datos permiten definir variables, realizar cálculos matemáticos u obtener valores aleatorios. Por último, los bloques avanzados permiten administrar ficheros o establecer conexiones Bluetooth.

Por otra parte, existen muchas aplicaciones de terceros no oficiales que permiten utilizar lenguajes de programación de alto nivel para controlar el robot, como pueden ser Python, Lua, .Net, C/C++, Java o Robomind (del cual hablaremos en la sección 2.3.4).

2.3.2. Moway

El robot Moway[21] es un robot desarrollado como herramienta educativa. Moway se compone por un conjunto de sensores y actuadores que le permiten recibir información del mundo real y reaccionar a la misma según las instrucciones que se le hayan proporcionado.

Los actuadores de los que dispone el robot Moway son: dos ruedas, luces LEDs y un altavoz. Por otra parte, los sensores de los que dispone Moway son:

- Un sensor de luz que le permite detectar el nivel de luz ambiente.
- 2 sensores de linea, mirando hacia el suelo, que detectan diferencias de color.
- 4 sensores de obstáculos en la parte frontal del robot que detectan la distancia a los mismos.
- Un micrófono para detectar el nivel de ruido.
- Un sensor de temperatura.
- Un acelerómetro que detecta inclinación y fuerzas.

La aplicación gratuita Moway World permite crear diagramas de flujo para programar el comportamiento del robot, con funciones para activar y controlar los diferentes actuadores del robot Moway en función de la información obtenida por los sensores. También permite programar su funcionamiento con Scratch, el cual se ha analizado en la sección 2.2.

Moway tiene varios módulos que pueden ser añadidos para ampliar su funcionalidad o manejo, como son: módulos de radiofrecuencia, placas Raspberry Pi, tarjetas wifi o una cámara para captar imágenes en tiempo real. También se ofrece una alternativa a Moway de código abierto, llamada Mowayduino. Mowayduino está basado en Arduino y tiene toda la funcionalidad y potencia que ofrece Arduino. Los kits que se disponen inicialmente para Mowayduino son los mismos que para Moway. Para programar un Mowayduino se proporciona el entorno de programación oficial de Arduino.

Un uso común de Moway es como robot *sigue-lineas*. Los sensores de linea de los que dispone proporcionan información para poder conocer el estado del robot con respecto a una linea, es decir, si está sobre una linea o no. Al tener 2 sensores, puede conocer si se está saliendo de la linea (quizás porque esta realiza una curva) y en la dirección en la que esto ocurre. Por ejemplo, si el sensor derecho deja de detectar la linea, quiere decir que: o la linea está realizando una curva hacia la izquierda o que el robot Moway está moviéndose hacia la derecha fuera de la linea (lo cual ocurre en ambos casos, si se analiza la situación desde el punto de vista de la linea). Por tanto, al estar saliéndose de la linea por la derecha, el robot Moway tendrá que moverse hacia la izquierda para mantenerse dentro del la misma.

En general, Moway ofrece un robot para programarlo y configurarlo junto a un entorno de programación, como puede ser Scratch. Es un buen inicio en la robótica pero su software privativo deja las opciones de ampliación y de creación de complementos y terceras partes un poco limitadas.

Moway, junto a Robomind, será uno de los pilares en los que se basará el simulador que se desarrollará, en especial la parte que tiene que ver con las características del simulador.

2.3.3. Karel the Robot

Karel⁴ es un lenguaje de programación creado por R. Pattis en su libro *Karel the robot: A gentle introduction to the Art of Programming* (El robot Karel: Una introducción gentil al arte de programar) en 1981[41]. En él, Pattis enseña conceptos de programación siguiendo una metodología *Early bird*[8], que consiste en enseñar primero los conceptos más importantes.

⁴El nombre de Karel the robot es nombrado en honor a Karel Čapek, un escritor checo que introdujo el término *robot* por primera vez.

Karel se mueve en un escenario discreto con celdas y con el que se puede interactuar. Las órdenes básicas del robot Karel son: move para avanzar una posición en la dirección que está mirando; turnLeft para girar 90° en sentido contrario a las agujas del reloj; putBeeper y pickBeeper para depositar o recoger, respectivamente, un *beeper* (localizador) en la posición en la que se encuentra Karel; y por último, turnOff, para apagarse a si mismo y finalizar la ejecución del programa.

Adicionalmente, Karel también proporciona instrucciones para comprobar si hay obstáculos o un *beeper* en la posición que se encuentra. Toda esta funcionalidad viene acompañada de instrucciones de control, bucles y operaciones aritméticas.

2.3.4. Robomind

Robomind Academy[47] es una plataforma que busca potenciar y extender el *Computational Thinking* entre alumnos de Educación Primaria y Secundaria. Sus creadores opinan⁵ que la informática debería tener el mismo estatus que las matemáticas o la lengua, algo esencial en la formación del estudiante. Argumentan que, a través del *Computational Thinking* se podrán realizar avances en el cuidado de la salud o la industria, algo totalmente necesario en pleno Siglo XXI.

Actualmente, Robomind ofrece un entorno de programación multiplataforma: aplicación de escritorio, entorno web y aplicación para Tablet. En la figura 2.2 podemos ver un ejemplo de como es el entorno de programación que propone Robomind.

El proyecto en sí está orientado a que profesores y alumnos trabajen conjuntamente en desarrollar las actividades, permitiendo que los profesores puedan realizar un seguimiento del avance y trabajo de los alumnos a cargo. También permite que los alumnos puedan registrarse independientemente y aprender a su propio ritmo. Los cursos están divididos entre alumnos de Educación Primaria y Secundaria.

Las soluciones antes mencionadas son de pago. Dependiendo de si eres profesor o estudiante y de si quieres (o no) utilizar la versión de escritorio, se ofrecen distintos planes de compra. A parte, Robomind ofrece planes gratuitos para alumnos con características más limitadas o una cantidad inferior de cursos. Adicionalmente, Robomind participa en el proyecto *La Hora del Código*[14], ofreciendo una introducción a Robomind completamente gratuita. Una de sus funciones más interesantes es la de poder traducir el código escrito en Robomind para poder utilizarlo con tu propio Lego Mindstorm⁶.

Por último, tanto la interfaz, la página web, la aplicación o los tutoriales está

⁵En la sección *Visión* de la página web oficial de Robomind Academy se puede ver cuales son los principios y metas detrás del proyecto (<https://www.robomindacademy.com/go/robomind/about#vision>). Consultado el 2 de febrero de 2016.

⁶Ya se ha hablado de Lego Mindstorm en la sección 2.3.1. Para más información sobre Lego Mindstorm NXT se puede acudir a [37].

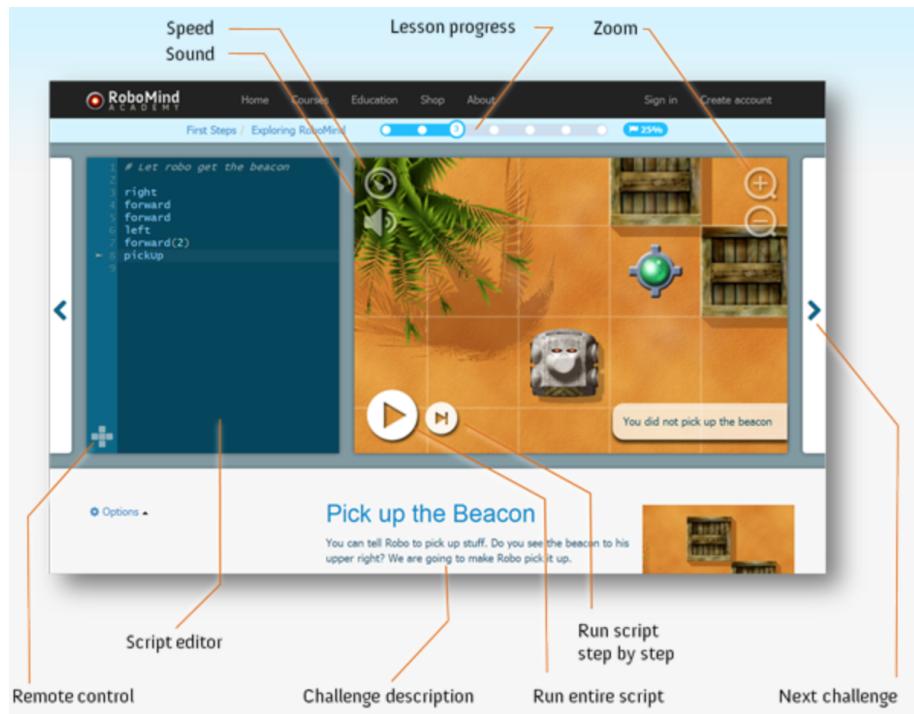


Figura 2.2: Entorno de programación en Robomind Academy. Obtenido de [47].

completamente en inglés, ofreciendo una cierta barrera de entrada a todo aquel interesado que no sea angloparlante.

Para ampliar información de como se programa Robomind, se puede consultar el apéndice D.

Como se puede ver, Robomind está formado por elementos de *Karel the robot*, como lo es el movimiento en un escenario discreto y la forma de interactuar con el entorno, y por *Turtle* de Logo y su capacidad de dejar un rastro por donde va pasando.

2.4. Aprendizaje Online

En la sección anterior se han analizado diferentes alternativas para aprender a través de la robótica. Algunas de estas alternativas tienen una gran componente Hardware, como puede ser Lego Minstorm. Por otra parte, Robomind, como simulador de un robot, permite al alumno iniciarse en la robótica sin necesidad de un robot.

Aunque es una práctica bastante utilizada, usar un robot no es la única alternativa para aprender a programar. Usar un entorno online para enseñar a programar es una

alternativa que ha obtenido notable popularidad últimamente. Como se vio en la sección 2.2, Scratch es un gran exponente del *Computational Thinking* que utiliza un entorno principalmente online para que niños de todo el mundo aprenda a programar. Aunque éste no es el único proyecto que existe en esta linea. CodeHS y CodeSchool son algunos de los proyectos con más repercusión que buscan extender la cultura del *Computational Thinking*, especialmente entre niños. En las siguientes secciones se estudiaran estas alternativas.

2.4.1. CodeHS

Otro proyecto muy extendido que promueve el *Computational Thinking* es CodeHS. CodeHS es una plataforma online con la que personas de todas las edades pueden aprender informática. Tienen cursos de introducción a la informática, programar con diferentes lenguajes, bases de datos, e incluso una versión modernizada de Karel the robot, el cual se ha analizado en la sección 2.3.3.

CodeHS ofrece muchas opciones de aprendizaje independiente, incluso con tutores personalizados para resolver dudas de los usuarios, está especialmente dirigido a enseñar programación en colegios e institutos. Ofrece planes de pago para implantar la plataforma en las clases o herramientas de seguimiento y evaluación para que el profesor pueda supervisar el rendimiento de los alumnos. Aunque los planes gratuitos ofrecen un reducido número de cursos, permite al usuario programar libremente en una gran selección de lenguajes que tiene disponible, como pueden ser Javascript, Ruby o Python.

2.4.2. Code.org

Code.org[15] es una plataforma on-line cuyo objetivo es extender el desarrollo de las habilidades relacionadas con el pensamiento computacional entre niños de todas las edades. Actualmente cuenta con más de 8 millones de usuarios y está respaldada por una gran comunidad de educadores y gente influyente de todos los sectores, no solamente el de la informática.

Uno de los proyectos con más repercusión es el *Hour of Code*[14] (Hora del Código) que intenta que todos los niños dediquen una hora diaria a programar. Esta idea está apoyada por muchas empresas que ponen sus recursos y productos a disponibilidad de Code.org para que puedan hacer uso de estos. Uno de ellos, por ejemplo, es Robomind, que como ya se comentó en la sección 2.3.4, tiene una modalidad gratuita con pruebas para aprender a utilizar el simulador del robot Robomind. Marcas como Disney, Flappy bird o Rovio⁷ ponen a disposición de Code.org y los niños el uso de sus personajes, intentando captar la atención de los niños y haciendo la actividad

⁷Rovio es una empresa conocida por desarrollar los populares juegos de smartphone Angry Birds.

altamente atractiva. También, personas influyentes del panorama internacional se ofrecen a explicar conceptos de programación durante la Hora del Código a los niños.

Para programar, en Code.org se utiliza una versión simplificada de Scratch (el cual se ha estudiado en la sección 2.2) con la misma mecánica. Code.org propone muchos cursos independientes y decenas de horas de aprendizaje, con actividades nuevas para captar la atención del niño.

Capítulo 3

Análisis de objetivos

Este Trabajo Fin de Grado consiste en desarrollar un módulo de simulación de un robot para integrarlo en la plataforma Descubre la programación(descubre.inf.um.es).

El robot, denominado *Robode*, será simulado como un robot real, con movimientos y reacciones realistas y en un entorno continuo. El simulador dispondrá de una serie de características básicas:

- Robode será un robot de dos ruedas que se moverá simulando el máximo realismo posible.
- Cada rueda de Robode se moverá de manera independiente, simulando un motor para cada rueda.
- Robode se encontrará en un circuito finito limitado por *fronteras*.
- El circuito podrá tener *lineas* pintadas en el suelo y *obstáculos*.
- Las *lineas* pintadas el suelo no colisionarán con Robode.
- Los *obstáculos* podrán colisionar con Robode, produciendo un efecto en el contacto de ambos cuerpos, desplazándolos según la velocidad y características físicas de cada uno.
- Dispondrá de 2 sensores orientados hacia abajo que detectan diferencias de color. Esta capacidad se usará para detectar *lineas* pintadas en el suelo de color negro.
- Dispondrá de sensores de proximidad delanteros y traseros, que detectarán si se va a producir una colisión con un *obstáculo* o una *frontera*.

- Los sensores serán ajenos al número de cuerpos que están detectando. Solo distinguirán si se está detectando colisiones o no.

El simulador se integrará con Descubre, una plataforma para aprender programación utilizando el lenguaje iJava¹. Por tanto, será necesario entender el funcionamiento interno de la plataforma y modificarla apropiadamente para poder integrar Robode con Descubre. De esta manera, se añadirán las funciones necesarias para poder crear programas en iJava y controlar el robot. Los objetivos relativos a la creación del lenguaje que controlan el robot son:

- Se proporcionará un conjunto de funciones que permitan controlar el robot.
- Se podrá mover el robot definiendo una fuerza para cada rueda.
- Se podrá conocer el estado de cada sensor, es decir, si está detectando un objeto o no.
- Se dotará únicamente de las funciones básicas del lenguaje, para favorecer que los alumnos realicen funciones más complejas por ellos mismo.

3.1. Tecnologías usadas

Robode utiliza una serie de tecnologías web sobre las que trabaja para poder integrarse en el entorno de Descubre y crear un simulador en un entorno online. A continuación se describirán brevemente estas tecnologías.

- **HTML** o *HyperText Markup Language* (lenguaje de marcas de hipertexto) es un estándar que proporciona la estructura de las páginas web. En concreto, se ha utilizado la última versión de HTML, HTML5, que proporciona elementos como *canvas*, el cual usaremos para dibujar nuestro robot.
- **CSS** son las siglas para *Cascading Style Sheets*. Es un lenguaje para definir el estilo (o presentación) de la página web sobre HTML.
- **Javascript**, como se le conoce comúnmente, es un lenguaje para programar páginas web que trabaja junto a HTML. Es un lenguaje interpretado por los navegadores, con orientación a objetos basada en prototipos, imperativo, con tipado débil y dinámico. Javascript está estandarizado por ECMA International² y su nombre oficial es ECMAScript. La versión más extendida de Javascript actualmente es ECMAScript 5. Aunque una nueva versión se hizo pública en junio de 2015[27].

¹Para más información sobre la plataforma Descubre y el lenguaje iJava se puede consultar la sección 1.3.

²Página oficial de ECMA International (<http://www.ecma-international.org>).

3.2. Herramientas utilizadas

Las herramientas que se han usado para desarrollar este proyecto se describen a continuación:

- **Atom**³ como editor de código. Acompañado de paquetes *linter*⁴ para JavaScript, HTML y CSS que detectan la mayoría de los errores sintácticos que se producen a la hora de programar. También se han utilizado diferentes paquetes para tabular y organizar el código automáticamente.
- Como herramienta de control de versiones se ha utilizado **Git**⁵ y como repositorio **GitHub**⁶.
- Para las pruebas, ejecución y depuración de la aplicación se ha utilizado tanto **Google Chrome**⁷ versión 48, como **Safari**⁸ versión 9.0.3.
- Para el desarrollo de esta documentación se ha utilizado **MacTex** como motor de *Latex*⁹ y **TexMaker**¹⁰ para la redacción del documento. También se ha utilizado **BibTex**¹¹ como gestor bibliográfico.

