UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Arqueología informática: Análisis, diseño e implementación del funcionamiento del ábaco matemático con Scratch

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Sergio Puche García

Tutor: Xavier Molero Prieto

Curso 2017-2018

Resum

El treball planteja i realitza una investigació bidireccional. En primer lloc es desenvolupa un estudi de caràcter històric al voltant del passat de les màquines de còmput, centrat en el cas concret dels àbacs. D'aquesta manera, i posant el focus sobre tres exemples específics (els àbacs japonès, xinès i rus), s'estudien els mecanismes d'aquestes eines de càlcul considerades com a predecessores de l'ordinador contemporani. A la segona part del treball s'inverteix la perspectiva i es proposa utilitzar un llenguatge de programació eminentment modern i singular com Scratch per a simular el comportament dels tres àbacs estudiats. Per últim, es planteja compartir els resultats mitjançant la pàgina web del Museu d'història de la informàtica de la Universitat Politècnica de València.

Paraules clau: àbac, Scratch, història de la computació

Resumen

El trabajo plantea y realiza una investigación bidireccional. En primer lugar se desarrolla un estudio de carácter histórico sobre el pasado de las máquinas de cómputo, centrado en el caso concreto de los ábacos. De este modo, y poniendo el foco sobre tres ejemplos específicos (los ábacos japonés, chino y ruso), se estudian los mecanismos de estos instrumentos de cálculo considerados como predecesores del ordenador contemporáneo. En la segunda parte del trabajo se invierte la perspectiva y se propone utilizar un lenguaje de programación eminentemente moderno y singular como Scratch para simular el comportamiento de los tres ábacos estudiados. Por último, se plantea compartir los resultados a través de la página web del Museo de historia de la informática de la Universitat Politècnica de València.

Palabras clave: ábaco, Scratch, historia de la computación

Abstract

This academic work lays out and conducts a two-way research. Firstly, a historical study is developed around the past of calculating machines, focused in the concrete case of abacuses. This way, and bringing three specific examples into focus (the Japanese, Chinese and Russian abacuses), the mechanisms of these calculating machines considered as predecessors of the contemporary computer are studied. In the second part of the research the point of view is inverted and the use of an eminently modern and unique programming language as it is Scratch is proposed to simulate the behavior of the three examined abacuses. Lastly, it is proposed to share the results on the Computing Science History Museum website, belonging to the Universitat Politècnica de València.

Key words: abacus, Scratch, history of computation

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En este primer capítulo se presentará sucintamente el trabajo, cuál es la motivación que lo inspira, los objetivos que tratará de abarcar, la estructura del mismo y una breve explicación sobre la bibliografía consultada para su desarrollo.

1.1 Motivación

El campo de la arqueología informática pretende aunar los esfuerzos de distintas disciplinas (informática, historia, matemáticas...) para lograr un estudio en profundidad del pasado de la informática hasta nuestros días. Una forma de entender la arqueología informática podría ser desde un punto de vista de simple explicación de los contextos históricos en los que se desarrollaron los distintos instrumentos de cálculo.

No obstante, el camino también puede recorrerse en el sentido opuesto, y en lugar de un mero estudio del pasado que ayude a comprender las sociedades que dieron origen al ábaco o el papel que éste interpretó en ellas, es posible aprovechar la informática para crear una simulación exacta aunque virtual de estos distintos ábacos para hacerlos accesibles desde la comodidad de un navegador web (en nuestro caso en el sitio del Museo de informática de la Escuela técnica).

En el caso de este trabajo concreto, el foco se sitúa en el ábaco. El ábaco es uno de los más lejanos antepasados de las computadoras modernas en cuanto a su función de herramienta de cálculo, aunque sigue vigente hasta nuestros días gracias a su sencillez. Para nosotros los informáticos, el ábaco suscita interés tanto por su naturaleza de «antepasado lejano» del ordenador, como por la potencia computacional que se puede llegar a alcanzar con una herramienta aparentemente trivial.

1.2 Objetivos

Este trabajo pretende ofrecer a la vez una visión general desde la perspectiva de la arqueología informática de lo que ha sido el avance de los instrumentos de cálculo y computación a lo largo de la historia, así como poner el foco en uno de los primeros (pero a la vez duradero hasta nuestros días): el ábaco.

Más específicamente, esto podría concretarse en los siguientes objetivos:

1. Proporcionar una visión general del desarrollo a lo largo de la historia de distintas máquinas dedicadas al cálculo.

2. Conocer la historia de los distintos tipos de ábacos que han sido desarrollados en distintos continentes y épocas.

3. Profundizar en el funcionamiento de algunos de estos ábacos, explicando sus técnicas y algoritmos de cálculo.

4. Implementar en Scratch tres simuladores que reproduzca el comportamiento de distintos ábacos.

5. Crear una página web en el sitio del Museo de Informática dedicada al ábaco en el que se incluyan los simuladores implementados.

1.3 Estructura de la memoria

Este trabajo consta de cinco capítulos, además de un apéndices finales. A continuación se enumeran los capítulos del documento y se realiza un muy breve resumen del contenido de cada uno de ellos:

Capítulo 1. Capítulo introductorio en el que se describen la motivación y los objetivos que han guiado el trabajo. También se describe brevemente la estructura del trabajo y se realiza un comentario sobre la bibliografía.

Capítulo 2. Se describe el contexto histórico general en el que se desarrollaron los primeros instrumentos de ayuda al cálculo. Más específicamente, se describen los distintos ábacos inventados en América, Asia y Europa. Además, se pone el foco sobre tres ábacos concretos (el japonés, el chino y el ruso) y se profundiza en el funcionamiento de los mismos.

Capítulo 3. Aquí se explica superficialmente la historia y la concepción del lenguaje de programación Scratch. También se tratará de hacer una introducción al entorno, así como de explicar los conceptos y rudimentos esenciales del mismo.

Capítulo 4. Una vez conocidos los mecanismos tanto de los tres ábacos objeto de estudio como del lenguaje Scratch, se procederá a implementar un simulador de cada uno de ellos. En este capítulo se describe todo el proceso así como los fragmentos más significativos del código.

Capítulo 5. Finalmente, se hace una recopilación de los objetivos perseguidos y los logros alcanzados mediante el trabajo, y se hace un breve comentario sobre posibles ampliaciones o mejoras futuras.

1.4 Notas sobre la bibliografía

Para llevar a cabo la realización de este trabajo se han utilizado como materiales de consulta las obras citadas en la bibliografía final. Fundamentalmente, se han realizado consultas a libros de texto en inglés o español y a algunas páginas web especializadas en alguno de los temas relevantes para el trabajo. También se ha utilizado puntualmente algún artículo de revistas especializadas.