³Página oficial de Atom (<https://atom.io>).

⁴Un *linter* analiza el código escrito en busca de errores tipográficos. Para más información sobre el *linter* de Atom, se puede consultar la página web de la herramienta: <https://atom.io/packages/linter>.

⁵Página oficial del proyecto Git (<https://git-scm.com>).

⁶Página oficial de GitHub (<https://github.com>)

⁷Página oficial de Google Chrome (<https://www.google.es/chrome/browser/desktop/>).

⁸Página oficial de Safari (<http://www.apple.com/es/safari/>).

⁹Página oficial del proyecto Latex (<https://www.latex-project.org>).

¹⁰Página oficial de TexMaker (<http://www.xmlmath.net/texmaker/>).

¹¹Página oficial de BibTex (<http://www.bibtex.org>).

Capítulo 4

Diseño y resolución del trabajo realizado

Después de estudiar cual es la situación actual de la enseñanza de la programación a un nivel global y los diferentes proyectos que promueven el *pensamiento computacional*, en este capítulo se analizarán cuales son las necesidades del proyecto Robode y como se han resuelto.

Como ya se ha mencionado anteriormente, se ha creado un simulador de un robot, al que hemos denominado *Robode*, que se integrará en la plataforma Descubre la programación. Robode se encuentra dentro de un mundo en el que puede moverse con libertad y con el que interactua. El robot posee dos ruedas que se mueven independientemente (con un motor para cada una de ellas) y dispone de sensores que informarán de posibles colisiones con obstáculos. También, el robot puede detectar color en el suelo debajo del mismo. Ésta capacidad se usará para detectar líneas pintadas de negro en el suelo.

Adicionalmente, el robot y el mundo en el que este se encuentra será dibujado en un elemento *canvas* de HTML5, al igual que ocurre en la plataforma Descubre. El *canvas* establece un *lienzo* en el que se pueden dibujar gráficos¹. De igual modo, HTML5 proporciona muchas funciones para dibujar en el *canvas* de manera relativamente sencilla.

Como se puede ver, Robode es la unión de varios de los proyectos que se han analizado en el capítulo 2. Por una parte, la idea principal de crear un simulador es igual a la que se ha realizado en Robomind (el cual se analiza en la sección 2.3.4) pero simulando un robot en un mundo continuo. El robot comparte características físicas con el robot Moway (analizado en la sección 2.3.2), como es la disposición de sensores (tanto externos como inferiores) y la cantidad y disposición de las ruedas.

¹Los gráficos pueden ser en 2 o 3 dimensiones, pero en el simulador que se desarrollará solo se hará uso de gráficos 2D.

Por último, se ejecutará en un entorno web, como bien puede ser el ejemplo de las plataformas CodeHS (sección 2.4.1) y Code.org (sección 2.4.2).

En las secciones siguientes se va a estudiar como se ha diseñado y creado Robode, que características tiene y cual ha sido la implementación llevada a cabo. También se verá como se ha modificado la plataforma Descubre la programación para integrar el simulador en ella. Pero antes, se analizará el nuevo modelo de programación que se ha creado y en el que se basa el simulador.

4.1. Modelo de programación del simulador

El modelo de programación que presenta iJava actualmente consiste en crear una función principal `main` sin parámetros y que devuelve el tipo `void`. También, se puede usar la función `animate()`, tomando como argumento una función y ejecutándola en bucle. El resto es similar a un lenguaje imperativo. Un programa de ejemplo en iJava se puede encontrar en el código 1.2.

El modelo de programación que se plantea para usar el simulador no es muy diferente. Para definirlo, se han estudiado los lenguajes de programación que usan Moway, Turtle, Scratch y Arduino y que ya se han mencionado anteriormente en este documento.

Primeramente se tiene una función principal, que se llamará `setup` y que no recibirá parámetros y devuelve el tipo `void`. Esta función cumple el mismo papel que la función `main` de iJava, pero se ha cambiado el nombrado para adecuarlo al estándar de Arduino y darle el significado de establecer los parámetros del robot y su comportamiento inicial.

Por ejemplo, en la función `setup` se podría programar el robot de manera cuando inicia el programa, éste gire 180° y el programa termine. En `setup` se puede programar un comportamiento estándar del robot. Pero también es necesario permitir que el robot reaccione al entorno y a posibles cambios que en este se producen. Por tanto, es necesario añadir otro elemento al modelo.

La función `loop` ejecutará en bucle todas las instrucciones que en ella se encuentren. De esta manera, se permite una ejecución infinita que permita reaccionar a cambios del entorno. Esto se hará con la información que le llegue al robot a través de los sensores.

No es necesario indicar que la función `loop` se ejecute en bucle, a diferente de iJava. En iJava se usaba la función `animate` que recibe una función como argumento y esta es ejecutada en bucle. La función `loop` se comportará automáticamente de esta manera. En realidad, lo que hará el motor de iJava será llamar por debajo a la función `loop` en bucle.

En cuanto a la funcionalidad que ofrece la API del simulador para poder controlar al robot, se busca que las funciones ofrecidas sean lo más básicas posibles.

La intención es que el programador tenga los elementos básicos y suficientes para manejar el robot. Si quisiera funciones más complejas, tendría que programarlas él mismo, contribuyendo así a su aprendizaje.

A continuación vamos a listar todas las funciones que ofrece la API para controlar el robot. Además, la API también ofrece un conjunto de variables de tipo boolean que permitirán saber si los sensores están detectando colisiones o no.

- `initRobot ()`: Inicia, muestra y coloca el robot en su posición inicial.
- `power (potenciaIzq, potenciaDer)`: Establece la potencia con la que se moverá cada rueda. El primer argumento es para la rueda izquierda mientras que el segundo para la derecha. El valor de potencia estará entre 40 y -40. Un valor positivo moverá la rueda en dirección frontal y un valor negativo, posterior.
- `stop ()`: Detiene el robot.
- `left ()`: Gira el robot 90º en dirección contraria al sentido de las agujas del reloj.
- `wait (n_milisegundos)`: Crea una espera igual al número de milisegundos que se le pasan como argumento a la función.
- `sensorNW`: Sensor en la posición Noroeste.
- `sensorNE`: Sensor en la posición Noreste.
- `sensorSW`: Sensor en la posición Suroeste.
- `sensorSE`: Sensor en la posición Sureste.
- `collisioning`: Devuelve `true` si alguno de los sensores anteriores está detectando una colisión.
- `sensorLL`: Sensor inferior izquierdo que detectará si está sobre una linea.
- `sensorLR`: Sensor inferior derecho que detectará si está sobre una linea.

Como se puede ver, se ofrecen diferentes variables de tipo boolean para saber si el robot está detectando una colisión o si está sobre una linea con alguno de los sensores. Por otra parte, se ofrece la función `wait` que produce una espera de tiempo² en la que no se ejecutarán nuevas instrucciones.

Un ejemplo que muestra el nuevo modelo de programación se puede ver en el código 4.1.

²En la sección 4.3.6 se explica en detalle como se ha conseguido este efecto en el sistema *monohilo* que implementa Javascript.

```

1 int delay;
2 int potenciaDer;
3 int potenciaIzq;
4 void setup(){
5     delay = 1000; // 1 segundo en milisegundos
6     potenciaDer = 10;
7     potenciaIzq = 10;
8     //iniciamos el robot
9     initRobot();
10 }
11
12 void loop(){
13     //movimiento de las ruedas
14     power(potenciaIzq, potenciaDer);
15     // esperamos
16     wait(delay);
17     // detenemos el robot
18     power(0, 0);
19     //esperamos el doble de tiempo
20     wait(delay * 2);
21 }
```

Listado de código 4.1: Programa de ejemplo que mueve al robot con el nuevo Modelo de programación establecido para el simulador.

Los pasos descritos del funcionamiento del programa se enumeran a continuación:

1. Se declaran las variables `delay`, `potenciaDer` y `potenciaIzq`.
2. Se ejecuta la función `setup()`:
 - Se establecen los valores de las variables `delay`, `potenciaDer` y `potenciaIzq`.
 - Se inicia el robot. Esto solo creará el robot en la posición inicial.
3. Se ejecuta la función `loop()`:
 - Se establece la potencia de las rueda derecha a `potenciaDer` y de la rueda izquierda a `potenciaIzq`.
 - Se produce una espera de cantidad `delay` (1 segundo) en el que no se admiten nuevas instrucciones.
 - Se establece la potencia de las rueda derecha a 0 y de la rueda izquierda a 0.
 - Se produce una espera de 2 segundos en el que no se admiten nuevas instrucciones.

4. Se repite el paso 3.

Con respecto a función `loop`, puesto que ya existe un bucle infinito en el programa, no es una buena práctica crear bucles infinitos³ en el simulador.

Por ejemplo, la forma de reaccionar a las colisiones sería la que se muestra en el código 4.2. En lugar de utilizar un bucle `while` que espere hasta que se produzca una colisión, se comprueban con condiciones `if` o `if-else`.

```

1 void setup() {
    initRobot();
3 }

5 void loop() {
    //por defecto, hacia delante
    power(10, 10);
    // Si colision delante, retrocedo hacia la izquierda
9     if (sensorNW || sensorNE) {
        power(-10, -20);
11    }
    //si colision detras avanzo torciendo
13    if (sensorSW) {
        power(20, 10);
15    }

17    if (sensorSE) {
        power(10, 20);
19    }
}

```

Listado de código 4.2: Ejemplo de buenas prácticas en el uso de la API del simulador.

Recapitulando, tenemos una función principal `setup` que puede ser complementada con la función `loop`, la cual ejecutará en bucle infinito que permitirá reaccionar al entorno según pasa el tiempo.

En la sección siguiente se estudiará como se ha modificado iJava para implementar este nuevo modelo de programación.

4.1.1. Modificación del lenguaje iJava

Para poder conseguir lo que se ha descrito anteriormente, es imprescindible realizar ciertos cambios en el compilador de iJava para que reconozca y ejecute las nuevas funciones de la API del robot.

³Obviamente, no hay ningún problema en definir bucles que recorran variables o realicen cálculos, por ejemplo.

Principalmente, el compilador de iJava consta de tres partes: el analizador sintáctico, el analizador semántico y el *sandbox*. El *sandbox* contiene la implementación de las funciones de librería descritas en el analizador sintáctico y ejecuta (en un entorno seguro) el código Javascript que se genera a partir del código fuente en iJava si todo el proceso de compilación se ha realizado con éxito.

Para que las funciones de la API sean parte del lenguaje y se ejecuten como una más de iJava, hay que realizar tres grandes cambios. Estos son: (a)modificar el analizador semántico para cambiar la función principal `main` por `setup`; (b) añadir la definición de las funciones al analizado sintáctico y (c) añadir la implementación de las funciones al *sandbox*.

Para el cambio (a), hay que modificar el analizador semántico de iJava (clase `iJavaSemantic`). El analizador semántico busca si la función `main` se ha definido y, si no la encuentra, devolverá un error que se mostrará por pantalla. Además, en el código Javascript que se genera como ejecución del programa del usuario (función `eval()`), se ha definido que la función `loop()` se ejecute en un bucle infinito siempre y cuando se haya definido en el programa del usuario.

Para añadir las definiciones de nuevas funciones (cambio (b)), hay que añadir al analizador sintáctico (clase `iJavaParser`) la definición de dichas funciones, así como el número y tipo de parámetros que obtendrán las funciones. Para el caso de las variables de los sensores, también hay que añadirlas como variables del sistema (de iJava) de tipo `boolean`.

Por último, el cambio (c) se consigue implementando las funciones de la API en el *sandbox* de iJava (clase `iJavaSandbox`). Las funciones y variables tienen que tener el mismo nombre, tipo y argumentos que las definidas en el analizador sintáctico para que se puedan ejecutar y el compilador las detecte.

En el apéndice F se incluye las secciones más importantes del código que implica los cambios arriba mencionados.

Adicionalmente a estos cambios, también se han eliminado las funciones de librería de gráficos y de entrada estándar (teclado y ratón) puesto que no aportaban interés a la programación con un robot y además podría desvirtuar el modelo de programación del simulador.

4.2. Integración en Descubre del simulador

Descubre la programación[28] es una plataforma online que está enfocada en enseñar a programar a alumnos de Educación Primaria y Secundaria. Para ello, se provee de un entorno que permite programar en iJava[49] y compilar el código ejecutado. La salida se muestra por una pantalla dispuesta para ello.

Esto se realiza gracias al compilador que implementa Descubre. Dicho compilador obtendrá el código escrito en iJava y lo analizará en busca de errores. Si el análisis

sintáctico y semántico no encuentra ningún problema, entonces realizará una traducción de iJava a Javascript para ejecutar dicho código en el navegador. La salida se mostrará por el elemento *canvas* de HTML5.

Para explicar como está creada la plataforma Descubre, se muestra el diagrama de clases de Descubre en la figura 4.1. La clase principal de Descubre es (*index*) , que representará al fichero HTML principal.

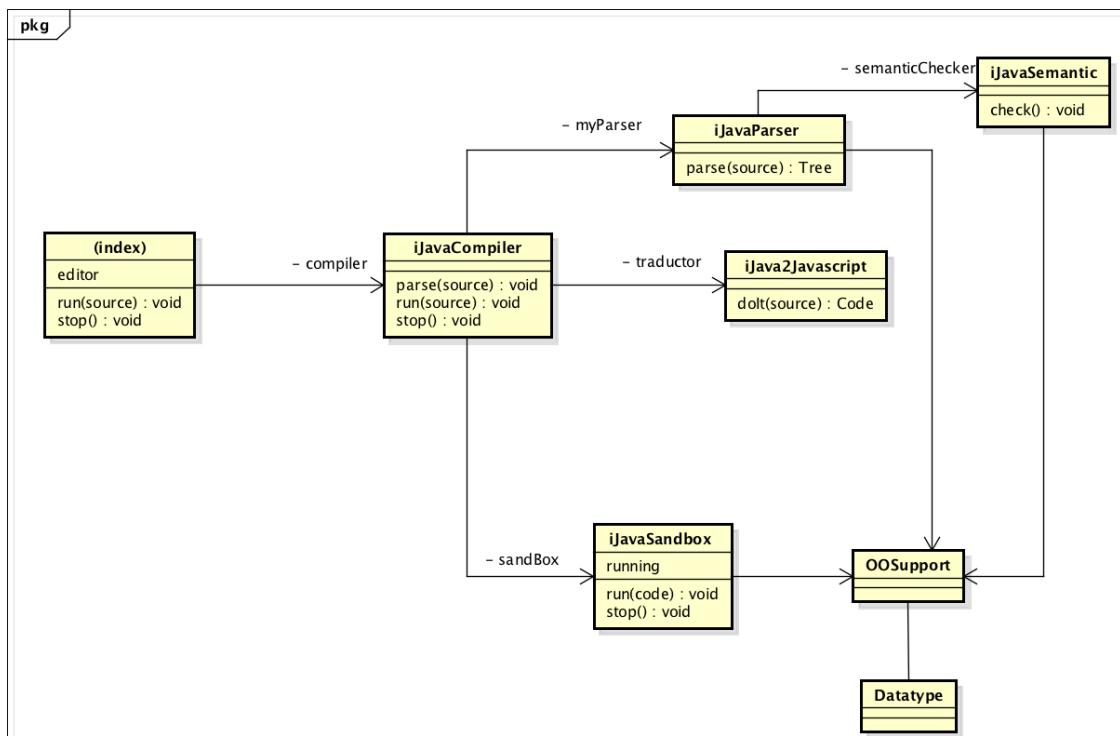


Figura 4.1: Diagrama de clases de la plataforma Descubre: punto de partida.

Para entender como se relacionan todas estas clases y cual es la comunicación entre ellas, en la figura 4.2 se puede ver un diagrama de comunicación simple que se produciría cuando un usuario hace click en el botón *run* de Descubre.

En resumen, el diagrama de comunicación muestra cuales son los pasos que se llevan a cabo para ejecutar un código escrito en Javascript. También hay que tener en cuenta que el *canvas* no es una clase al uso. Aún así, se ha decidido incluirlo de esta manera para poder representar quién realiza la acción de pintar (método `draw()`). Esto cambiará cuando entre en juego el simulador.

Es la clase *iJavaSandbox* la que ejecuta el código convertido a Javascript con la función `eval()` de Javascript. En esta clase estará la implementación de toda la función de librería que ofrece iJava en el lenguaje: funciones matemáticas, para pintar el *canvas*, entrada y salida, etc.

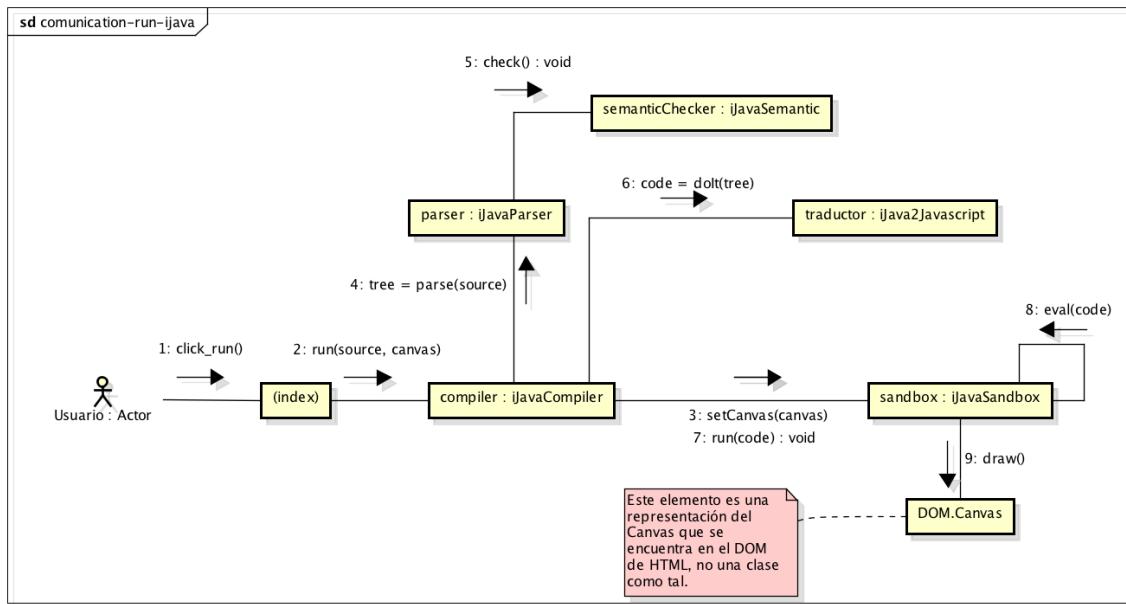


Figura 4.2: Diagrama de comunicación de la función `run()`.

En las secciones siguientes se analizará que cambios se han realizado para integrar el simulador del robot en Descubre.

4.2.1. Modificación de la plataforma Descubre

En la sección 4.1.1 se ha hablado de los pasos necesarios para modificar el compilador de iJava para añadir nuevas librerías. Pero para añadir el simulador, es necesario añadir toda la lógica de la simulación al compilador. El control del robot y del mundo se implementa en la clase Robode.

La clase Robode recibirá los mensajes del *Sandbox* que le dirá como tiene que mover el robot (potencia de las ruedas, detenerse por completo, etc) y, a la misma vez, Robode le comunicará al *Sandbox* cuando un sensor está detectando colisiones. Simulator encapsula detalles de la simulación, como puede ser la configuración inicial del robot (posición, tamaño, etc) o del mundo (ancho y alto por ejemplo).

Además, se han representado elementos básicos del motor físico que se ha utilizado para la simulación. Esto es importante ya que otorga una visión general del funcionamiento del motor físico y su relación con el simulador y Robode. Esto último será útil cuando se vea el uso e implementación del motor físico en la sección 4.3.

En la figura 4.4 se puede ver el nuevo diagrama de clases con la inserción del simulador a iJava. También, en la figura 4.3, se pueden ver los pasos que se producirían al ejecutar un código iJava que ejecuta funciones del robot.

Como se puede ver, el *Sandbox* envía dos mensajes diferentes para comunicarse con Robode indicando el *canvas* y que tiene que ejecutar cierta orden. Ahora, es la clase Robode la que dibujará en el *canvas* y no el *Sandbox*. La razón de esto se explica más adelante, en la sección 4.3.6.

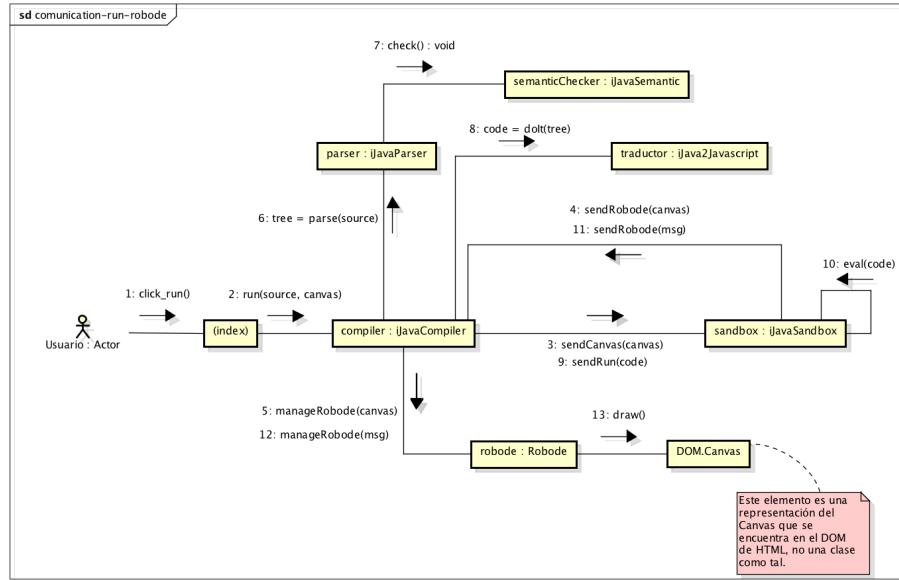


Figura 4.3: Diagrama de comunicación que se produce cuando se ejecuta un código Java después de integral el simulador con Descubre.

4.3. Motor físico

El simulador representa un mundo con un robot y una serie de obstáculos que interfieren entre ellos. Los obstáculos pueden colisionar entre ellos o con el robot. También es necesario simular líneas en el suelo y la detección de las mismas. Todo ello con un paso del tiempo realista. Para ello, se necesita utilizar un motor físico que simule la creación de cuerpos en el mundo, su movimiento por el mismo y el contacto entre ellos, generando una reacción similar a la producida en el mundo real.

Para la creación del mundo y de las físicas que gestionan las colisiones y la interacción entre todos los elementos del simulador se han estudiado dos motores físicos: PhysicsJS⁴ y Box2D⁵ [9]. Dos librerías muy versátiles y de software libre que trabajan sobre entornos web.

⁴Página oficial de la librería PhysicsJS (<http://wellcaffeinated.net/PhysicsJS>).

⁵El manual de uso de Box2D se puede encontrar en [12]. La documentación del proyecto se puede encontrar en (<http://www.box2dflash.org/docs/2.1a/reference/>).

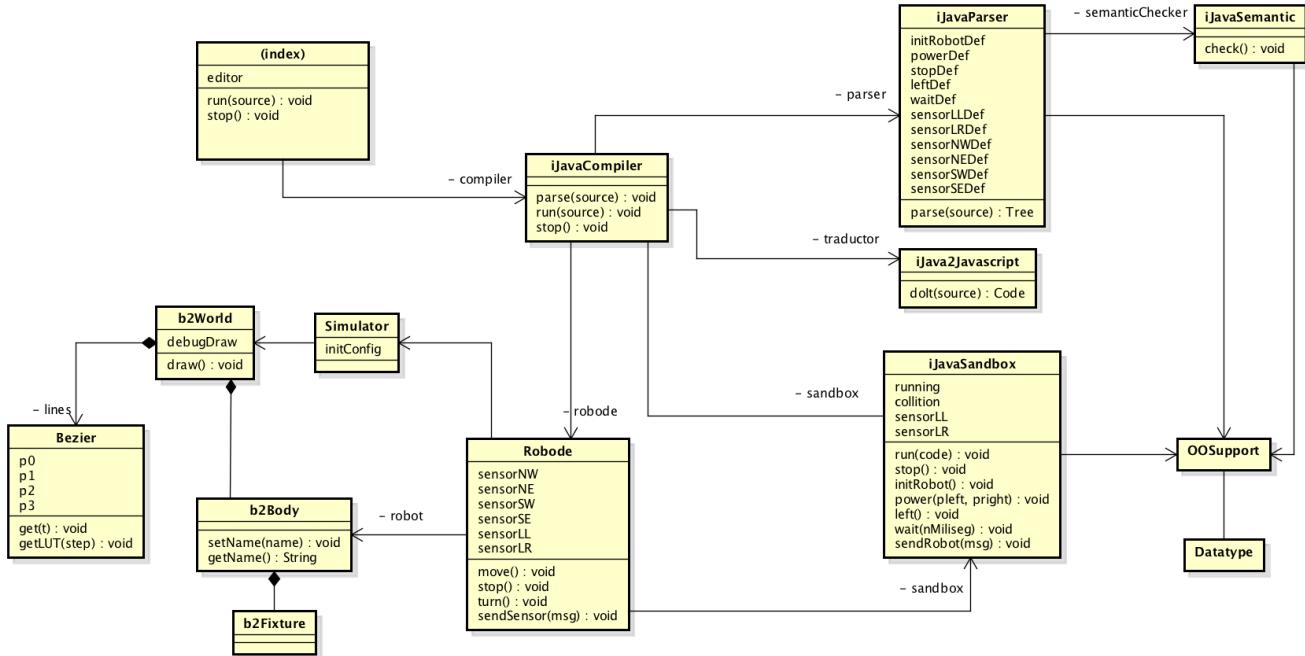


Figura 4.4: Diagrama de clases de la aplicación después de integrar el simulador con Descubre.

PhysicsJS es una librería escrita enteramente en Javascript por Jasper Palfree que aún está en fase beta. Por otra parte, Box2D es una famosa librería implementada originalmente para Flash⁶ con una gran comunidad de programadores utilizando dicha librería, lo cual siempre es de ayuda para resolver dudas y solucionar problemas.

Box2D ha sido portada a Javascript dos veces. Una de ellas por Yayushi Ando y se llama Box2Djs⁷ pero el proyecto no ha sido continuado y está desactualizado. El otro intento de llevar Box2D a Javascript ha culminado en Box2Dweb⁸. Como se puede leer en [12, capítulo 13], Box2D usa una serie de aproximaciones para representar las colisiones de manera eficiente. Principalmente, son dos las limitaciones que podría afectar al simulador del robot: (a) existe un desliz de aproximadamente 0.5 cm entre la forma que tiene un cuerpo y la colisión del mismo y (b) Box2D no maneja correctamente los cuerpos cóncavos.

Al final, se ha establecido que la librería a utilizar será Box2Dweb. En cuanto a las limitaciones que puedan afectar a Robode, estas no son un gran problema a la solución del simulador puesto que: el desliz producido en limitación (a) es muy

⁶Página oficial de la librería Box2D para Flash (<http://www.box2dflash.org>).

⁷Página oficial de Box2Djs (<http://box2d-js.sourceforge.net>).

⁸El repositorio de Box2Dweb está alojado en GitHub (<https://github.com/hecht-software/box2dweb>).

pequeño y en el caso del simulador, imperceptible. Con respecto a la limitación (b), en el caso de que se quiera construir cuerpos cóncavos, se puede solucionar creando más de un cuerpo convexo que cree una forma *global* cóncava.

Una vez que se ha aclarado que la librería utilizada es Box2Dweb, por simplicidad, a partir de ahora nos referiremos a ésta simplemente como Box2D.

4.3.1. Definición del Mundo

En Box2D existen una serie de objetos que representan al mundo y los diferentes elementos que lo componen. Estos son: el Mundo (World), los Cuerpos (Bodies), los Accesorios (Fixtures) y las Articulaciones o elementos constrictores (Joints).

El núcleo de Box2D es el objeto *World*, el cual define los parámetros básicos que regirán la simulación más tarde, como puede ser el ratio de actualización o la gravedad. También crea y almacena el resto de objetos de Box2D, como los *Bodies*, las *Fixtures* y los *Joints*, entre otros.

Para la creación de nuestro mundo, hemos decidido que tendrá una vista *top-down*, es decir, una vista del mundo desde arriba, desde el cielo. De esta manera se consigue una vista superior de todo el circuito simplificando la comprensión del movimiento del robot sobre el circuito.

Una de las características más importantes de Box2D es la posibilidad de dotar a la aplicación que se está creando de un efecto gravitatorio de manera nativa al motor. Lo normal sería establecer éste valor con un vector de dirección hacia el suelo (hacia la parte baja de la pantalla) con la fuerza que se quiera simular. Esto provocaría que los cuerpos del mundo estuvieran constantemente sometidos a una fuerza que los mueve en dirección a ese vector, normalmente hacia el suelo.

En el caso de Robode, al tener una vista superior del circuito, establecer una gravedad provocaría que el robot y el resto de cuerpos del circuito se movieran de forma anómala. Por esto, es necesario crear un mundo sin gravedad. En el listado del código 4.3 podemos ver como se ha establecido la gravedad del mundo a un vector con valor 0 en ambos ejes de coordenadas.

```

1 Simulator.World = new b2World(
2     new b2Vec2(0, 0), //gravity
3     true //allow sleep
4 );

```

Listado de código 4.3: Definición del objeto *World* en Box2D con gravedad 0 y permitiendo que los cuerpos sean capaces de dormir.

Otra característica importante en Box2D y que tiene un gran impacto en el rendimiento de la simulación es que los cuerpos puedan *dormir*. Un cuerpo *dormirá* cuando éste esté inactivo, es decir, cuando no esté siendo sometido a ninguna fuerza o so-

bre el se aplican fuerzas, pero éstas no son capaces de alterar su estado (posición, ángulo de giro, etc)⁹. En la línea 3 del listado de códigos 4.3 se puede ver como se ha establecido que los cuerpos del mundo sean capaces de dormir. Así, se consigue que el motor no simule los cuerpos que estén inactivos, mejorando el rendimiento de la aplicación.

Por último, queda definir el *paso del tiempo* en Box2D. Para ello, es necesario ejecutar la función `Simulator.World.Step()` cada vez que queramos que se ejecute un *paso* en nuestro mundo (esto es equivalente a un *tic* de reloj). Esta función ordenará al motor que aplique todas las fuerzas necesarias sobre los cuerpos del mundo (gravedad, colisiones, deslizamientos, etc). Obviamente, es en este momento cuando se ejecutará la detección de colisiones por parte del motor de físicas. Justo después de ejecutar un paso se deben eliminar las fuerzas que afectan a los objetos para que se produzca un movimiento y efecto natural de las distintas fuerzas en los cuerpos. Esto se consigue con la función `Simulator.World.ClearForces()`.

En la figura 4.5 se muestra una representación de lo que ocurre en el motor de Box2D cada vez que se ejecuta un *step*. En el código 4.4 se puede ver con más detalle las funciones básicas que se deben ejecutar para que se produzca una actualización del mundo en Box2D.

```

1 // ...
2 Simulator.World.Step(
3     1 / 60, //frame-rate
4     10, //velocity iterations
5     10 //position iterations
6 );
7
8 Simulator.World.ClearForces();
9
10 // ...

```

Listado de código 4.4: Actualización del mundo en Box2D.

En el apéndice G se analizará en detalle las características básicas y más importantes de los objetos `Body`, `Fixture` y `Joint` de Box2D para entender como se crean elementos en Box2D. Por último, en la sección siguiente se explicará como se ha construido el robot y sus diferentes partes.

⁹Esto ocurre generalmente cuando un objeto está sobre una superficie cuya forma y la del cuerpo hace que éste se mantenga quieto. Un ejemplo de esto puede ser un cuadrado sobre una superficie plana y horizontal con una gravedad hacia el suelo.

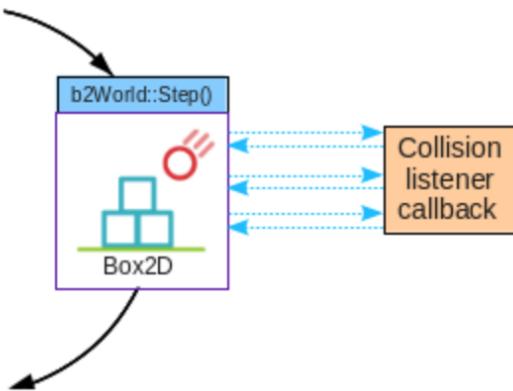


Figura 4.5: Representación de lo que ocurre cada vez que se realiza la llamada a la función `Step()` en Box2D. Obtenido de [9].