Para la parte del contexto histórico y sobre las distintas máquinas calculadoras, y en concreto sobre los ábacos a través de los siglos y en diferentes lugares, se han utilizado obras más generalistas como las dos clásicas de Ifrah sobre la historia de los números y de los instrumentos de computación [5, 6] y otras, que se pueden encontrar en las referencias [2, 4, 7, 11, 13, 14]. Sobre la historia de los ábacos en concreto la obra más completa, aunque ya un tanto clásica, es la de Pullan [11], mientras que la de Kojima [7] contiene algunos datos muy interesantes sobre los orígenes del *soroban* y anécdotas curiosas sobre el mismo.

Sobre el manejo, técnicas y algoritmos para la realización de cálculos aritméticos utilizando el ábaco, el libro usado como manual básico ha sido el de Kojima [7], cuyos detallados algoritmos, numerosos ejemplos y ejercicios se circunscriben únicamente al *soroban*, pero realmente son trasladables sin dificultad a los casos del *suanpan* y del *schoty*. Un buen complemento al manual de Kojima es el documento publicado por la *Liga para la educación del soroban en Japón*, que plantea unos algoritmos muy sencillos para realizar operaciones atómicas de la forma más eficiente posible, acompañados de multitud de ejercicios [8]. Aparte de este manual existen otros documentos más sucintos, pero que también aportan técnicas muy válidas para el manejo de los ábacos e incluso algunos tratan operaciones como exponenciales y raíces, temas no cubiertos en el manual de Kojima [1, 3, 13, 15, 16]. Además, han sido de gran utilidad para tomar un primer contacto con los ábacos algunos simuladores implementados previamente por otros programadores y hechos accesibles como recursos web. [1]

El tercer grupo de materiales consultados son los referidos al lenguaje y entorno de programación Scratch. Sobre Scratch se han utilizado manuales, artículos y documentos de referencia. Entre ellos hay desde libros de consulta o manuales orientados al aprendizaje del lenguaje y de técnicas de programación, hasta otros utilizados para ofrecer un contexto más general sobre los orígenes de Scratch. Estas obras son las referenciadas en [9, 10, 12]. También se ha consultado de forma asidua la información publicada en la Wiki oficial de Scratch, situada dentro de su propio sitio web [17].

[1] Simulador de *soroban*: http://www.alcula.com/soroban.php, consultado el 23 de abril de 2018.
Simulador de *suanpan*: http://www.its.caltech.edu/~antonycs/abacus-bin/, consultado el 23 de abril de 2018.
Simulador de *schoty*: http://micro.magnet.fsu.edu/electromag/java/abacus/, consultado el 23 de abril de 2018.

CAPÍTULO 2
Contexto histórico

Una de las características definitorias del ser humano como especie diferenciada ha sido desde sus orígenes su capacidad para crear instrumentos que le ayuden en aquellas tareas que resultaban imposibles o muy difíciles de llevar a cabo sin más herramientas que el cuerpo y el cerebro humanos. Desde sistemas de palancas y poleas para realizar labores imposibles para la fuerza física de una simple persona, hasta aviones o naves espaciales que llevan al ser humano a lugares a los que ni podría aspirar a llegar utilizando sus pies. En nuestro caso, como informáticos, nos interesan especialmente las herramientas que se utilizaban en aquellos tiempos en los que no existían no ya los ordenadores ni las calculadoras electrónicas, sino ni tan siquiera el cálculo escrito. Además, conocer las herramientas de cálculo primitivas (o no tanto) nos permite observar desde una perspectiva general los distintos caminos que permitieron llegar durante el siglo XX al desarrollo de la computadora moderna, que no es sino el resultado de las ideas matemáticas e ingenieriles concebidas a lo largo de muchos siglos.

En el caso concreto de este trabajo, además de esta visión más general de los instrumentos de cálculo, se sitúa el foco sobre uno de los más antiguos y a la vez más longevos: el ábaco. Bajo este concepto se engloban todo tipo de instrumentos basados en una tabla o estructura sólida sobre la que se realizan cálculos mediante piedras o pequeñas cuentas de cualquier tipo. En la segunda parte de esta sección se proporciona una perspectiva de la larga historia de los ábacos, explicando las diferencias y el funcionamiento básico de los más importantes.

Por último, se dedica la parte final de este apartado al estudio de los algoritmos de cálculo con ábaco para las cuatro operaciones aritméticas básicas, una forma alternativa (pero no necesariamente menos precisa ni eficiente) a los cálculos en papel o en una máquina digital.

2.1 Una breve historia de las máquinas calculadoras

Las primeras herramientas de cuenta y cálculo utilizadas por el ser humano fueron sin duda los dedos de sus propias manos y, en el caso de algunas culturas tropicales, también de sus pies [2]. Estas «herramientas», aunque resulten extremadamente simples, han servido a lo largo de la historia y hasta la misma actualidad para llevar a cabo cuentas y operaciones de adición y sustracción sen-

cillas. Aunque pudiera parecer que solo es posible representar números hasta el 10, en muchas culturas se desarrollaron métodos basados en el uso no solo de los dedos, sino también de falanges y nudillos, lo cual permitía alcanzar magnitudes de millares o incluso millones [6].

Los palos y huesos tallados fueron probablemente el siguiente instrumento de conteo en ser inventado. Estos consistían en maderas, huesos o piedras, generalmente de forma alargada, sobre los cuales se realizaban una serie de incisiones, cada una de las cuales representaba una unidad, de tal modo que resultaba posible representar números arbitrariamente grandes y de forma permanente, al contrario de lo que sucedía con las manos. Así pues, con estos instrumentos se hizo posible llevar a cabo inventarios y, en general, poseer registros numéricos de cualquier cosa. Los primeros palos tallados datan del Paleolítico Superior, de entre el 35 000 y el 20 000 antes de Cristo [6]. En la Figura 2.1 pueden observarse ejemplares de esa época. No obstante, esta técnica ha logrado perdurar gracias a su sencillez y facilidad de uso hasta los tiempos modernos, como por ejemplo en las famosas muescas que los pistoleros del Oeste americano hacían en sus Winchester.