4.3.2. Construcción del robot

Para la creación del robot, un referente importante ha sido el robot de Moway. La forma y disposición de los sensores es similar a la simulada en Robode. Otro objetivo a cumplir era el de crear una simulación que imitara un comportamiento natural del robot a la hora de moverse, de chocar o de detenerse. Por tanto, se han establecido diferentes propiedades en los cuerpos para cumplir este objetivo.

Anteriormente ya se ha mencionado que el robot tendrá 2 ruedas, 2 sensores inferiores (que se usarán para detectar líneas) y otros 4 que detectan colisiones. Todos estos elementos del robot están modelados en cuerpos separados. No obstante, será necesario que se mantengan unidos y se muevan a la vez. En la figura 4.6 se puede ver un esqueleto del robot con las partes diferenciadas y los elementos que unen a las diferentes partes (en azul).

La creación de cada una de las partes se explicará en las secciones siguientes.

Cuerpo principal

El cuerpo principal está modelado como un rectángulo. Éste será más largo que ancho y tendrá el resto de cuerpos (sensores y ruedas) atados a él con joints. El cuerpo principal de Robode será dinámico, así como los sensores y ruedas.

La forma se ha creado creando una fixture con forma poligonal. Se ha utilizado la función `SetAsBox()` que permite definir el ancho y alto del polígono, creándolo con forma rectangular. El ancho del robot será de 6 píxeles mientras que su largo medirá 10 píxeles.

En el listado de código 4.5 se puede ver el código que construye el cuerpo principal de Robode.

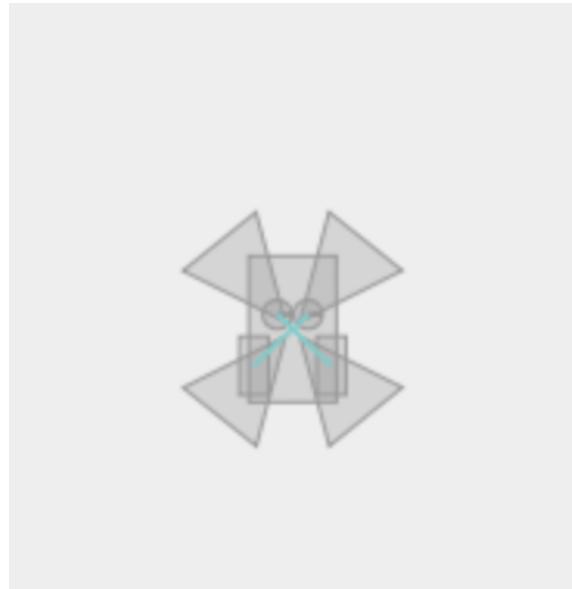


Figura 4.6: Esqueleto del robot que muestra los diferentes *bodies* y *joints* que lo forman.

```

1 // Body definition
2 var bodyDef = new b2BodyDef();
  bodyDef.type = b2Body.b2_dynamicBody;
4 bodyDef.position.Set(posX, posY);
  bodyDef.linearDamping = 8;
6 bodyDef.angularDamping = 8;

8 // Fixture definition
9 var fixDef = new b2FixtureDef();
10 fixDef.density = 40;
  fixDef.friction = 1;
12 fixDef.restitution = 0;
// Polygon Shape as a box
14 fixDef.shape = new b2PolygonShape();
  fixDef.shape.SetAsBox(width, height);

16 // Create BODY robot
18 var robot = Simulator.World.CreateBody(bodyDef);
  robot.setName("robot");
20 // Add fixture to body

```

```
22 robot.CreateFixture(fixtureDef);
```

Listado de código 4.5: Creación del cuerpo principal del robot utilizando la librería Box2dweb.

Ruedas

Las ruedas de Robode se simularán con dos polígonos con forma rectangular (método `SetAsBox()` de la clase `b2PolygonShape`). Su posición está retrasada con respecto al centro del cuerpo principal y su ancho es de 2 píxeles mientras que su alto mide 4 píxeles. En el listado de código 4.6 se puede ver como se han construido las ruedas. Como se puede ver, se ha mantenido los valores de densidad, fricción y restitución del cuerpo principal, reduciendo así la posibilidad de comportamientos anómalos en colisiones.

```
1 function createWheel(x, y) {
2
3     var bodyDef = new b2BodyDef();
4     bodyDef.type = b2Body.b2_dynamicBody;
5     bodyDef.position.Set(x, y);
6
7     var fixDef = new b2FixtureDef();
8     fixDef.density = 40;
9     fixDef.friction = 1;
10    fixDef.restitution = 0;
11    fixDef.isSensor = false;
12
13    fixDef.shape = new b2PolygonShape();
14    fixDef.shape.SetAsBox(0.2, 0.4);
15
16    var wheelBody = Simulator.World.CreateBody(bodyDef);
17    wheelBody.setName("wheel");
18    wheelBody.CreateFixture(fixDef);
19    return wheelBody;
20}
```

Listado de código 4.6: Función que crea las ruedas de Robode.

Además, se creará un joint que mantendrá unidas las ruedas con el cuerpo principal de Robode. En el listado de código 4.7 se puede ver la creación de un del joint que une el coche con la rueda.

```
1 function addWheelJoint(mybody, mywheel) {
2
```

```

4   var revoluteJointDef = new b2RevoluteJointDef();
5   revoluteJointDef.Initialize(mybody, mywheel, mywheel.
6     GetWorldCenter());
7
8   revoluteJointDef.enableMotor = true;
9   revoluteJointDef.motorSpeed = 0;
10  revoluteJointDef.maxMotorTorque = Number.MAX_SAFE_INTEGER;
11  revoluteJointDef.enableLimit = true;
12
13  return Simulator.World.CreateJoint(revoluteJointDef);
14 }
```

Listado de código 4.7: Función que crea el joint que une la rueda al coche.

La función `Initialize()` recibe como parámetros los dos cuerpos que van a ser unidos por el joint y el punto en el que se realizará el ancla (para eso se utilizará la función `GetWorldCenter()` de la clase `Body` para obtener el punto del mundo en el que se encuentra un cuerpo).

Se activará la propiedad de motor en las ruedas y se le dará un valor inicial de 0. También se establecerá el valor máximo de torsión del joint para evitar que ante una fuerza de giro muy grande, el joint se mueva de su sitio. Por último, se activará un límite para que las ruedas no se muevan de su sitio ya que es esta propiedad la que define si el ancla será fija o permitirá movimiento con respecto a la separación de los dos cuerpos.

El movimiento del robot se simulará con el movimiento que se producirá en las ruedas. Este proceso se explicará en la sección 4.3.3.

Sensores

Los sensores de colisión tendrán forma de triángulo y los sensores inferiores forma de círculo. Más adelante se describirá como se les ha otorgado la forma a cada uno del sensor. Por otra parte, todos los sensores serán cuerpos dinámicos.

Box2D provee de manera nativa que los cuerpos (en concreto, la *fixture* que contiene) se definan como un *sensor*. Esto otorga una serie de características que hacen que los sensores de Robode puedan percibir cuerpos con los que colisionan. También, el comportamiento de estos *fixtures* cambiará y dejarán de colisionar con el resto de objetos del mundo, pero si *sintiendo* la colisión que se produciría.

Este cambio de comportamiento que se ha mencionado viene de dos direcciones: el motor de Box2D que deja de calcular colisiones para estos elementos y por la captura de la colisión por parte del simulador. Box2D dejará de sentir las fixtures que sean sensores de manera automática. Pero para poder controlar la colisión, es necesario utilizar la clase `b2CollisionListener` que capturará los eventos y los

pondrá a disposición del programador para manejarlos. Este tema se tratará en la sección 4.3.4.

Para definir un cuerpo en Box2D como sensor, se tendrá que establecer la propiedad `isSensor` a `true` cuando se construya la *fixture* del cuerpo que lo contendrá. En el listado del código 4.8 se puede ver como se crean los sensores. La forma que se les ha otorgado a los sensores se explicará en la sección posterior.

```

1  function Sensor(points, name, bodyAttached) {
2
3      // ...
4
5      //create our sensor
6      var bodyDef = new Simulator.Env.b2BodyDef();
7      bodyDef.type = Simulator.Env.b2Body.b2_dynamicBody;
8      bodyDef.position.Set(positionIni.x, positionIni.y);
9      bodyDef.setName(this.name);
10     this.body = Simulator.World.CreateBody(bodyDef);
11
12    var fixDef = new Simulator.Env.b2FixtureDef();
13    fixDef.isSensor = true;
14
15    // define the shape
16    // points variable contains the point's shape
17    if (points.length == 1) {
18        // just 1 point -> line sensor
19        fixDef.shape = new Simulator.Env.b2CircleShape(0.2);
20    } else {
21        // collision sensor
22        fixDef.shape = new Simulator.Env.b2PolygonShape();
23        var vPoints = [];
24        points.forEach(function(elem, index, array) {
25            vPoints[index] = new Simulator.Env.b2Vec2(elem.x, elem.y);
26        });
27        fixDef.shape.SetAsArray(vPoints, vPoints.length);
28    }
29
30    this.body.CreateFixture(fixDef);
31
32    // ...
33}

```

Listado de código 4.8: Función que construye un sensor de Robode.

En cuanto a la unión de los sensores al cuerpo principal, se vuelven a usar joints

de tipo Revolute para mantenerlos a la misma distancia, al igual que cuando se crearon las ruedas en la sección anterior. La diferencia aquí radica en que no activaremos un motor para los sensores y que desactivaremos la colisión que se crea con el cuerpo, permitiendo que *convivan* en el mismo espacio los sensores y los cuerpos.

Otra modificación con respecto al joint de las ruedas es que solo en los sensores de colisión, la propiedad `enableLimit` debe ser establecida a `false`. Si no se hiciera esto, el joint de los sensores de colisión bloquearía el movimiento del joint de las ruedas, impidiendo su movimiento y el coche no se movería.

Una vez se ha visto como se crean los sensores, se ve que hay muchas propiedades que afectan a la gestión de colisiones. Esto se verá más en profundidad en las secciones correspondientes a cada tipo de sensor.

Dando forma a los sensores

Los 4 sensores de colisión tendrán forma de triángulo y estarán dispuestos de forma que sobresalgan del cuerpo principal con la base del triángulo hacia el exterior, para así poder detectar una colisión antes de que ocurra. Las posiciones de cada uno de los 4 sensores se han descrito con las posiciones cardinales: noroeste, sureste, noreste y sureste. De esta manera, el robot tendrá 2 sensores que detectan colisiones en la parte delantera y otros dos en la trasera.

Los 2 sensores que detectan líneas tendrán forma de círculo simulando dos cámaras que miran hacia abajo, igual que ocurre en el robot Moway. Para dotar de forma a los sensores que detectan líneas, se puede utilizar la clase `b2CircleShape` y con un radio de 2 píxeles, se asigna la forma al *fixture* correspondiente.

No obstante, la forma de los sensores de colisión, al ser triangular, requiere que se definan los vértices del polígono previamente. Para realizar esto, utilizaremos la función `fixDef.shape.SetAsArray()`¹⁰ que recibe como parámetro los vértices del polígono a crear y el número de vértices del mismo. Obviamente, los vértices tendrán que formar un triángulo con la base hacia el exterior de Robode.

Lo último que hay que saber al respecto es que los puntos de los vértices tienen que pasarse a la función `fixDef.shape.SetAsArray()` con respecto al punto origen del cuerpo. En el código 4.9 se muestran los puntos de los vértices del sensor en la posición noreste. El resto de sensores tienen la misma longitud pero solo cambia el eje de coordenadas en el que se mueve.

```

1 var pointsTR = [
2   {
3     x: 0.1,
```

¹⁰Hay que tener en cuenta que, como ya se ha mencionado en la sección 4.3, las formas cóncavas no son soportadas por Box2D.

```

5     y: -0.1
}, {
6     x: 0.5,
7     y: -1.6
}, {
8     x: 1.5,
9     y: -0.8
10 }
11 ];

```

Listado de código 4.9: Vértices que forman el triángulo del sensor noreste de Robode.

4.3.3. Moviendo a Robode

El movimiento de Robode está basado en el movimiento de las ruedas. Al igual que ocurre en un coche, son las ruedas de Robode las que tiran del resto del cuerpo. Por tanto, serán las ruedas las que realicen y controlen el movimiento. Además, la fuerza que ejerza cada rueda es independiente.

Para mover una rueda, hay que aplicar una fuerza con respecto al vector que apunta en la dirección frontal a la propia rueda. Esto se hace utilizando la función `ApplyForce()` que recibe como parámetro el vector con la fuerza a realizar y el punto desde el que se realiza. El punto que se usará para aplicar la fuerza será el centro de la propia rueda. Una fuerza con valor positivo hará que el robot se mueva hacia delante. Mientras que una fuera negativa hará que este se mueva hacia atrás.

La fuerza que se aplicará a la rueda se mide en Newtons y la potencia máxima y mínima establecida será de 4K y -4K Newtons, respectivamente. Por defecto, la velocidad aplicada al motor se multiplicará por 100, siempre y cuando no se mantenga en los máximos y mínimos ya mencionados.

De esta manera se consigue que la rueda ejerza una fuerza hacia delante (o atrás) y el robot se desplace. Pero esto ocasionará tres problemas:

1. La velocidad angular de las ruedas cambiará y será diferente a la de cuerpo principal.
2. La velocidad angular de los sensores que detectan colisiones también se diferenciará a la del cuerpo principal.
3. Se creará una velocidad lateral (u ortogonal) que hará que el movimiento del robot no sea recto.

Los dos primeros problemas se resuelven fácilmente igualando la velocidad angular del cuerpo principal y las ruedas o sensores. Esto se hace con la función

`SetAngularVelocity()` de la clase `b2Body`. De esta manera se consigue que las ruedas y el robot la misma velocidad angular, impidiendo giros. Además, en el caso de los sensores, también es necesario ajustar el ángulo de giro de los mismos para forzar a que se mantengan en la misma posición con respecto al robot.

Para eliminar la velocidad lateral que se produce al mover el robot, basta con aplicar una velocidad lineal que contrarreste esta fuerza ortogonal, con el método `SetLinearVelocity()` de la clase `b2Body`.

Ahora ya se tiene un robot desplazándose por el mundo. Para conseguir que sea de manera realista, se dotará a las ruedas de una fricción fuerte (valor de 1) y al cuerpo principal un valor de deslizamiento lineal y angular de 8 (propiedades `linearDamping` y `angularDamping` de `b2Body`).

4.3.4. Detectando colisiones

La clase `b2Contact` de Box2D representa el contacto entre dos cuerpos. Esta contiene los dos cuerpos que se están *tocando* y otra información de interés. Box2D proporciona la clase `b2ContactListener` que permite la redefinición de 4 funciones clave en la gestión de colisiones. Estas son:

- Función `PreSolve()`, que se ejecutará antes de que se produzca por parte del motor de Box2D la solución de la colisión. Es útil para realizar cálculos previos al contacto. Útil también para anular un contacto y que la colisión no se produzca, por ejemplo.
- Función `PostSolve()`, similar a la anterior pero se ejecutará justo después de que la gestión de la colisión haya acabado.
- Función `BeginContact()` que permite controlar cuando un contacto empieza, es decir, cuando un cuerpo colisiona con otro.
- Función `EndContact()` que se ejecutará cuando un contacto termine, es decir, cuando un cuerpo deje de *tocar* a otro.

Es importante que estas funciones estén lo más optimizadas posible puesto que se ejecutarán cada vez que se produzca un contacto entre dos cuerpos. En el caso del simulador, solo necesitaremos redefinir las funciones `BeginContact()` y `EndContact`. Estas recibirán como parámetro un objeto `b2Contact` para poder obtener información del contacto que se está produciendo.

En el caso de Robode, puesto que se están simulando sensores muy simples que no distinguen entre detectar uno o más cuerpos, las funciones `BeginContact()` y `EndContact` simplemente contarán el número de cuerpos que está sintiendo ese sensor. Por tanto, al comenzar un contacto únicamente se notificará dicha colisión si

el sensor no estaba sintiendo nada previamente. De igual manera, cuando un contacto acaba, solo se notificará si el número de cuerpos sentidos por el sensor es 1 (que va a cambiar al 0).

En el listado de código 4.10 se muestra a modo ilustrativo la función `BeginContact()`. La función `EndContact()` sería similar pero decrementando el contador de colisiones.

```
1 contactListener.BeginContact = function(contact) {  
2  
3     if (!running) return;  
4  
5     var isSensorA = contact.GetFixtureA().IsSensor();  
6     var isSensorB = contact.GetFixtureB().IsSensor();  
7  
8     if (isSensorA != isSensorB) { // a XOR b: is any fixture a  
9         sensor?  
10        //find who is the sensor and who the body  
11        var bodySensed, bodySensor;  
12  
13        if (isSensorA) {  
14            bodySensor = contact.GetFixtureA();  
15            bodySensed = contact.GetFixtureB();  
16        } else {  
17            bodySensor = contact.GetFixtureB();  
18            bodySensed = contact.GetFixtureA();  
19        }  
20  
21        // if it's a robot part, do nothing  
22        if (isRobotPart(bodySensed.getBodyName(), robotparts))  
23            return;  
24  
25        /* Here, we got a collision sensor-body.  
26            if is the first collision detect by this sensor  
27            then send message to sandbox */  
28        if (bodySensor.nCollided === 0) {  
29            // notify new collision  
30            var message = {  
31                id: bodySensor.getBodyName(),  
32                state: "begin"  
33            };  
34            sendMessage("sensor", message);  
35        }  
36        //increment the collision count
```

```

36     bodySensor.addCollision();
37 }
38 }
```

Listado de código 4.10: Función BeginContact () definida para los sensores.

Como se puede ver en la linea 21, la comprobación acabará si el cuerpo percibido es una parte del coche, como una rueda o el propio cuerpo principal. La forma de comprobar si un cuerpo es parte del robot es con el nombre del cuerpo. Este se ha establecido en la creación de los cuerpos con la función setName () de la clase Body.

También, se puede ver como en las lineas 28 a 33 se notifica la colisión enviando un mensaje. En secciones posteriores se explicará el proceso de notificación y la comunicación del simulador con el resto de la aplicación.

4.3.5. Detectando lineas

Antes de poder hablar de como se detectan lineas, es necesario hablar de las propias lineas.

Una característica del simulador que estaba clara desde el comienzo del proyecto era la necesidad de que el robot pudiera seguir lineas. Para poder seguir lineas, es necesario poder calcular la distancia del robot a una linea. Esto implica el cálculo de la distancia de un punto a una linea. Pero, ¿qué forma tiene la linea?

Hay dos opciones para la forma de las lineas: líneas rectas o curvas. Las lineas rectas darían una forma muy simple al circuito y no se corresponden con los circuitos que podrían crearse en la realidad. Además, la programación del robot sigue-líneas sería muy simple para el niño (solo tendría que asegurarse de que cuando encuentra una linea, no se tuerce). Así que esto elimina la opción de usar rectas para modelar las líneas.

Curvas de Bezier

Entonces, una curva. Existen principalmente dos tipos de curvas usadas en computación gráfica: curvas de Catmull-Rom y curvas de Bezier.

Las curvas de Carmull-Rom¹¹ se definen con una serie de puntos de control o *splines* que definen la forma de la curva en base a la interpolación de los puntos que la forman. La curva pasará por todos los puntos que la forman y es posible añadir más puntos *sobre la marcha* para alargar la curva. Un ejemplo de curva de Catmull-Rom se puede ver en la figura 4.7.

El otro tipo de curva son las curvas de Bezier[43]. Las curvas de Bezier se definen por 2 puntos, el origen y el extremo de la curva, y por al menos un punto más

¹¹Para más información sobre las curvas de Catmull-Rom se puede consultar [20] o [54].

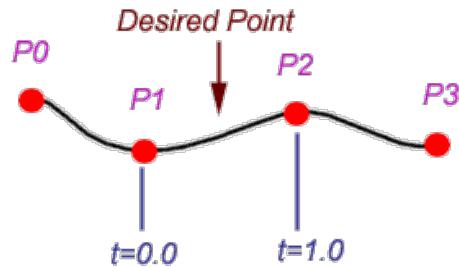


Figura 4.7: Ejemplo de curva de Catmull-Rom con 4 puntos que definen su forma. Obtenido de [20].

de control. Las curvas de Bezier con solo un punto de control se denominan curvas cuadráticas. Si tiene 2 puntos de control, se denomina curva de Bezier cúbica. Un ejemplo de curvas de Bezier cuadrática y cúbica se puede ver en las figuras 4.8 y 4.9, respectivamente.



Figura 4.8: Ejemplo de curvas de Bezier cuadrática. Obtenido de <http://pomax.github.io/bezierjs>.

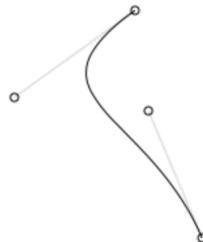


Figura 4.9: Ejemplo de curvas de Bezier cúbica. Obtenido de <http://pomax.github.io/bezierjs>.

Las curvas de bezier se definen en función de t , que se mueve en un intervalo de 0 a 1. Siendo $t=0$ el inicio de la curva, $t=1$ el final y $t=0.5$ el punto intermedio. Una linea recta también puede ser representada mediante curvas de Bezier siempre que los puntos de control estén sobre la linea. No obstante, estas curvas tienen una limitación:

no se puede representar un circunferencia con una (única) curva de Bezier. Esto es algo que no afecta a nuestro simulador.

Finalmente, para el modelado de líneas se ha decidido usar las curvas de Bezier, acompañado de la librería BezierJS¹² para Javascript. Esta librería ofrece una serie de funciones muy útiles para crear y manejar funciones. En el listado de código 4.11 se puede ver un ejemplo de las funciones más importantes de la librería.

```

1 var curve = new Bezier (150,40 , 80,30 , 105,150);
2
3 var mid = curve.get(0.5);
4
5 var steps = 10;
6 var points = curve.getLUT(steps);
```

Listado de código 4.11: Ejemplo de uso de las funciones de la librería BezierJS.

En la linea 1 del listado 4.11 se crea una curva cuadrática con los puntos (150, 40), (80, 30) y (105, 150). Concretamente, esta curva toma la forma de la curva de la figura 4.8. El segundo punto es el punto de control. Una curva cubica se definiría de forma similar, pero pasando como argumento un punto de control más.

En la linea 3 del listado 4.11 se obtiene el punto que está justo en la mitad de la curva. La función `getLUT()` devuelve un array de puntos equidistantes entre ellos dentro de la curva. El número de puntos es el valor de `steps` más 1.

La función `getLUT()` de la librería BezierJS será la clave para la detección de líneas en el simulador.

Detectando curvas de Bezier

Ahora que ya se ha aclarado que se usarán curvas de Bezier para representar las líneas en el simulador, solo queda saber la distancia de un punto (el sensor) al punto más cercano de la curva de Bezier. Este problema consiste en encontrar la proyección de un punto sobre una curva. Dicho problema ya ha sido estudiado anteriormente. Por ejemplo J. Ros en [48] estudia de manera matemática como resolver este problema.

Teniendo esto en cuenta, ésta fue la primera aproximación al problema que se realizó. Los resultados no fueron buenos: el punto en la curva no era exacto y con un coste computacional alto.

En el segundo intento de resolución, se utilizaba la función `getLUT()` de BezierJS para partir la curva en muchas líneas rectas y crear un cuerpo en Box2D con muchas `fixtures`¹³. De esta manera se conseguía formar una curva a partir de muchas líneas

¹²Página oficial del proyecto BezierJS (<http://pomax.github.io/bezierjs/>).

¹³Para crear la forma de línea recta en Box2D se utilizaba la clase `b2PolygonShape` estableciendo la forma con la función `SetAsEdge()` que recibe dos puntos y crea una línea entre ellos.

rectas. La detección de curvas, entonces, se realizaba igual que para los obstáculos (sección 4.3.4).

Esto solucionaba el problema, pero la curva tenía una apariencia muy fina y se obtenía un coste alto al mantener una cantidad tan grande de cuerpos pequeños. Aunque se simplificara la curva con un valor de `step` suficientemente pequeño en la función `getLUT()`, no se solucionaba el problema de la linea fina.

La tercera aproximación al problema dio resultado. Era volver a proyectar el punto (sensor) sobre la curva, pero aproximando los puntos de ésta con la función `getLUT()`. De esta manera, se obtenía una cantidad razonable de puntos en la curva y se calculaba la distancia del sensor a cada uno de estos, resultando el problema en el cálculo de distancia entre dos puntos. Algo computacionalmente menos costoso que calcular la proyección de un punto sobre una curva.

En el listado de código 4.12 se puede ver el código que detecta y notifica una *colisión* con una linea.

```

1 function detectLineCollision(sensor) {
2   var lines = Simulator.World.lines;
3   var anysensed = false;
4
5   // ...
6
7   var minDistance = Simulator.World.getDistanceCollitionLine();
8
9   lines.forEach(function(curve) {
10     var p = pointInBezier(curve, SensorPosition);
11     // if any sensor collided, notify!
12     var dist = distance(p, position);
13     if (dist <= minDistance) {
14       //notify the collision
15       var message = {
16         id: sensor.name,
17         state: "begin"
18       };
19       sendMessage("sensor", message);
20       anysensed = true;
21       return;
22     }
23   });
24   //if it detects no collitions, notify end collision
25   if (!anySensed) {
26     var message = {

```

```

28     id: sensor.name,
29     state: "end"
30   } ;
31   sendMessage("sensor", message);
32 }
}

```

Listado de código 4.12: Función que detecta líneas (curvas de Bezier).

El valor de distancia mínima (variable `minDistance`) se define con el grosor de la linea más el radio del sensor.

La diferencia de esta última solución a la anterior es que en ningún momento se está creando la curva como objeto en Box2D. Por tanto, se ha modificado el prototipo de `World` para que almacene las líneas (variable `lines`). No obstante, siguen sin ser representadas con un objeto `Body` para ahorrar coste y para facilitar el poder pintar distinto grosor de linea según convenga (por ejemplo, si se hace zoom en el circuito).

Con respecto al dibujo de las líneas, se explicará más adelante en la sección 4.3.6.

4.3.6. Animando a Robode

Para entender como se ha realizado la animación del simulador, primero es necesario conocer algunos conceptos de Javascript.

Javascript proporciona una serie de funciones para gestionar timers en la ejecución de un programa. Estos son:

- `setTimeout(function, delay)`: Ejecuta la función `function` después de un tiempo especificado por `delay`. Devuelve un `id` que identifica al timer.
- `setInterval(function, delay)`: Ejecuta la función `function` de manera periódicamente cada vez que se cumple el `delay`. Devuelve un `id` que identifica al timer.
- `clearTimeout(id)` y `clearInterval(id)`: funciones que eliminan el timer asociado a ese `id`.

El uso normal que se haría en la aplicación (el simulador junto a Descubre) será ejecutar la función `setInterval(function, delay)` con un `delay` indicando cada cuanto tiempo se quiere ejecutar la función `function` (que en nuestro caso contendrá `World.Step()` y la actualización de los gráficos).

Por tanto, cada vez que se produzca un evento (por ejemplo, una interrupción de teclado o ratón), la ejecución de Javascript tratará dichos eventos.

No obstante, Javascript tiene una limitación: implementa un sistema *monohilo*. Esto implica que si hay una función que aún no ha terminado de ejecutarse cuando

ocurre un timer o un evento, estos se verán retrasados hasta que dicha función acabe. Para más información, se puede consultar [45], [53] y [46].

Además, hay que tener en cuenta el diseño de la aplicación (descrito en la sección 4.2). Esta tendrá por una parte el *sandbox* (clase `iJavaSandbox`) y por otra el simulador y la aplicación. El *sandbox* controlará la ejecución del programa escrito en iJava y enviará instrucciones al simulador (clase `Robode`). La aplicación atenderá a los eventos de teclado y ratón junto al simulador, que moverá el robot sobre el mundo.

A parte de todo esto, el Modelo de programación que se usará en el simulador (sección 4.1) permitirá detener la recepción de instrucciones con la función `wait(n_miliseg)` (API del simulador). Esto hará que la ejecución del *sandbox* se bloquee un determinado tiempo. Pero esto no debe de ocurrir con el resto de la aplicación.

Se hace necesario separar ambas ejecuciones en dos hilos de ejecución diferentes. Un hilo ejecutaría la aplicación principal junto al simulador. El otro hilo ejecutaría el *Sandbox*, el cual podría detenerse y realizar una *espera activa* cuando se ejecute la función `wait()`. De esta manera se consigue que el simulador se ejecute y sea controlado por el *Sandbox* y las instrucciones que en éste se ejecutan.

Para conseguir esto, en Javascript existen los Web Workers[40]. Básicamente, un Web Worker permite ejecutar tareas en segundo plano separado del hilo principal. Es decir, se consigue un sistema *multihilo* en Javascript. La comunicación del Web Worker con el resto de elementos se hace mediante mensajes, como los que se han visto anteriormente en los códigos 4.10 o 4.12 y que vienen reflejados en el diagrama de clases de la figura 4.4.

Aún así, los Web Workers tienen sus limitaciones. La más importante es que no se puede interactuar con el DOM¹⁴ de HTML. Esto hace que no se pueda dibujar en el *canvas*.

Entonces, la solución radica en convertir el *Sandbox* en un Web Worker y que sea éste el que controle la ejecución del simulador mediante mensajes. De esta manera se consiguen separar el hilo de ejecución de la aplicación principal y el hilo que ejecutará las instrucciones de iJava.

En el hilo principal se ejecutará la función `setInterval()` con un valor de `delay` equivalente a 60 FPS (frames por segundo). Cada vez que se ejecute dicha función, se actualizarán las fuerzas del mundo, el movimiento del robot y se dibujará por pantalla el estado actual del mundo. En el código 4.13 se muestra la ejecución de la función `setInterval()`.

¹⁴El DOM (Document Object Model o Modelo de Objetos del Documento en español) es un API de programación para documentos de HTML. Parte fundamental de HTML y que contiene toda la información relacionada a la página web

```

1 idInterval = window.setInterval(function() {
2     Simulator.World.Step(
3         1 / 60, //frame-rate
4         10, //velocity iterations
5         10 //position iterations
6     );
7
8     // Simulator.World.drawLines();
9     Simulator.World.DrawDebugData();
10    Simulator.World.ClearForces();
11
12    updateMovement();
13}, 1000 / 60);

```

Listado de código 4.13: Función `setInterval()` que se ejecutara 60 veces por segundo.

En el hilo que ocupa el *Sandbox*, se ejecutarán las instrucciones del programa y se atenderán los mensajes que lleguen desde el compilador pidiendo que se detenga o se reanude la ejecución (con un nuevo código), por ejemplo.

Dibujando a robode

Una vez que se ha aclarado como es la actualización del mundo, queda especificar como se dibuja el mismo.

Box2D provee un elemento para dibujar los objetos del mundo en el *canvas* de HTML5, el objeto `b2DebugData`. Este pinta la forma de los cuerpos sobre el *canvas* automáticamente.

Pero como ya se ha mencionado anteriormente, las lineas no han sido representadas como elementos de Box2D, por tanto, se usarán las funciones nativas de *canvas* para realizar esto. En el código 4.14 se puede ver el código que dibuja lineas implementado en el prototipo de la clase `World`.

```

1 Simulator.Env.b2World.prototype.drawLines = function() {
2     var ctx = this.m_debugDraw.m_ctx;
3     var scale = this.getWorldScale();
4     ctx.save(); //save the current context
5     ctx.lineWidth = this.lineThickness * scale;
6     ctx.strokeStyle = "black";
7     this.lines.forEach(function(l) {
8         var points = l.points;
9         ctx.beginPath();
10        ctx.moveTo(points[0].x, points[0].y);

```

```

12     if (points.length > 3) {
13         ctx.bezierCurveTo(points[1].x, points[1].y, points[2].x,
14                             points[2].y, points[3].x, points[3].y);
15     } else {
16         ctx.quadraticCurveTo(points[1].x, points[1].y, points[2].x,
17                               points[2].y);
18     }
19     ctx.stroke();
20 });
21 ctx.restore(); // restore the saved context
22 };

```

Listado de código 4.14: Función que dibuja las curvas de bezier en el *canvas*.

Para poder conseguir que esta función se ejecute, y ademas se pinten las lineas lo primero (para que esté en un plano inferior al robot y el resto de elementos) ha sido necesario modificar levemente la librería de Box2D. Esto era necesario porque Box2D limpia el *canvas* antes de dibujar nada (borrando las lineas si se pintan antes de llamar a esta función y dibujando las lineas sobre el robot si se llama después).

Esto se ha realizado añadiendo la llamada a la función `drawLines()` (si ésta está definida) en la implementación de la clase `b2DebugDraw`. La función `drawLines()` se ha definido como un prototipo del objeto `World`. El código 4.15 muestra esta modificación.

```

1 // Draw function
2 h.prototype.DrawDebugData = function(){
3     if(this.m_debugDraw!=null) {
4         //CLEAR THE CANVAS!
5         this.m_debugDraw.m_sprite.graphics.clear();
6         // my draw lines function (if it's defined)
7         if(this.drawLines){
8             this.drawLines();
9         }
10        // end re-definatoin
11        //(...)
12    }
13 }

```

Listado de código 4.15: Modificación de la librería de Box2D para poder dibujar de manera correcta las lineas.