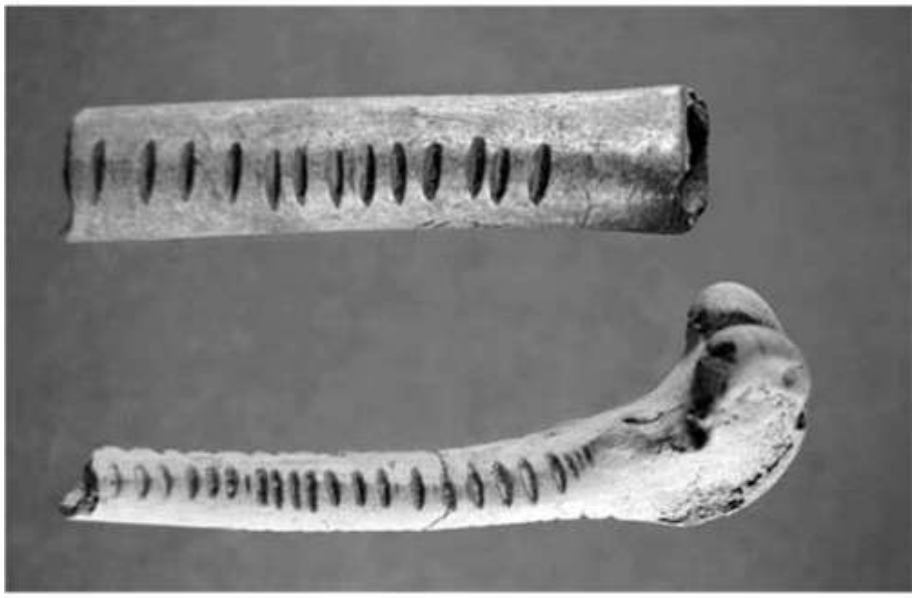

Figura 2.1: Huesos con incisiones utilizados como instrumentos de cuenta. Datan de hace 28 000 años. Hallados en Le Cellier, Dordoña (Francia)

Otro método de cuenta de indudable antigüedad es el uso de piedras pequeñas formando grupos o montones para representar cantidades. Por la naturaleza efímera de este procedimiento es difícil conocer exactamente a qué época se remonta. Lo que sí sabemos es que es común a culturas de todo el mundo, y que por ejemplo es el que usaban griegos y romanos para enseñar a contar y a realizar operaciones aritméticas a sus niños. De ahí la curiosa polisemia de la palabra cálculo en castellano y otras lenguas romances. Cálculo procede etimológicamente del latín *calculus*, piedra [4, 6, 11]; aunque en castellano solo conserve su significado original cuando se aplica en contextos de problemas de salud. Y es que, al contrario que los palos tallados, el uso de piedras facilitaba mucho la realización de operaciones matemáticas, o al menos de las más básicas. También a partir de

estas técnicas se desarrollaron los primeros ábacos que, al fin y al cabo, no son más que tablas con pequeñas piedras o cuentas organizadas de un cierto modo para representar un número determinado.

El resto de la Edad Antigua y toda la Edad Media estarían dominadas por el uso del ábaco o de instrumentos similares (tablas de cálculo, el sistema americano precolombino basado en cuerdas, etc.). Todo ello se abordará con más detalle en la próxima sección, por lo que aquí realizaremos un salto temporal hasta la Edad Moderna, cuando los avances en la tecnología y en los conceptos matemáticos, y en especial la adopción del sistema numérico posicional de origen indio e importado a través de los árabes, permitieron la invención de nuevos e ingeniosos mecanismos de cálculo cada vez más complejos.

El primero de ellos fue el ábaco neperiano o varillas de Napier que, pese a su nombre, tienen muy poco que ver con un ábaco en su mecanismo de funcionamiento. Inventado por el matemático escocés John Napier en 1617, el sistema constaba de 10 varillas, numeradas del cero al nueve, y cada una de ellas tenía nueve subdivisiones con los múltiplos del número de la varilla en cuestión ordenados de arriba a abajo y cada uno de ellos con sus dos dígitos divididos por una diagonal que cumplía la función de agrupar visualmente un acarreo con el número siguiente al que debía sumarse. La varilla del cuatro, por ejemplo, tendría las siguientes anotaciones de arriba a abajo: $0/4$, $0/8$, $1/2$, $1/6$, $2/0$, $2/4$, $2/8$, $3/2$ y $3/6$. Para realizar una multiplicación, las varillas se ordenarían de acuerdo con uno de los dos factores y, a continuación, se podría leer cada uno de los productos parciales de forma horizontal. No obstante, para obtener el total aún era necesario que el usuario realizase manualmente la suma de los subtotales. En la Figura 2.2 puede observarse un ejemplo para operar con el factor 2 394.

La primera máquina calculadora propiamente dicha fue creada por el astrónomo alemán Wilhelm Schickard en 1623. De ella se tiene constancia gracias a la correspondencia escrita entre Schickard y su amigo Kepler, y a las notas y bocetos hallados, ya que el inventor alemán solo construyó una de estas máquinas y fue destruida en un incendio al año siguiente al de su creación. En la Figura 2.3 puede observarse una reproducción de lo que debió de ser la máquina original. El aparato, al que su autor bautizó con el nombre de «reloj calculante», era capaz de realizar las operaciones de suma y resta sin la intervención del usuario más que para introducir los valores, mientras que la multiplicación y la división sí que precisaban de la participación directa de éste. El mecanismo interno que dirigía el funcionamiento del reloj calculante se basaba en un sistema de ruedas dentadas y cilindros que implementaban la lógica matemática de las operaciones aritméticas [5].

Un par de décadas más tarde, en 1642, el matemático y filósofo francés Blaise Pascal inventó una máquina similar (pero de forma totalmente original, pues no pudo tener constancia de ningún modo del reloj calculante de Schickard) a la que llamó la «Pascalina» (*Pascaline*). En este aparato se introducían los dos números de forma secuencial mediante unas ruedas, y el sistema de engranajes dentados desarrollado por Pascal se encargaba automáticamente de realizar la suma y los acarreos y de mostrar los resultados a través de unas ranuras. La Pascalina solo era capaz de realizar la operación de adición de forma mecánica, aunque se podía restar mediante la suma del complementario, pero aún así alcanzó gran