Adicionalmente se ha añadido un efecto de *zoom*¹⁵ que modifica el tamaño de todos los elementos, permitiendo crear circuitos más grandes y que se vea todo in-

¹⁵Esto se ha realizado modificando la propiedad `scale` del objeto `b2DebugDraw`.

dependientemente del tamaño del *canvas*. Junto a esto, también se ha añadido un *scroll* tanto vertical como horizontal por la pantalla para facilitar la visión del circuito cuando se realiza *zoom*.

Capítulo 5

Conclusiones y vías futuras

En el presente Trabajo Fin de Grado se ha abordado la creación de un simulador de un robot para integrarlo en la plataforma Descubre la programación.

La meta de este trabajo es clara: ayudar a expandir el pensamiento computacional entre los alumnos de Educacion Primaria y Secundaria de la Región de Murcia.

Para ello, se ha construido una herramienta para aprender a programar sobre una plataforma online con recorrido en la Región de Murcia.

En la creación del simulador hay claros precedentes, como son los robot Moway y Arduino y la plataforma CodeHS. Se ha creado un robot que comparte características con Moway, que se programa en cierta forma como Arduino y en una plataforma similar a CodeHS.

Se puede decir que se han cumplido en su totalidad los objetivos que se planteaban en la sección 3. Se ha creado un robot de dos ruedas que se mueve por un mundo continuo, que tiene sensores que detectan colisiones y que ofrece la lógica para que se hagan programas sigue-lineas.

Además, se proporciona una librería que permite controlar el robot y programar el comportamiento del mismo. El Modelo de programación creado ofrece a los alumnos una nueva forma de programar, como se hace en los robots de Arduino. Así, el alumno podrá desarrollar nuevos tipos de programas, enfrentándose a retos nuevos.

No obstante, este trabajo planteaba cuatro grandes dificultades a resolver: (a) la comprensión y modificación de la plataforma Descubre y su compilador; (b) la representación de lineas mediante curvas de Bezier y la detección de las mismas por parte del robot; (c) la creación de un modelo de programación diferente al que proporciona la plataforma Descubre y (d) la creación del simulador que modelará el mundo donde se encuentra el robot y, por supuesto, su comportamiento.

La plataforma Descubre la programación permite programar y ejecutar código iJava. Por tanto, es necesario comprender su funcionamiento y ser capaz de ampliar la librería de funciones para poder controlar al robot. También, con la creación de un modelo de programación, se ha modificado Descubre para poder soportar este tipo

de programas. Esto se ha realizado mediante la creación de un sistema *multihilo* en iJava que ha permitido el correcto funcionamiento de la plataforma y del simulador.

En cuanto a la simulación del robot, se ha usado una librería física, Box2D, que realizará el trabajo de calcular y resolver las colisiones que se produzcan en el mundo. También nos permite dibujar el mundo sobre el canvas de HTML5. La creación y configuración de elementos en Box2D para que se creara un mundo realista también ha sido un punto a tener en cuenta y que se ha resuelto satisfactoriamente.

Por último, la representación de líneas mediante curvas de Bezier es un problema bastante complejo. Era necesario calcular la distancia de un punto al punto más cercano dentro de la curva. O lo que es lo mismo, la proyección de un punto sobre la curva. Esta dificultad se ha resuelto gracias a la librería BezierJS, que ha simplificado los cálculos para obtener la distancia a la curva de forma computacionalmente barata.

En comparación a los sistemas que hay actualmente, opino que se ha creado una opción bastante interesante y atractiva. Mediante la robótica, se puede programar un robot en un lenguaje imperativo y con sintaxis moderna, especial para aprender.

Si lo comparamos con proyectos como Lego Mindstorm EV3 o Arduino, Robode permite iniciarse en la robótica sin necesidad de hacer una inversión de dinero ni tener conceptos de electrónica o circuitos.

En el caso de Moway o Scratch, la programación se hace mediante bloques. Este sistema puede ser bueno como iniciación y para aprender ciertos conceptos básicos (como las condiciones y el flujo de instrucciones) pero se vuelve complejo en aplicaciones grandes.

Por otra parte, el robot Robomind simula un robot en un mundo discreto. La programación en este simulador puede ser al principio interesante, pero se termina volviendo repetitiva y con una libertad y complejidad limitada a la hora de crear programas. Robomind no deja de ser una unión entre Turtle de Logo y Karel the robot.

En cambio, con *Robode*, se puede crear una gran variedad de programas. Desde programas que solo eviten obstáculos, hasta programas que simulen movimientos complejos con un respaldo en las matemáticas y físicas de movimiento que lo apoye. Además, al estar integrado con Descubre, su uso es gratuito y libre para cualquier niño que quiera iniciarse en la programación. Esto es especialmente importante, y más si se compara con alternativas actuales como CodeHS, que requiere de una suscripción para poder usar los servicios que tiene disponible en su totalidad.

En cuanto al despliegue de la aplicación en Descubre, se ha buscado durante todo el desarrollo que el simulador introdujera los mínimos cambios posibles. El sistema principal se ha mantenido mayormente intacto. Solo sería necesario añadir a la plataforma una nueva sección para programar con el simulador y ciertos cambios estéticos.

De cara al futuro se plantean diferentes vías futuras con la finalidad de extender la

aplicación y dotarla de cierta funcionalidad que produzca una mejora en la aplicación.

Una vía futura clara es la mejora de la apariencia del simulador. Se podría dibujar el simulador con un conjunto de sprites que lo hagan más atractivo. También, una mejora útil sería la de crear un editor de circuitos que permita de manera visual y sencilla crear circuitos para luego importarlos en Descubre y que se pueda programar el comportamiento del robot en entornos diferentes.

También sería importante estudiar la creación de circuitos con obstáculos dinámicos, que se muevan por la pantalla junto al robot. De esta manera, se añadiría una connotación de videojuego al simulador. El usuario tendría que programar el comportamiento del robot de manera que cumpla una serie de objetivos (llegar a un sitio concreto sin chocar con ningún enemigo, por ejemplo). Este modelo también permitiría la creación de retos dentro de la plataforma, motivando así a los alumnos a superarse a sí mismo.

Apéndice A

Edad escolar en diferentes Sistemas Educativos

Incluso en la Unión Europea, los Sistemas Educativos difieren en la edad de escolarización de los estudiantes. Por ello, se ha confeccionado una tabla que intenta resumir la edad estándar en la que un niño está escolarizado durante las distintas etapas educativas¹. La información que se muestra a continuación está basado en [59] y [17].

Se tomarán como referencia los nombres de las etapas del Sistema Educativo Español (Educación Infantil, Primaria, Secundaria y Bachillerato) para mostrar los años que pasan los estudiantes. Con respecto a la selección de países para realizar la comparación, se escogen los más representativos en los que se han realizado los estudios sobre el aprendizaje de la programación y donde más extendido están las plataformas que se mencionan en el presente documento.

País	Infantil	Primaria	Secundaria	Bachillerato
España	0-6	6-12	12-16	16-18
Estados Unidos	3-6	6-10	10-14	14-18
Reino Unido	2-5	5-11	11-16	16-18
Alemania	0-6	6-10	10-16	16-19
Francia	2-6	6-11	11-16	16-18
Bélgica	0-2.5/3	2.5/3-6	6-12	12-18
Irlanda	4-6	6-12	12-15	15-19

Tabla A.1: Comparativa de edades de escolarización en diferentes Sistemas Educativos con respecto a las etapas del Sistema Educativo Español.

¹Se hablará de las etapas en las que el Sistema Educativo referido está bajo la responsabilidad del Ministerio de Educación (o equivalente) del propio país. Esto no excluye a la educación en centros privados.

58 APÉNDICE A. EDAD ESCOLAR EN DIFERENTES SISTEMAS EDUCATIVOS

En el caso del Sistema Educativo Americano, existen muchas vías en la formación de un niño, como bien detalla A. Corsi-Bunker en [17]. Dependiendo de si se escoge una vía privada o dependiendo del estado, los años pueden variar. Aún así, en la figura A.1 se muestra el modelo estándar (sistema K-12).².

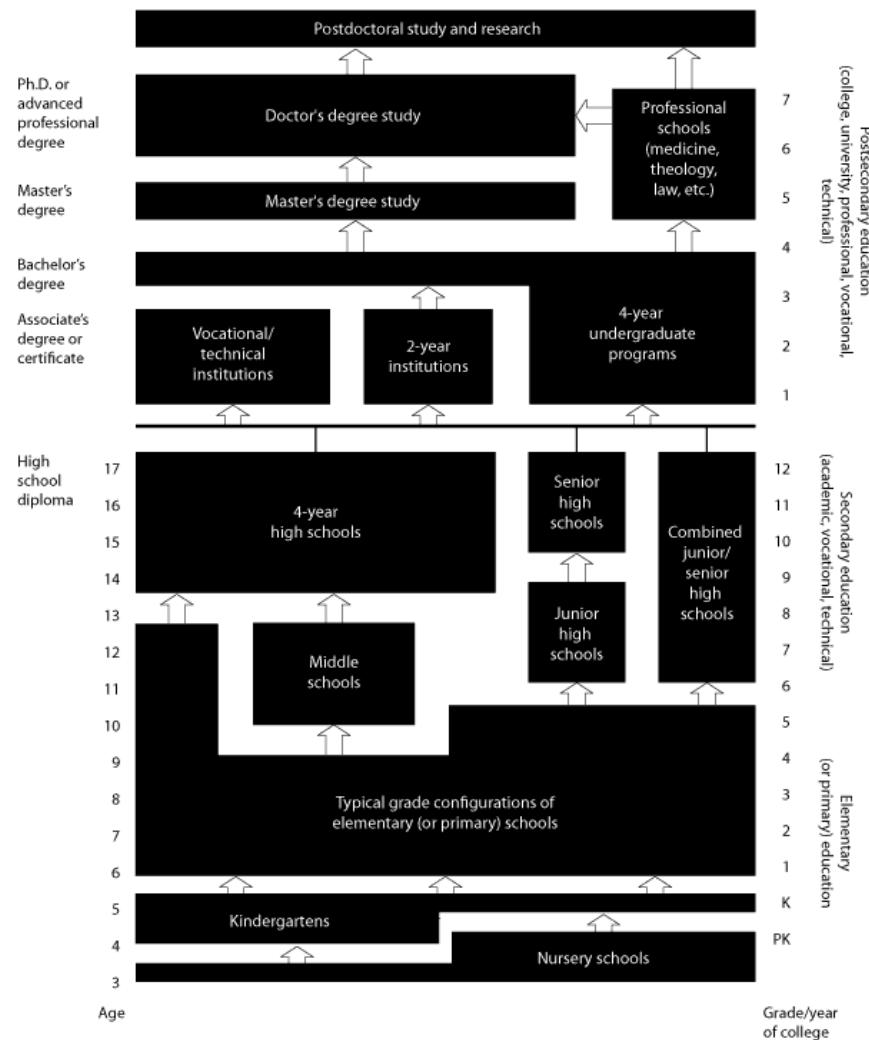


Figura A.1: Tabla que muestra los típicos patrones de progresión en el Sistema Educativo Americano. Fuente: U.S. Department of Education, National Center for Education Statistics, Annual Reports Program. Obtenido de http://nces.ed.gov/programs/digest/d11/figures/fig_01.asp

²Para más información sobre el Sistema Educativo Americano recomiendo que se acuda directamente a la fuente: U.S. Department of Education, National Center for Education Statistics (<http://nces.ed.gov>)

Apéndice B

Lenguaje Logo

A continuación se realizará una revisión de las principales características del lenguaje Logo para tener una visión general de su funcionamiento y sintaxis. Este análisis se basa en el trabajo de [23, p.274-305]. Si el lector quiere ampliar información sobre el lenguaje Logo, puede consultar [25] o [33].

words, sentences, operaciones y comandos

Hay dos tipos de datos principales: *words* y *sentences*. *words* está formado por 0 o más caracteres, sin espacios. Algunos ejemplos son: "SUN", "CAT", "5", "97!", "" (cadena vacía). El tipo de datos *sentences* se forma por un conjunto de *words* separado por espacios, como por ejemplo "HELLO WORLD" o "3 + 2 = 5". Un numero es una *word* pero compuesto únicamente por dígitos.

En el lenguaje Logo existen operadores para tratar con los tipos *words* y *sentences*. También se pueden encontrar una serie de operaciones predefinidas en el lenguaje que ofrecen funcionalidad extra, como controlar la entrada/salida de un programa, saber la fecha o manejo de *words* y *sentences*. En la tabla B.1 vemos un resumen de las operaciones elementales del lenguaje Logo, el número de argumentos que requiere y un ejemplo de entrada con su salida correspondiente.

Si se intentara ejecutar la operación WORD "CAT" "DOG", se produciría un error al ser uno de sus parámetros un tipo *sentence* y no *word* (el primer argumento contiene un espacio). De igual manera, las operaciones SUM, DIFFERENCE, MAXIMUN y MINIMUN deben recibir argumentos numéricos o se produciría un error.

También es posible encadenar varios operadores. De esta manera, la cadena de instrucciones FIRST OF BUTFIRST OF "CAT" devolverá el *word* "A".

<i>Operador</i>	N. argumentos	Entrada	Salida
FIRST	1	"CAT"	"C"
-	-	"CAT AND DOG"	"CAT"
LAST	1	"CAT"	"T"
-	-	"CAT AND DOG"	"DOG"
BUTFIRST	1	"CAT"	"AT"
-	-	"CAT AND DOG"	"AND DOG"
BUTLAST	1	"CAT"	"CA"
-	-	"CAT AND DOG"	"CAT AND"
COUNT	1	"CAT"	"3"
-	-	"CAT DOG"	"2"
WORD	2	"CAT" "DOG"	"CATDOG"
SENTENCE	2	"CAT" "DOC"	"CAT DOG"
-	-	"CAT" "DOG"	"CAT DOG"
PRINT	1	"CAT"	CAT
TIME	0	-	"5:42 PM"
DATE	0	-	"1/31/2016"
SUM	2	"-5" "5"	"0"
DIFERENCE	2	"5" "5"	"0"
MAXIMUN	2	"-5" "5"	"5"
MINIMUN	2	"3" "5"	"3"

Tabla B.1: Resumen de las operaciones elementales que ofrece el lenguaje Logo.

Variables, comentarios y definiendo nuevas operaciones

Para definir variables basta con usar el operador **MAKE** seguido de "(comilla doble), el nombre de la variable y el valor que se quiere asignar. Para hacer uso de la variable, utilizamos el símbolo :(dos puntos) seguido del nombre de la variable. Para hacer comentarios se utiliza el símbolo ;(punto y coma). Podemos ver un ejemplo en el código B.1.

```

1 MAKE "ADULTO 18
3 PRINT :ADULTO ; salida -> 18

```

Listado de código B.1: Ejemplo de definición de nuevas variables en el lenguaje Logo.

También es posible crear nuevas operaciones. Se utilizan las palabras reservadas TO y END para marcar el inicio y final de la definición, respectivamente. A continuación es necesario declarar el nombre con el que se definirá la operación que se está

creando (el cual no puede coincidir con ninguna palabra reservada en el lenguaje, como es común en la mayoría de lenguajes de programación). Es opcional la declaración de parámetros que se pasaran a la operación. Por último, y antes de la palabra END, se incluirán las instrucciones que realizará la operación.

En el código B.2 se puede ver un ejemplo de definición de nuevas operación.

```

1  TO SALUDAR
2    MAKE "SALUDO "HELLO WORLD"
3    OUTPUT SALUDO
4  END
5
6  PRINT SALUDAR ; salida -> HELLO WORLD
7
8  TO PENULTIMO :algo
9    OUTPUT FIRST OF LAST :algo
10 END
11
12 PRINT PENULTIMO OF "CAT BIRD DOG" ; salida -> BIRD
13
14 PRINT PENULTIMO OF "CAT" ; salida -> A

```

Listado de código B.2: Definición de nuevas operaciones en el lenguaje Logo.

Operadores aritméticos

A parte de los operadores SUM y DIFFERENCE antes mencionados, Logo también otorga la posibilidad de utilizar los operadores aritméticos clásicos. La única condición es que la salida generada debe ser tratada (PRINT) o almacenada (en variables, como veremos en la sección siguiente) de alguna manera. Los operadores más importantes son +, -, *, / y sqrt.

```

1 PRINT 2*3 ; salida -> 6
2
3 MAKE "RUEDAS 4
4
5 PRINT :RUEDAS ; salida -> 4

```

Listado de código B.3: Ejemplo de uso de operadores aritméticos en el lenguaje Logo.

Operadores **IF** y **REPEAT**

Se pueden crear operaciones condicionales utilizando la palabra clave **IF** seguida de una sentencia y una lista de órdenes a ejecutar si se evalúa dicha sentencia a verdadero. La lista de órdenes se declara entre corchetes ([]) y para la sentencia se pueden utilizar los operadores infijos =, > y <. En el código B.4 podemos ver un ejemplo de uso.

```
1 MAKE "PETALOS 4  
3 IF :PETALOS > 3 [ PRINT "TREBOL DE LA BUENA SUERTE" ]
```

Listado de código B.4: Condiciones en el lenguaje logo con el operador **IF**.

Para repetir una serie de instrucciones, Logo ofrece el comando **REPEAT**, que va seguido de un valor numérico y la lista de instrucciones. El valor numérico (que puede ser parametrizado) es el número de veces que se repetirá en bucle las instrucciones. En el código B.5 vemos un ejemplo de este comando.

```
1 REPEAT 10 [ PRINT "HOLA MUNDO" ]
```

Listado de código B.5: Condiciones en el lenguaje logo con el operador **IF**.

Apéndice C

Programando a Turtle de Logo

A parte de toda la funcionalidad que ofrece Logo como lenguaje, Turtle aporta una librería para poder controlar a nuestra criatura que pintará la pantalla. Ésta suele ser tradicionalmente representada con una tortuga o una flecha. Principalmente, se podrá controlar el movimiento de nuestra tortuga y el giro de ésta. También se le podrá ordenar que deje de pintar para permitir movimiento por la pantalla sin rastro. La mayoría de estos comandos pueden abreviarse para simplificar la escritura.

El resumen sobre la funcionalidad de Turtle que se muestra en las secciones siguientes es un compendio formado a partir de [55], [5] y [3].

Moviendo a Turtle

La tortuga siempre se moverá según el eje de coordenadas. El punto (0, 0) se encontrará normalmente en medio de la pantalla y es la posición inicial de la tortuga. El eje de coordenadas y crece positivamente en dirección Norte (hacia arriba).

Los comandos `forward` y `backward` mueven la tortuga hacia delante y atrás, respectivamente, según la posición en la que esté mirando. Los comandos `setx`, `sety` y `setxy` se utilizan para establecer la tortuga en el eje de coordenadas x, y o en ambos, respectivamente.

La orden `home` devuelve a la tortuga a la posición (0, 0) de la pantalla. Adicionalmente, se puede ocultar o mostrar (si ya está oculta) la tortuga con las órdenes `showturtle` y `hideturtle`.

Otra función muy importante de Turtle, es la opción de girarla. Podemos decidir el giro con las instrucciones `right` y `left` y el ángulo de giro (el valor tiene que ser un número positivo). Un ángulo de giro de 380° sería equivalente a girar la tortuga 20° .

Pintando con Turtle

Hasta ahora, cualquier desplazamiento de Turtle por la pantalla dejaba una linea o rastro. Con la orden `penup`, la tortuga dejará de pintar hasta que se ejecute la orden `pendown`. También se puede ejecutar la instrucción `clearscreen` que limpiará la pantalla de cualquier rastro dejado por la tortuga.

La instrucción `label` acepta un argumento y mostrará por pantalla su contenido. El argumento debe ser de tipo *word* o *sentence*¹.

Resumen de instrucciones de Turtle

En la tabla C.1 se muestra un resumen de las órdenes que puede recibir Turtle.

Orden	Abreviatura	Argumentos
<code>forward</code>	<code>fd</code>	Longitud del movimiento
<code>backward</code>	<code>bk</code>	Longitud del movimiento
<code>home</code>	-	-
<code>setx</code>	-	Nueva coodenada X
<code>sety</code>	-	Nueva coodenada Y
<code>setxy</code>	-	Nueva coodenada X e Y
<code>right</code>	<code>rt</code>	Ángulo en el sentido de las agujas del reloj
<code>left</code>	<code>lt</code>	Ángulo en el sentido contrario a las agujas del reloj
<code>penup</code>	<code>pu</code>	-
<code>pendown</code>	<code>pd</code>	-
<code>clearscreen</code>	<code>cs</code>	-
<code>label</code>	-	Una <i>word</i> o <i>sentence</i>
<code>showturtle</code>	<code>st</code>	-
<code>hideturtle</code>	<code>ht</code>	-

Tabla C.1: Resumen de órdenes en Turtle usando el lenguaje Logo.

¹Para más información sobre los tipos de datos *words* y *sentences*, como se forman y cual es su funcionamiento, se puede consultar la sección B.

Apéndice D

Programación con Robomind Academy

En Robomind se plantea la programación de un robot simulado que está dentro de un escenario. Este escenario es una cuadrícula por la que se mueve el robot. El robot puede interactuar con el escenario pintando o limpiando el suelo, recogiendo y dejando objetos, o comprobando el estado de su casillas adyacentes (si hay algún objeto, obstáculo o está pintado).

A continuación vamos a enumerar las funciones básicas para controlar el movimiento o acciones del robot de Robomind¹. Adicionalmente, Robomind también permite usar definición de variables y nuevas funciones, estructuras de control y bucles, típicos de cualquier lenguaje de programación moderno.

- Moviendo el robot:

- `forward` o `forward(n)`: Mover el robot 1 o n pasos hacia el frente.
- `backward` o `backward(n)`: Mover el robot 1 o n pasos hacia el atrás.
- `left` o `left(n)`: Girar el robot 90° hacia la derecha, 1 o n veces.
- `right` o `right(n)`: Girar el robot 90° hacia la derecha, 1 o n veces.

- Interactuando con el escenario:

- `pickUp`: Coger el objeto que se encuentra delante del robot.
- `putDown`: Dejar el objeto justo delante del robot.
- `eatUp`: *Comer* el objeto en frente del robot. Implica destruirlo y no poder volver a interactuar con él.

¹Para una información más detallada se puede consultar <https://www.robomindacademy.com/go/robomind/help>.

- `paintWhite`: Pinta un rastro blanco por las cuadrículas por las que se desplazará el robot.
 - `paintBlack`: Pinta un rastro negro por las cuadrículas por las que se desplazará el robot.
 - `stopPainting`: Deja de pintar.
- El robot puede comprobar, de manera independiente, el estado de las casillas adyacentes a sí mismo, excepto hacia atrás. Para representar cada dirección (*front*, *left* y *right*), se usará una *X*. Todas estas instrucciones devuelven un valor *booleano*.
 - `XIsClear`: Comprueba si la casilla *X* está vacía.
 - `XIsObstacle`: Comprueba si en la casilla *X* hay un obstáculo, generalmente una pared o un objeto que impide su avance.
 - `XIsBeacon`: Comprueba si en la casilla *X* hay un objeto (con el que podrá interactuar).
 - `XIsWhite`: Comprueba si la casilla *X* está pintada de blanco.
 - `XIsBlack`: Comprueba si la casilla *X* está pintada de negro.

Apéndice E

Programando con Scratch

En general, la plataforma Scratch tiene tres grandes componentes:

1. **Un editor visual** que presenta un conjunto de bloques cuya forma ofrece pistas de como pueden encajar los diferentes bloques.
2. **Un interprete** que puede traducir y ejecutar el flujo de código que forman los diferentes bloques unidos entre sí.
3. **Una interfaz** que muestra la salida (tanto de texto como gráfica) del código ejecutado por el interprete. También permite controlar la entrada del ratón, de las teclas, etc.

En la figura E.1 podemos ver un ejemplo del editor y la interfaz así como de un programa de ejemplo que mueve al personaje seleccionado por la pantalla, cambiado su color.

Cuando se programa con Scratch, la meta final es la de animar uno o más elementos por la pantalla. Estos elementos son los *sprites* (imágenes), los cuales vienen predefinidos en una galería o pueden ser proporcionados por el propio usuario. Por tanto, los bloques irán enfocados a otorgar movimiento al *sprite* en cuestión.

En cuanto a los tipos de bloques, actualmente están divididos en 11 categorías distintas:

- **Movimiento:** Incluye funciones para mover, girar o establecer la posición y ángulo de giro del *sprite*.
- **Apariencia:** Esta sección contiene funciones tanto para modificar la apariencia del *sprite* como para mostrar mensajes en la pantalla a modo de diálogos de los *sprites*.
- **Sonido:** Funciones para controlar el sonido de la aplicación que se está creando con Scratch. Permite reproducir sonidos precargados en la plataforma o

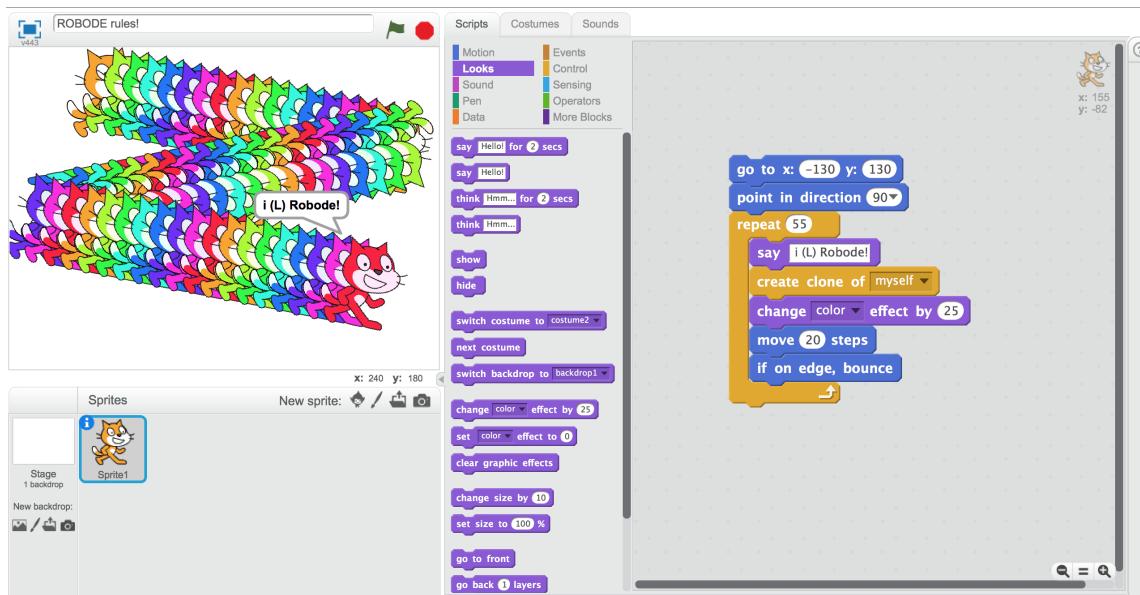


Figura E.1: Ejemplo de la interfaz de usuario y el editor de código que ofrece Scratch.

subidos por el programador, diferentes instrumentos o establecer el volumen de la aplicación.

- **Pinzel:** Permite pintar el escenario de forma similar a Turlte, dejando un rastro por donde se va desplazando (para más información sobre Turtle, consultar sección 2.1.1).
- **Datos:** Creación de variables y listas y modificación de los valores de las mismas.
- **Eventos:** Bloques para detectar y reaccionar a ciertos eventos como el click de un ratón sobre un *sprite* o el teclado.
- **Control:** Incluye bloques con las condiciones y bucles clásicos.
- **Percepción:** Esta sección contiene funciones para detectar eventos de teclado y ratón. También provee funciones para establecer lapsos de tiempo.
- **Operadores:** Bloques para utilizar operadores matemáticos.
- **Bloques propios y Extensiones:** Scratch permite crear tus propios bloques y usar extensiones integradas con la aplicación como *Lego WeDo* y *PicoBoard*.

Apéndice F

Modificación de iJava para el simulador

Cambio en el analizador semántico

Estos cambios se han realizado en la clase `iJavaSemantic`, función `checkDeclaredFunctions`:

```
1 function checkDeclaredFunctions() {
2     var mainFound = false;
3     for (var i = 0; i < declaredFunctions.length; i++) {
4         //cambio la palabra clave "main" por "setup"
5         if (declaredFunctions[i].id === "setup") {
6             mainFound = true;
7         }
8         // ...
9     }
}
```

Listado de código F.1: Código que muestra los cambios que se han realizado en la clase `iJavaSemantic` para cambiar la función principal `main` por `setup`.

En el código F.2 se muestra el código Javascript que se genera de la ejecución del código Javascript 4.1

```
1 running = true;
2 var __setup = null;
3 var __loop = null;
4 var __delay = 0;
5 var __potenciaDer = 0;
6 var __potenciaIzq = 0;
7 var __setup = function() {
```

```

8   __delay = 1000;
10  __potenciaDer = 10;
10  __potenciaIzq = 10;
11  initRobot();
12 }
13 var __loop = function() {
14     power(__potenciaIzq, __potenciaDer);
15     wait(__delay);
16     power(0, 0);
17     wait(__delay * 2);
18 }

20 try {
21     if (__setup) {
22         __setup();
23         if (__loop)
24             animate(__loop);
25     } else stop();
26 } catch (e) {
27     error(e);
28 }
```

Listado de código F.2: Ejemplo de código Javascript que se genera cuando se ejecuta el código iJava 4.1.

Cambios en el analizador sintáctico

Los cambios en el analizador sintáctico corresponden a la clase `iJavaParser`. Es necesario añadir la definición de nuevas variables, funciones o símbolos del sistema en la función `init()`, que será la que los añada como parte del lenguaje. Se muestran el el siguiente código.

```

1 // Sensores de colisiones individuales
2 system_variable("sensorNW", BooleanDatatype, false);
3 system_variable("sensorNE", BooleanDatatype, false);
4 system_variable("sensorSW", BooleanDatatype, false);
5 system_variable("sensorSE", BooleanDatatype, false);

6 //sensores de líneas individuales
7 system_variable("sensorLR", BooleanDatatype, false);
8 system_variable("sensorLL", BooleanDatatype, false);
```

```

//sensores de colisión y líneas grupales
12 system_variable("collisioning", BooleanDatatype, false);

14 // wait (n_millisec)
library_function("wait", new FunctionDatatype(VoidDatatype, [
16   datatype: IntegerDatatype
17 ]));
18 //initRobot ()
library_function("initRobot", new FunctionDatatype(VoidDatatype,
19   []));
20 //stopRobot()
library_function("stop", new FunctionDatatype(VoidDatatype, []));
22 //motors(vel_motor_izq, vel_motor_der)
library_function("power", new FunctionDatatype(VoidDatatype, [
24   datatype: IntegerDatatype
25   },
26   datatype: IntegerDatatype
27 ]));
28 // left()
library_function("left", new FunctionDatatype(VoidDatatype, []));

```

Listado de código F.3: Definición de las funciones y variables de la API del simulador.

Cambios en el Sandbox

La implementación de las funciones y variables de la API se realizan en la clase `iJavaSandbox` y se pueden ver en el código siguiente:

```

1 var roboderunning = false;

3 // sensors
var sensorNE = false,
5   sensorNW = false,
    sensorSE = false,
7   sensorSW = false;
var sensorLR = false,
9   sensorLL = false;
var collisioning = false;

13 function wait(millis) {
  //Add delay time to runtime.timeLimit

```

```
15     if (runtime) {
16         if (runtime.deep > -1) {
17             runtime.timeLimit[runtime.deep] += millis;
18         }
19     }
20
21     var timestamp = (new Date()).getTime() + millis;
22     while ((new Date()).getTime() < timestamp) {
23         //do nothing
24     }
25 }
26
27 function power(lspeed, rspeed) {
28
29     if (!roboderunning) {
30         var msg = {
31             message: "Error: Primero es necesario iniciar el
32                     robot con la funcion: 'iniciarRobot()'."
33         };
34         error(msg);
35         return;
36     }
37
38     var message = {
39         fn: "move",
40         params: [lspeed, rspeed]
41     };
42     sendMessage("robode", message);
43 }
44
45 function stop() {
46     if (!roboderunning) {
47         var msg = {
48             message: "Error: Primero es necesario iniciar el
49                     robot con la funcion: 'iniciarRobot()'."
50         };
51         error(msg);
52         return;
53     }
54
55     var message = {
56         fn: "stop",
57         params: []
58     }
```

```

    };
    sendMessage("robode", message);
}

function manageSensors(message) {
    switch (message.id) {
        case "sensorNW":
            sensorNW = (message.state === "begin");
            break;
        case "sensorNE":
            sensorNE = (message.state === "begin");
            break;
        case "sensorSW":
            sensorSW = (message.state === "begin");
            break;
        case "sensorSE":
            sensorSE = (message.state === "begin");
            break;
        case "sensorLR":
            sensorLR = (message.state === "begin");
            break;
        case "sensorLL":
            sensorLL = (message.state === "begin");
            break;
    }
    collisioning = sensorNW || sensorNE || sensorSW || sensorSE;
}

```

Listado de código F.4: Implementación de las funciones y variables de la API en el Sandbox de iJava.