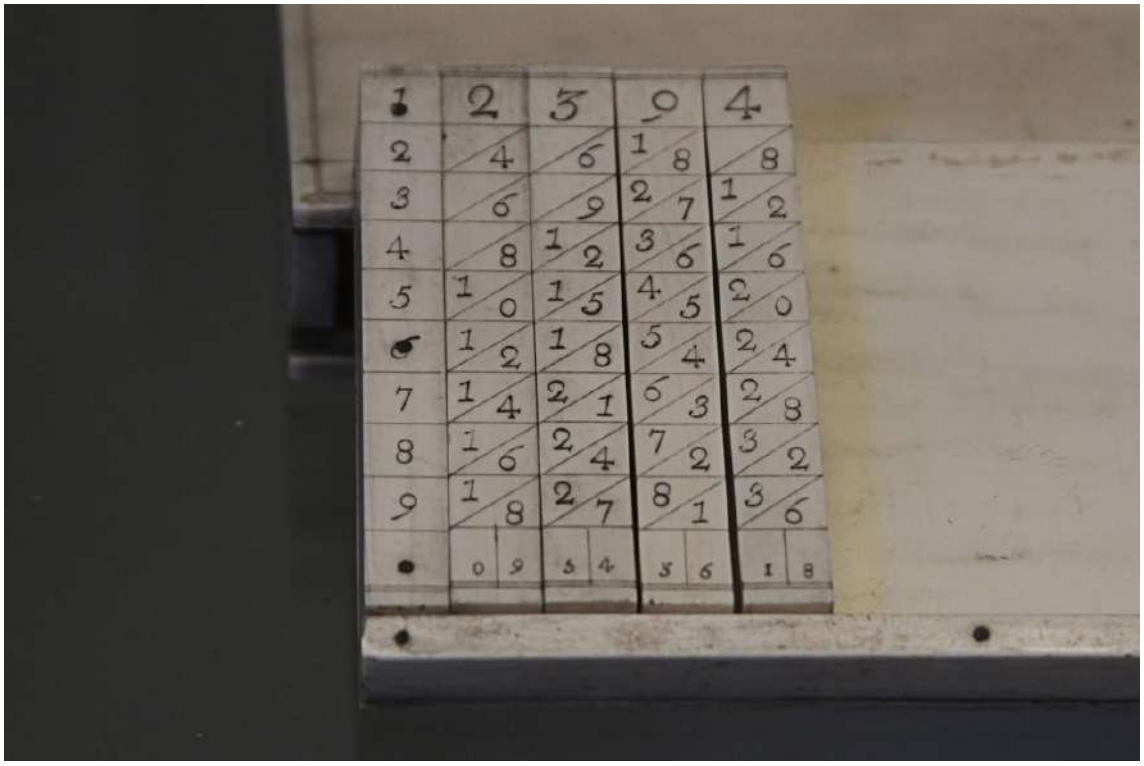

Figura 2.2: Juego de varillas de Napier fabricadas en marfil colocadas para realizar una multiplicación sobre el número 2 394. Si por ejemplo queremos leer el resultado al multiplicarlo por 3, bastaría con leer de derecha a izquierda la fila 3 realizando los acarreos pertinentes para al final obtener el producto, 7 182

popularidad y llevó a otras muchas personas a realizar mejoras sobre la misma, con distintos grados de utilidad real. En la Figura 2.4 aparece una reproducción de esta máquina.

En 1694 el matemático y filósofo alemán Gottfried Leibniz construyó la primera máquina capaz de realizar las cuatro operaciones aritméticas básicas de forma semiautomática, es decir, que la intervención del operario se reducía únicamente a algunos procesos mecánicos. Sin embargo, la complejidad de los mecanismos necesarios para lograr esto provocaban numerosos fallos y dificultades en el funcionamiento, todo ello sumado a algún error en la implementación que hacía que fallasen algunos acarreos [5], lo que podría considerarse un verdadero predecesor de los *bugs* informáticos. El núcleo central de estos mecanismos era la llamada «rueda» o «cilindro de Leibniz» que, como su nombre indica, consistía en una pieza en forma de cilindro dentado que en su rotación activaba la rueda de conteo de la máquina. Aunque la creación de Leibniz nunca llegó a popularizarse debido a estas imperfecciones y al escaso interés por lo comercial de su creador, sirvió de inspiración para máquinas posteriores y supuso todo un hito en la historia de las calculadoras. De hecho, el cilindro de Leibniz se utilizaría como «motor de cálculo» en casi todas las máquinas calculadoras mecánicas posteriores. Puede observarse una réplica de la máquina original de Leibniz en la Figura 2.5

La primera máquina de cálculo que realmente alcanzó una verdadera popularidad fuera del ámbito científico y que gozó de éxito comercial fue el «Aritmómetro» (*Arithmomètre*), desarrollado por el ingeniero y empresario francés Charles-Xavier Thomas de Colmar en 1822. Se trataba de una máquina basada en los mis-

Figura 2.3: Reproducción del reloj calculante de Schickard realizada según los bocetos originales del autor.

mos mecanismos que la de Leibniz (como el ya citado cilindro de Leibniz), pero con mejoras en los aspectos que fallaban en la original y alguna funcionalidad añadida como la puesta a cero de los valores de los registros. Por lo tanto, aunque no presentaba ninguna novedad sustancial desde un punto de vista matemático, el Aritmómetro más bien resultó de verdad relevante por su funcionalidad y facilidad de uso. En la Figura 2.6 puede observarse un ejemplo de Aritmómetro y apreciar el énfasis que puso su autor en la estética y en la sencillez de uso para el gran público, en especial si se compara con la mayoría de máquinas calculadoras anteriores. A partir de la publicación de esta máquina habría un crecimiento exponencial durante todo el resto del siglo XIX y hasta principios del XX en el número de inventos desarrollados en el ámbito de las calculadoras matemáticas analógicas, aunque casi todas ellas realizaban las mismas funciones que el Aritmómetro, pero con algunas utilidades añadidas como la entrada de valores por teclado, la salida impresa en papel (dando origen de las máquinas registradoras), etc. [5]

Toda esta línea de invenciones en calculadoras convergería tiempo después con los avances en el campo de la electricidad y con el nacimiento de la electrónica, así como con las ideas del matemático inglés Charles Babbage, quien planteó pero no llegó a construir del todo una «máquina analítica» (*Analytical Engine*) que contaba ya con todos los elementos que componen la estructura propia de un ordenador moderno: un sistema de entrada de datos, otro de salida para los resultados, una memoria que almacenase los cálculos intermedios o finales y una unidad de proceso aritmético (ésta sí que fue construida y se conserva un ejemplar que puede contemplarse en la Figura 2.7). Además, la máquina hubiera sido capaz de ser programada mediante tarjetas perforadas y, de hecho, Ada Byron, amiga

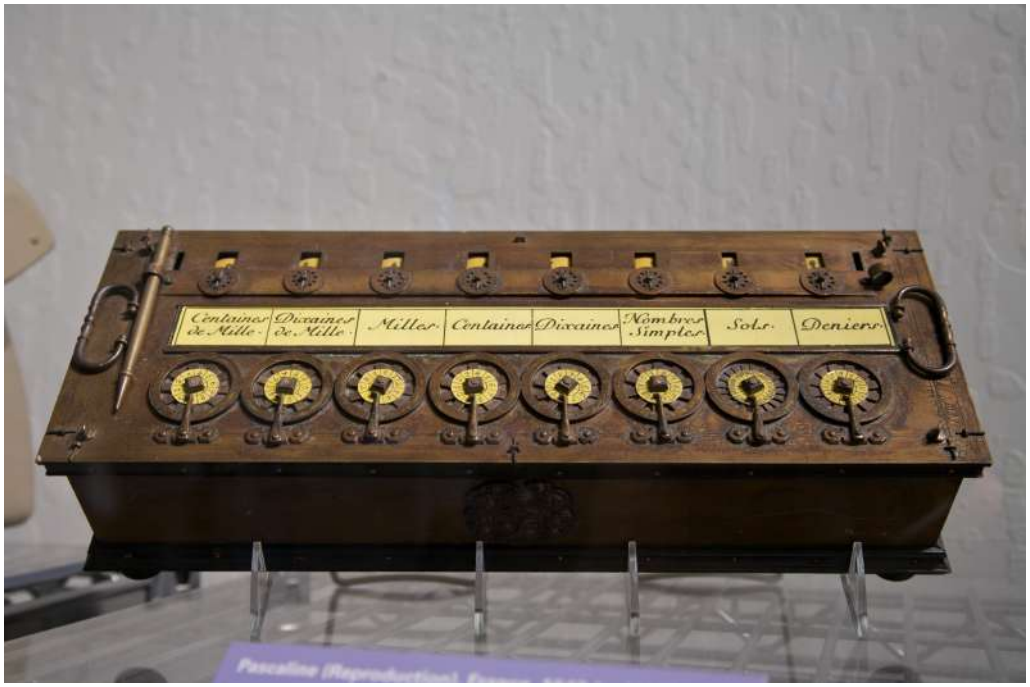

Figura 2.4: Una reproducción de la Pascalina de Blaise Pascal. Pueden observarse los mecanismos de entrada de datos a través de las ruedas de la parte inferior, así como la representación del resultado en los pequeños agujeros de la parte superior.

de Babbage, llegó a diseñar y publicar algunos programas para dicha máquina, razón por la que es considerada como la primera programadora de la historia.

De la convergencia de todas estas sendas de desarrollo tecnológico, así como de la necesidad de una gran potencia de cálculo que decantase el conflicto bélico por parte de las potencias combatientes durante la Segunda Guerra Mundial, nacieron las primeras computadoras propiamente dichas de la historia, como la Zuse, la Mark I o la ENIAC (*Electronic Numerical Integrator And Computer*). Puede observarse una imagen de esta última en la Figura 2.8. No obstante, todo ello queda ya fuera del alcance de este capítulo.

2.2 El ábaco a través de los tiempos

Los orígenes de los distintos instrumentos que hoy en día englobamos bajo el concepto «ábaco» yacen en las técnicas de conteo y cálculo mediante pequeñas piedras [5, 6, 4, 2]. Esta técnica, usada sin más refinamiento, suponía el uso de grandes cantidades de guijarros, hasta el punto de que podía resultar inmanejable en la práctica para números medianamente elevados.

La solución a este problema consistió en la creación de unas tablas con marcas y símbolos con valores asociados. Sobre ellas se colocaban unas piedras pequeñas u otros objetos que sirviesen como cuentas. Dependiendo del punto de la tabla sobre el que se depositasen las cuentas y de lo que las marcas y símbolos señalasen para ese punto, las cuentas pasaban a representar uno u otro valor. Este instrumento, al que los autores en inglés se suelen referir como *counting table* (tabla de cuentas o de cálculo) es considerado por algunos como un ábaco primitivo

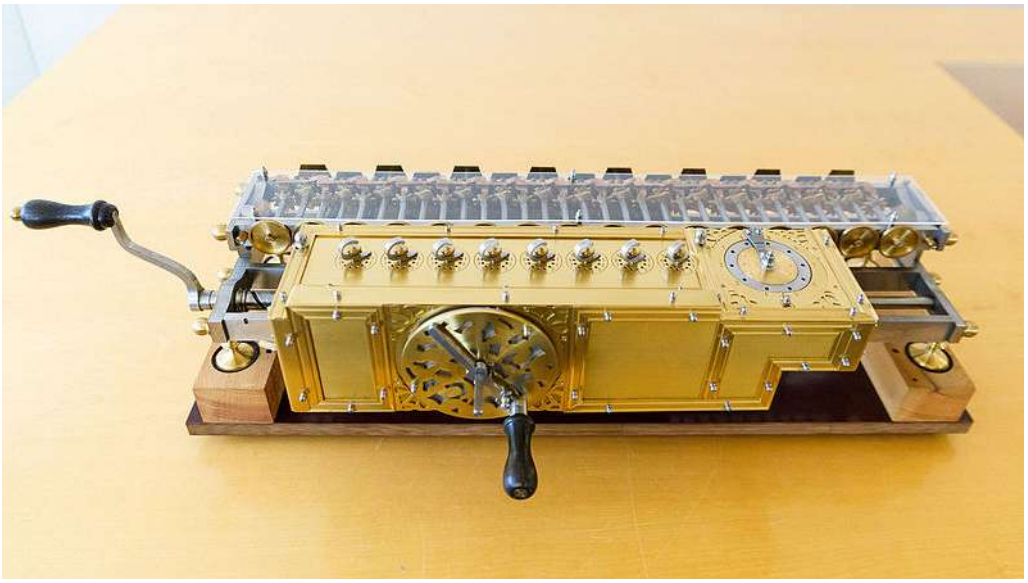

Figura 2.5: Reproducción de la máquina calculadora de Leibniz.

[6], mientras que otros parecen considerarlo como un predecesor directo, pero no como un ábaco propiamente dicho [2].

Se cree que esta invención pudo tener su origen en Sumeria con el auge de la civilización, la agricultura y las primeras ciudades y mercaderes, que tal vez crearan sus primeros ábacos realizando dibujos en la arena y depositando sobre ella piedras (Pullan cree que la palabra ábaco pudo ser adoptada por los griegos desde el término semita *abaq*, que significa arena o polvo) [11]. Hay que insistir en que esto es una conjetura, pues no ha llegado ningún ábaco sumerio hasta nuestros días, presumiblemente por estar hechos de materiales perecederos o poco resistentes, tal vez madera o arcilla. Ifrah especula con que, de acuerdo con el sistema de numeración sumerio de base sexagesimal, estos ábacos podían estar formados por columnas que representarían de forma alterna múltiplos de 10 y de 60, más o menos del siguiente modo: 1, 10, 60, $10 - 60$ (diez veces sesenta), 60^2, etc. [6]

Este tipo de ábaco llegó, a través de intercambios culturales y comerciales, a Grecia y de allí a Roma. Fueron estas civilizaciones quienes le dieron el nombre de *abax* o *abakon* en griego y de *abacus* en latín, en ambos casos traducible como tabla o tablilla [11, 2, 6]. El primer ábaco griego del que se tiene constancia es la tabla o ábaco de Salamina, datado alrededor del siglo V a.C. y hallada cerca de la isla de Salamina en el siglo XIX. Se trata de una pieza rectangular de mármol blanco de 149 por 75 centímetros con dos grupos de cinco y once líneas horizontales paralelas divididas cada una de ellas en dos partes iguales por sendas líneas verticales, de tal modo que forman rectángulos abiertos por el lado exterior de igual tamaño. Además, un conjunto de símbolos en griego antiguo rodea la tabla, representando distintas cantidades de unidades monetarias griegas de la época. Según Ifrah, los rectángulos del conjunto mayor de líneas representarían distintos múltiplos de 10 de talentos y dracmas, las dos monedas griegas más valiosas (en ese orden), mientras que la parte más pequeña serviría para contar cantidades de monedas menores como óbolos o *chalkoi* [6]. Se recomienda ver la Figura 2.9 para

Figura 2.6: Aritmómetro de Thomas de Colmar. Como puede observarse, la mayor diferencia con respecto a su antecesora creada por Leibniz es la elegancia y estética del Aritmómetro, al estar éste orientado a la venta al público, mientras que la construcción de la máquina de Leibniz solo respondía a una motivación de curiosidad científica.

una mejor comprensión de cómo debía ser el funcionamiento de esta herramienta.