Apéndice G

Creación de elementos en Box2D

La creación de elementos (*bodies*, *fixtures*, *joints*, etc) en Box2D utiliza un *patrón factoría*, teniendo que definir primero en un objeto plantilla (definición) las características que tendrá el elemento a crear y luego diciéndole a `World` que cree dicho elemento a partir de la plantilla.

A continuación se explicarán las características principales de los elementos de Box2D y sus propiedades mas importantes.

Objeto Body

Los cuerpos en Box2D pueden ser dinámicos (tipo `b2_dynamicBody`) o estáticos (tipo `b2_staticBody`). Un cuerpo dinámico podrá moverse y se verá afectado por el entorno y otros cuerpos. Un cuerpo estático será inmune a fuerzas como la gravedad y su posición no cambiará. Robode y todos sus componentes serán cuerpos dinámicos mientras que las fronteras del mundo, serán cuerpos estáticos. También, un cuerpo está activo si su propiedad `sleep` está activa. El efecto de un cuerpo inactivo ya se ha explicado en la sección 4.3.1.

Un cuerpo conoce los *fixtures* asociados a él (que puede ser más de uno) y los *joints* a los que está atado.

Es el objeto `Body` el que contiene todas las propiedades físicas y que afectan al movimiento del mismo, como por ejemplo: la velocidad lineal y angular, el ángulo de rotación o la masa. Por supuesto, también la posición. Para obtener la posición de un cuerpo se utiliza el método `GetWorldCenter()` para obtener la posición en el mundo del centro de masas del cuerpo.

Las propiedades `linearDamping` y `angularDamping`¹ sirven para reducir la velocidad lineal y angular respectivamente cuando en el frenado.

¹Aquí, con *damping* se refiere al efecto de suelo mojado y el deslizamiento que un cuerpo tiene sobre una superficie, en especial cuando el cuerpo frena.

Adicionalmente, se ha modificado el prototipo de las clases `b2Body` y `b2BodyDef` (clase plantilla para crear *bodies*) para añadir una característica más al cuerpo, el nombre que se le asigna al cuerpo. Esto se usará principalmente para poder distinguir estos objetos fácilmente en una colisión.

Objeto **Fixture**

Los objetos `b2Fixture` definen la apariencia del objeto al que están asociado y las características físicas que le harán interactuar con el mundo de una u otra manera.

Box2D proporciona dos clases principales para definir la forma de las *fixtures*: `b2CircleShape()` y `b2PolygonShape()`. La primera representa a una forma circular y es necesario indicar el radio del círculo a crear. La segunda forma representa a un polígono. Los polígonos tienen dos formas principales de definirse:

- Con forma de caja definiendo su ancho y largo (con el método `SetAsBox()`).
- Con una forma libre. Es necesario indicar los vértices del polígono y su cantidad (con el método `AsArray()`). Es así como se pueden crear triángulo o trapecios en Box2D, por ejemplo. En este caso es importante no crear formas cóncavas, pues provocarán errores en la simulación.

Además, un *fixture* tiene propiedades de densidad, fricción y restitución que definirá la interacción de este con el mundo. La densidad es usada para calcular la masa del cuerpo al que está asociada la *fixture*. La fricción se usa para hacer los cuerpos deslizarse de una manera realista. La restitución de un cuerpo es la elasticidad de este ante un impacto y según su valor, si rebota más o menos contra una superficie.

Los valores de fricción y restitución oscilan entre 0 y 1. Siendo 0 la anulación de la propiedad y 1 el máximo valor que se le puede dar.

Objeto **Joint**

Un *Joint* es un elemento constrictor entre dos cuerpos, que controla el movimiento que hacen los cuerpos pegados por el joint. El punto por el que se une un cuerpo a un joint se denomina *ancla*.

Hay distintos tipos de joints que limitan el movimiento de los cuerpos de diferentes maneras. A continuación se nombrarán los más importantes:

- `Revolute` es un joint que fuerza a dos cuerpos a compartir el mismo *ancla* con un cierto ángulo de libertad. También se puede configurar un *motor* que hará girar el ancla y por tanto a los cuerpos.

- `Distance` es un joint que permite ata los cuerpos pero como si estuvieran sujetos a una cuerda. Los cuerpos tendrán una distancia máxima que se pueden separar (la longitud de la cuerda) pero no una mínima, pudiendo incluso chocar.
- `Prismatic` es un joint que simula el movimiento de un cuerpo únicamente sobre un eje de coordenadas, permitiendo movimientos como los de un ascensor.

En el simulador solo usaremos *Joints* de tipo *Revolute*, definidos por la clase `b2RevoluteJoint`. En la inicialización del joint se deben indicar los dos cuerpos y el punto de ancla. También hay que establecer las propiedades relacionadas al motor (propiedad `enableMotor`), si se quiere que exista un límite de movimiento o no (propiedad `enableLimit`) y el valor máximo de torsión que se permite (propiedad `maxMotorTorque`).

Apéndice H

Construcción de circuitos

El circuito estará limitado por fronteras para prevenir que el robot se salga del mismo. La creación de las fronteras se puede ver en el código H.1.

```
1  var scale = Simulator.config.scaleWorldIni;
2
3  var bodyDef = new Simulator.Env.b2BodyDef();
4  bodyDef.type = Simulator.Env.b2Body.b2_staticBody;
5  bodyDef.setName("border");
6
7  var fixDef = new Simulator.Env.b2FixtureDef();
8  fixDef.shape = new Simulator.Env.b2PolygonShape();
9
10 // lower border
11 fixDef.shape.SetAsBox(this.width / scale, 0.01);
12 bodyDef.position.Set((this.width / scale) / 2, this.height /
13                      scale);
14 Simulator.World.CreateBody(bodyDef).CreateFixture(fixDef);
15
16 // top border
17 bodyDef.position.Set((this.width / scale) / 2, 0);
18 Simulator.World.CreateBody(bodyDef).CreateFixture(fixDef);
19
20 // right border
21 fixDef.shape.SetAsBox(0.01, this.height / scale);
22 bodyDef.position.Set(this.width / scale, (this.height / scale) /
23                      2);
24 Simulator.World.CreateBody(bodyDef).CreateFixture(fixDef);
25
26 // left border
```

```
26     bodyDef.position.Set(0, (this.height / scale) / 2);  
      Simulator.World.CreateBody(bodyDef).CreateFixture(fixture);
```

Listado de código H.1: Función que crea las fronteras del mundo.

La posición y tamaño de las fronteras se crearán en función de la escala inicial del mundo. Por otra parte, las fronteras se definirán como objetos `b2_staticBody` para que permanezcan inmutables en nuestro mundo.

Un obstáculo se crearía de manera similar a como se han creado los ruedas o el cuerpo principal.

Bibliografía

- [1] A logo primer. http://el.media.mit.edu/logo-foundation/what_is_logo/logo_primer.html, Último acceso en enero 2016.
- [2] Proyecto Código 21. Página oficial del proyecto código 21 promovido por el departamento de educación de gobierno de navarra. <http://codigo21.educacion.navarra.es>, Consultado el 9 de enero de 2016.
- [3] Harold Abelson. A. disessa. turtle geometry. *Cambridge (Mass.)*, 1980.
- [4] Code Academy. Página oficial de code academy. <https://www.codecademy.com>, Consultado el 12 de diciembre de 2015.
- [5] Turtle Academy. Página oficial del proyecto turtle academy. <https://turtleacademy.com>, Consultado el 25 de enero de 2016.
- [6] Noreen S. Ahmed-Ullah. Cps to add computer science as core subject. *Chicago Tribune*, Diciembre 2013.
- [7] Arduino. Página oficial del proyecto arduino. <https://www.arduino.cc/>, Consultado el 9 de enero de 2016.
- [8] Joseph Bergin, J Eckstein, M Manns, H Sharp, M Voelter, E Wallinfgord, K Marquardt, J Chandler, and A Fricke. Pedagogical patterns. early bird. <http://csis.pace.edu/~bergin/PedPat1.2.html#earlybird>, 2005. Consultado el 3 de febrero de 2016.
- [9] Box2d. Página oficial del proyecto box2d. <http://box2d.org>, Consultado el 1 de abril de 2015.
- [10] Judy Ann Brown, Dorothy M Fitch, and Scott W Earle. 101 ideas for logo, gr. 3-10, 1995.
- [11] Alan Bundy. Computational thinking is pervasive. *Journal of Scientific and Practical Computing*, 1(2):67–69, 2007.

- [12] Erin Catto. User manual. box2d. <http://www.box2d.org/manual.html>, 2007. Consultado el 1 de febrero de 2016.
- [13] Douglas H Clements. Effects of logo and cai environments on cognition and creativity. *Journal of Educational Psychology*, 78(4):309, 1986.
- [14] Code.org. Página oficial del proyecto hora del código de code.org. <https://hourofcode.com/>, Consultado el 30 enero de 2016.
- [15] Code.org. Página oficial de code.org. <https://code.org>, Consultado el 9 de enero de 2016.
- [16] Social Committee Commission to the European Parliament, the European Economic and the Committee of the Regions. Draft 2015 joint report of the council and the commission on the implementation of the strategic framework for european cooperation in education and training (et 2020). new priorities for european cooperation in education and training. http://ec.europa.eu/education/documents/et-2020-swd-161-2015_en.pdf, Agosto 2015.
- [17] Antonella Corsi-Bunker. *Guide to the Educational System in the United State*. University of Minnesota. Internal Student and Scholar Service.
- [18] Sean Coughlan. Computer science part of english baccalaureate. *BBC News*, Febrero 2013.
- [19] Julie Sarama Douglas H. Clements, Michael T. Battista. Logo and geometry. *Journal for Research in Mathematics Education. Monograph*, 10:i–177, 2001.
- [20] Robert Dunlop. Introduction to catmull-rom splines. <http://www.mvps.org/directx/articles/catmull/>, Julio 2005.
- [21] Moway Education. Página oficial de moway education. <http://moway-robot.com>, Consultado el 10 octubre de 2015.
- [22] L Fernández Muñoz, R Peña, F Nava, and A Velázquez Iturbide. Análisis de las propuestas de la enseñanza de la programación orientada a objetos en los primeros cursos. *Actas de las VIII Jornadas de Enseñanza Universitaria de la Informática (JENUI'02)*, pages 433–440, 2002.
- [23] Wallace Feurzeig et al. Programming-languages as a conceptual framework for teaching mathematics. final report on the first fifteen months of the logo project. 1969.

- [24] Department for Education UK Government. Statutory guidance. national curriculum in england: computing programmes of study. <https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study>, Septiembre 2013.
- [25] Michael Friendly. *Advanced LOGO: A language for learning*. Psychology Press, 2014.
- [26] Raspberry Pi Foundation. Página oficial de raspberry pi. <https://www.raspberrypi.org>, Consultado el 15 enero de 2016.
- [27] ECMAScript International. Ecma-262, ecmascript 2015 language specification. <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>, Junio 2015.
- [28] Descubre la programación. Página oficial del proyecto descubre la programación. <http://descubre.inf.um.es>, Consultado el 9 de enero de 2016.
- [29] Descubre la programación. Página oficial del proyecto descubre la programación. guía del lenguaje. <http://descubre.inf.um.es/curso.php>, Consultado el 9 de enero de 2016.
- [30] Maude Lemaire. Incorporating computer science into an elementary school curriculum. 2014.
- [31] Yuen-Kuang Cliff Liao and George W Bright. Effects of computer programming on cognitive outcomes: A meta-analysis. *Journal of Educational Computing Research*, 7(3):251–268, 1991.
- [32] Curly Logo. Página oficial del proyecto curly logo. una implementación web de la famosa turtle de logo. <http://drj11.github.io/curlylogo/>, Consultado el 31 de enero de 2016.
- [33] Fundación Logo. Página oficial de la fundación logo. recursos para aprender y enseñar logo. <http://el.media.mit.edu/logo-foundation/resources/books.html#language>, Consultado el 31 de enero de 2016.
- [34] Fundación Logo. Página oficial de la fundación logo. <http://el.media.mit.edu/logo-foundation>, Consultado el 9 de enero de 2016.
- [35] José Marcos. Los colegios de madrid impartirán clases de programación. *El País*, Septiembre 2014.

- [36] Andrew Mcgettrick, Roger Boyle, Roland Ibbett, John Lloyd, Gillian Lovegrove, and Keith Mander. Grand challenges in computing: Education. a summary. *The Computer Journal*, 48(1):42–48, 2005.
- [37] Lego Mindstorm. Página oficial de lego minsturm. <http://www.lego.com/es-es/mindstorms/>, Consultado el 20 de enero de 2016.
- [38] Lego Mindstorm. Página oficial de lego minsturm. sección aprende a programar. <http://www.lego.com/es-es/mindstorms/learn-to-program>, Consultado el 20 de enero de 2016.
- [39] Richard Moss. *Creating a mathematical environment through programming: A study of young children learning Logo*. PhD thesis, University of London, 1985.
- [40] Mozilla Developer Network. Web workers api. https://developer.mozilla.org/es/docs/Web/API/Web_Workers_API, Noviembre 2015 Consultado el 6 de febrero de 2016.
- [41] Richard Pattis, J Roberts, and M Stehlik. Karel the robot. *A gentle introduction to the Art of Programming*, 1981.
- [42] Roy D Pea and D Midian Kurland. Logo programming and the development of planning skills. technical report no. 16. 1984.
- [43] Pomax. A primer on bézier curves. <http://pomax.github.io/bezierinfo>, Consultado el 5 de febrero de 2016.
- [44] V Renumol, S Jayaprakash, and D Janakiram. Classification of cognitive difficulties of students to learn computer programming. *Indian Institute of Technology, India*, 2009.
- [45] John Resig. How javascript timers work. <http://ejohn.org/blog/how-javascript-timers-work/>, Febrero, 2008 Último acceso en enero 2016.
- [46] John Resig and Bear Bibeault. *Secrets of the JavaScript Ninja*. Manning, 2013.
- [47] RoboMind. Página oficial de robomind. <https://www.robomindacademy.com>, Consultado el 10 de octubre de 2015.
- [48] Jazz Ros. Projecting a point on a bézier curve. <http://jazzros.blogspot.com.es/2011/03/projecting-point-on-bezier-curve.html>, Marzo 2011.

- [49] Juan Antonio Sánchez Laguna. ijava: un nuevo lenguaje para facilitar el paso del paradigma imperativo al orientado a objetos. *Jornadas de Enseñanza Universitaria de la Informática (15es: 2009: Barcelona)*, 2009.
- [50] Code School. Página oficial de code school. <https://www.codeschool.com>, Consultado el 9 de enero de 2016.
- [51] Scratch. Página oficial del proyecto scratch. <https://scratch.mit.edu>, Consultado el 30 enero de 2016.
- [52] Janet Siegmund, Christian Kästner, Sven Apel, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, and André Brechmann. Understanding understanding source code with functional magnetic resonance imaging. In *Proceedings of the 36th International Conference on Software Engineering*, pages 378–389. ACM, 2014.
- [53] Erin Swenson-Healey. The javascript event loop: Explained. <http://blog.carbonfive.com/2013/10/27/the-javascript-event-loop-explained/>, Octubre 2013 Último acceso en enero 2016.
- [54] Christopher Twigg. Catmull-rom splines. <https://www.cs.cmu.edu/~fp/courses/graphics/asst5/catmullRom.pdf>, Marzo 2003.
- [55] Brown University. Logo tutorial. un tutorial de turtle y el lenguaje logo dentro de la brown university. <http://cs.brown.edu/courses/bridge/1997/Resources/LogoTutorial.html>, Consultado el 30 de enero de 2016.
- [56] EU Code Week. Página oficial del proyecto eu code week. <http://codeweek.eu>, Consultado el 9 de enero de 2016.
- [57] Jeannette M Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.
- [58] Jeannette M Wing. Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 366(1881):3717–3725, 2008.
- [59] Eurydice Datos y Cifras. Educación y formación. La estructura de los sistemas educativos europeos 2014/15: diagramas. http://eacea.ec.europa.eu/education/eurydice/documents/facts_and_figures/education_structures_es.pdf, Noviembre 2014.