También los romanos, así como los etruscos antes que ellos, hicieron uso de estos ábacos en forma de tabla. Uno de los modelos era de hecho una mesa con diversas secciones marcadas con símbolos numerales romanos, pero solo aquellos que eran múltiplos 10: I, X, C, M, etc. Sobre estas incisiones se colocarían los *calculi* para representar los números deseados y realizar las operaciones aritméticas. Por los escritos de autores como Plutarco o Apuleyo también se conoce que existían versiones portátiles de estos ábacos, consistentes en tablas vacías cuya superficie se cubría con arena o cera, y sobre estos materiales se dibujaba el área de cálculo deseada y se depositaban las cuentas para proceder a realizar las operaciones.

Más interesante resulta lo que Ifrah y Pullan llaman la «primera calculadora de bolsillo» [11, 6], también de origen romano. En realidad se trata del primer ábaco conocido en el sentido moderno del término, es decir, una estructura o marco relativamente pequeño con cuentas integradas que se pueden desplazar sobre él para representar números. Este objeto estaba hecho de metal y constaba de una estructura con nueve líneas paralelas hechas mediante incisiones en el metal, todas ellas divididas en una línea superior más corta y otra inferior, de mayor longitud, con la excepción de la primera línea por la derecha que solo tendría parte inferior. Las incisiones inferiores contarían con cuatro piezas móviles y las de la superior solo con una. De derecha a izquierda, la primera línea, que en ocasiones aparece dividida en tres incisiones pequeñas, correspondería a fracciones de onza (*unciae*, equivalente a 1/12 parte de as, la moneda romana de la época). La segunda, normalmente señalada con el símbolo θ, permitiría contar onzas hasta 10, y el resto de columnas son las que corresponderían al cálculo numérico general, cada una de ellas asociada a un múltiplo de 10: I, X, C, M, etc. En cada columna, las cuatro cuentas de la parte inferior estarían dotadas de un valor base de 1 cada una de ellas, mientras que la superior valdría 5. De este modo, en la primera columna se podría representar desde 1 hasta 9, en la segunda de 10 a 90, etcétera, así como

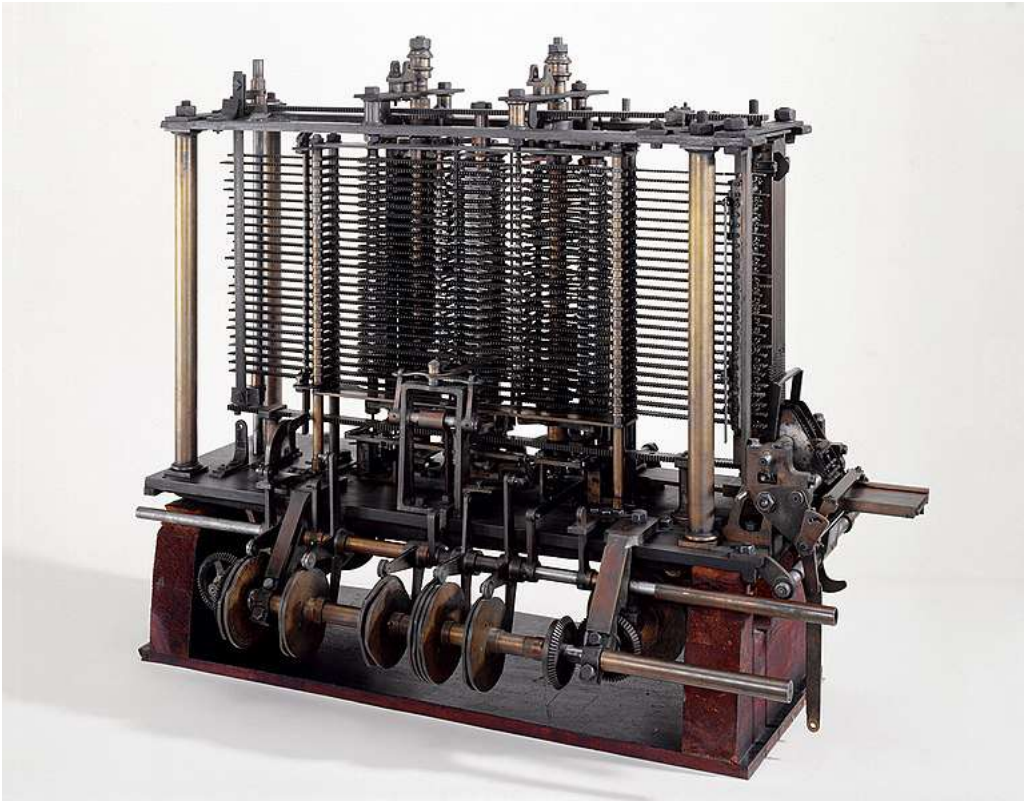

Figura 2.7: Pieza de la incompleta Máquina analítica de Charles Babbage; esta pieza se correspondería con lo que su autor llamaba el «molino» (*mill*), es decir, la unidad aritmético-lógica de la máquina.

combinar las columnas para representar cualquier número dentro del rango que permitiese el ábaco. Sorprendentemente, si se descarta la opción de las *unciae*, el diseño y comportamiento del ábaco romano de bolsillo es casi idéntico al del ábaco japonés o *soroban*, que estudiaremos más adelante con detenimiento. Todo ello puede verse en la reproducción de este tipo de ábaco mostrada en la Figura 2.10. Sin embargo, y pese a la aparente utilidad y podría decirse incluso modernidad de este instrumento, su relevancia en el mundo romano parece haber sido muy escasa frente a la mesa de cálculo, y parece que cayó en el olvido en Europa tras la caída de Roma.

Durante la Edad Media el desarrollo de los ábacos fue mínimo. Todos los comerciantes, recaudadores de impuestos y demás profesiones que requiriesen de la realización de operaciones matemáticas de cierta entidad seguían usándolo, por supuesto, pues con el sistema numeral romano en vigor, de naturaleza no posicional, no era posible realizar cálculos complejos de forma escrita. El ábaco que se usaba para dichos cálculos era el romano de mesa con ligeras variaciones (este instrumento se puede apreciar en la Figura 2.11), que requería de un profundo conocimiento y entrenamiento para ser capaz de realizar operaciones relativamente complejas como la multiplicación y la división, lo que llevaba a muchos comerciantes a verse obligados a contratar especialistas que les realizasen los cálculos [6]. El único cambio sobre este sistema fue introducido por el monje Gerbert de Aurillac, quien años más tarde se convertiría en el Papa Silvestre II. Este religioso conoció el sistema numeral árabe de origen indio poco antes del año 1000 en un viaje a *Al-Andalus*, y trató de introducirlo en la Europa cristiana con escaso éxi-

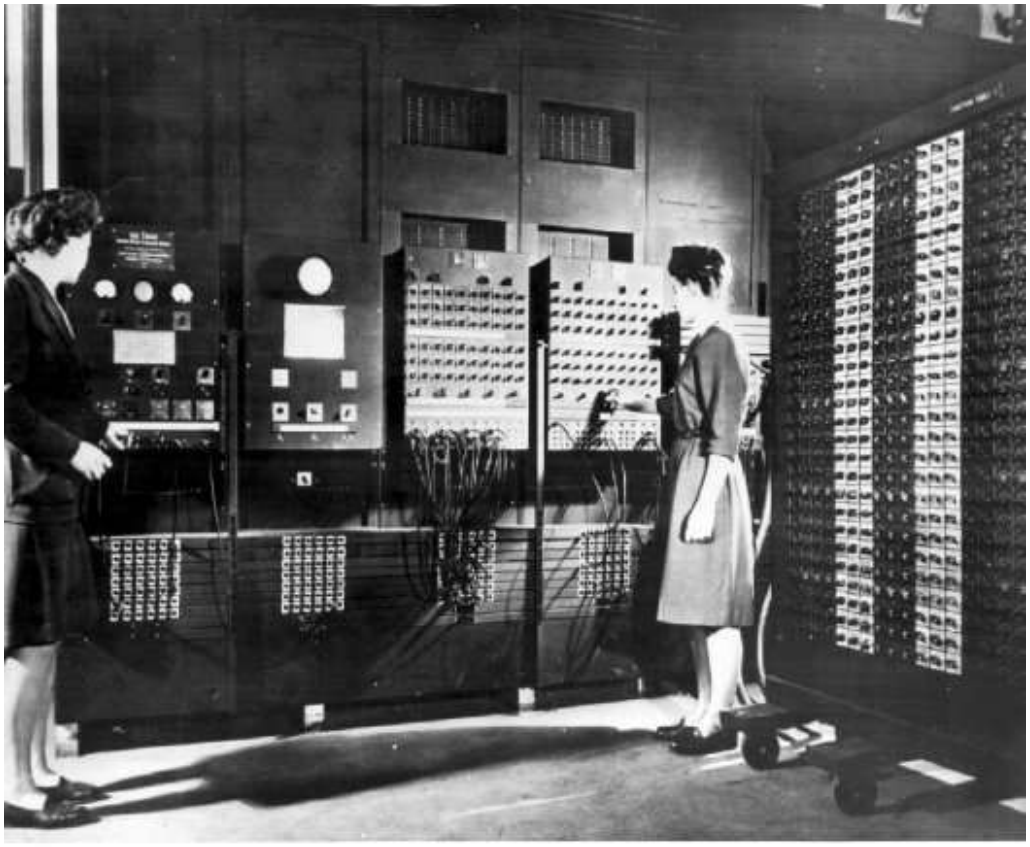

Figura 2.8: Algunas de las primeras programadoras de la historia trabajando con la ENIAC. Aunque su historia ha quedado en muchas ocasiones en segundo plano, fueron seis mujeres matemáticas las encargadas de la programación de esta máquina.

to, pues no sería hasta después de las cruzadas, allá por el siglo XII, cuando se comenzaría a popularizar de forma muy gradual el nuevo modelo numérico. Lo que sí logró Gerbert de Aurillac fue implementar un pequeño cambio en el ábaco de mesa romano: sustituir las pequeñas piedras que se usaban sobre el tablero (cada una con un valor base de uno) por unas piezas numeradas con los símbolos árabes del uno al nueve, lo cual permitía reducir enormemente el número de cuentas sobre la mesa y simplificar un poco el uso del instrumento.

Con la introducción y gradual generalización de los números árabes se generó lo que se llamó el enfrentamiento entre «algoristas» y «abacistas». Los algoristas defendían el uso del sistema de numeración árabe y el cálculo en papel, mientras que los abacistas se decantaban por la tradicional numeración romana y el uso de los ábacos para el cálculo aritmético. Como se puede suponer, el peso de la historia se decantó a favor de los algoristas tras el fin de la Edad Media, y el ábaco de tipo mesa fue desapareciendo de forma progresiva de Europa. Una imagen que describe a la perfección esta controversia entre algoristas y abacistas se muestra en la Figura 2.11. No obstante, quedaron algunos restos culturales durante siglos, como la figura del *exchequer* en Inglaterra, el alto funcionario encargado de las cuentas del tesoro, que tomaba el nombre del ábaco de tipo mesa que utilizaba originalmente para llevar a cabo la contabilidad.

En la América precolombina, por su parte, se originó un sistema de cuenta completamente original basado en cuerdas de colores y nudos. Este método se

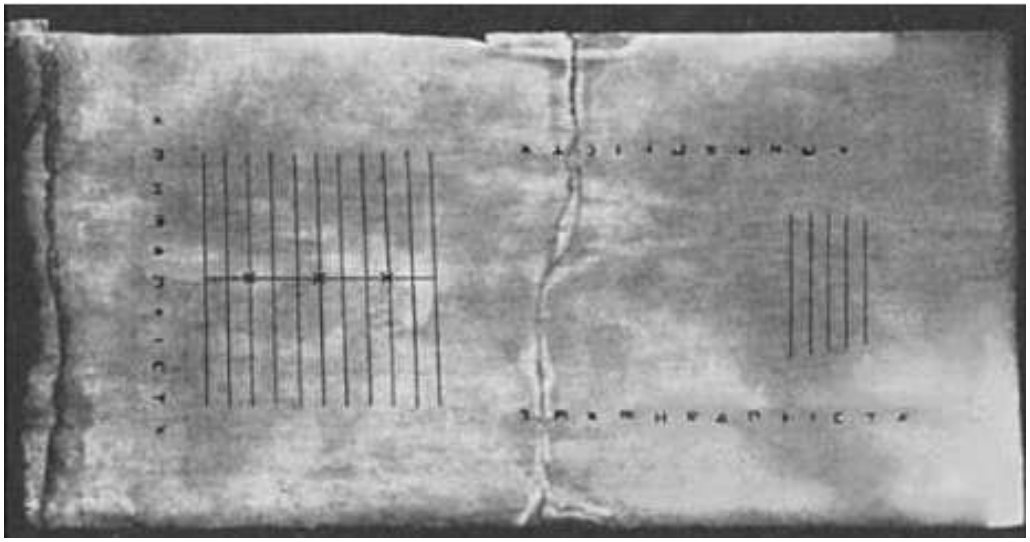

Figura 2.9: Tabla de Salamina, datada alrededor del siglo V a.C. De origen griego, es considerada como el ábaco o tabla de cálculo conservada más antigua de la historia. En la imagen pueden observarse tres elementos, dos de ellos conjuntos de líneas donde el operador colocaría las piedras para representar cantidades monetarias y hacer cálculos. El tercer elemento son los símbolos que aparecen alrededor de la tabla, y que indicarían los valores numéricos o monetarios atribuidos a los distintos elementos del tablero

desarrolló en el imperio inca, aunque pervivió mucho tiempo después de su desaparición entre las comunidades indígenas de la zona. Los *quipus*, pues así se llamaban, constaban de una cuerda horizontal de la que colgaban varias cuerdas anudadas sobre sí mismas con varios nudos. Cada una de estas cuerdas colgantes representaba un número, mientras que cada uno de los nudos equivalía a uno de los dígitos que componían el número completo. Los incas usaban un sistema decimal, por lo que cada uno de los nudos representaba las unidades, las decenas, las centenas, etc. El orden creciente iba de abajo hacia arriba, por lo que el nudo más bajo era el que representaba las unidades. A su vez, el valor de cada nudo dependía de su tamaño: un nudo simple equivalía a un uno (o 10, o 100...), uno doble a un dos, etc. Además, cada una de las cuerdas podía ser de un color, lo que proporcionaba un significado u otro al número representado: ganado, tributos, censos... No obstante, y aunque este sistema era muy eficiente, sencillo y organizado para guardar registros de valores numéricos, su utilidad como herramienta de cálculo parece bastante limitada. No obstante, se cree que los incas pudieron usar también ábacos de tipo mesa de cálculo llamados *yupanas* que tendrían un funcionamiento muy similar a los de Europa y Oriente Medio. En la Figura 2.12 puede observarse tanto un ejemplo de *quipu*, sostenido por la figura central, como una *yupana* que aparece en la parte inferior del grabado.

También durante la Edad Media, pero en China, se desarrolló un ábaco de piezas integradas desplazables. Este ábaco, llamado *suanpan*, *suan-pan* o *suan pan* (que significa tabla de cálculo en chino), resulta de especial interés por perdurar en uso hasta hoy en día, siendo probablemente el más antiguo aún en vigor, y por poseer ya todos los elementos que componen un ábaco moderno. Algunos autores defienden que el *suanpan* fue una evolución del ábaco romano que Ifrah llamaba «de bolsillo», que habría sido llevado a China a través de contactos comerciales [11]. No obstante, no hay pruebas definitivas que lo prueben, aunque

Figura 2.10: Ábaco romano de bolsillo. En la figura pueden apreciarse las dos columnas más a la derecha, diferentes al resto y que se usarían para realizar cálculos con *unciae*, las monedas de menor valor. El resto de columnas (I, X, C, etc.) representan potencias ascendentes de derecha a izquierda de 10, lo que da a este ábaco una capacidad de cálculo similar a la de ábacos orientales modernos como el *soroban*

el mecanismo de ambos instrumentos sea sospechosamente similar. Lo que sí se conoce es que el *suanpan* data al menos del siglo XII después de Cristo, cuando aparecen las primeras referencias en la literatura, aunque se piensa que su uso puede ser bastante anterior.

El ábaco chino o *suanpan* se compone de un marco rectangular de madera dividido en dos regiones separadas por una madera central, una mayor en la parte inferior y otra más pequeña en la zona superior. Atraviesan ambos espacios unas varillas, tradicionalmente de metal. Sobre cada una de estas varillas que quedan divididas también en dos partes por el separador central, se deslizan un total de siete cuentas de madera, cinco de ellas en la parte inferior y dos en la superior. Normalmente los ábacos chinos cuentan con 12 varillas, pero el número puede ser mayor, lo que simplemente implicaría una capacidad para representar y operar con números de mayor magnitud.

El funcionamiento del *suanpan* es el siguiente: las cinco cuentas de abajo representan una unidad cada una, mientras que las dos de arriba equivalen a cinco unidades, al igual que sucedía en el ábaco romano «de bolsillo». Esto significa que en cada varilla se puede representar cualquier número entre el cero y el 15. A su vez, si se opera en base decimal, cada columna tiene asociado un exponente de 10, el cual es incremental en el sentido derecha-izquierda (al operar, el *suanpan* se sitúa en posición horizontal). Generalmente las dos varillas más a la derecha se reservan para la parte decimal del número, de tal modo que las unidades se situarían en la tercera columna, y de derecha a izquierda y en base 10 los exponentes serían: 10^{-2}, 10^{-1}, 10^{0}, 10^{1},10^{2}, etc. No obstante, al ser todas las varillas iguales entre sí, si las necesidades imponen trabajar con enteros muy grandes o con números fraccionarios de muchos decimales, la varilla de las unidades podría desplazarse a izquierda o a derecha sin problema. La forma de operar con-

siste simplemente en deslizar las cuentas hacia la madera central para añadirlas al número representado o en alejarlas si se quiere dejar de contarlas o restarlas.

El ábaco chino presenta algunas redundancias que pueden resultar confusas, como por ejemplo que el número 15 pueda representarse hasta de tres formas: con todas las cuentas de la varilla de las unidades deslizadas hacia la madera central, con una cuenta de la parte inferior de las decenas y una de la parte superior de las unidades, o con una cuenta de la parte inferior de las decenas más las cinco de la zona inferior de las unidades. Esto se debe a que el ábaco cuenta con más piezas de las estrictamente necesarias para operar en base decimal, pero esto puede ser a su vez de ayuda en algunas técnicas de cálculo de acarreos o para operar en bases distintas a la decimal, y resulta especialmente adecuado para hacer cálculos en hexadecimal. Puede verse un ejemplo de *suanpan* en la Figura 2.13.

Algo más tardío es el ábaco ruso o *schoty*, mencionado por primera vez en 1658, y probablemente adaptado a partir del ábaco chino [4]. Al igual que el *suanpan*, el *schoty* aún puede verse en algunos pequeños comercios de Rusia, usado como herramienta para realizar cálculos rápidos y fiables. El ábaco ruso, a diferencia del chino, se opera en posición vertical, es decir, moviendo las cuentas a lo largo de las varillas de forma horizontal. Se compone de varias filas, normalmente ocho o más, con diez cuentas en cada una, excepto una que solo tiene cuatro. Las dos cuentas centrales de cada varilla suelen ser de un color distinto, generalmente negras frente al resto, que son blancas, para facilitar una mejor identificación visual de la posición de cada una.

El *schoty* suele usarse con una base decimal, para la cual está diseñado al poseer diez cuentas por varilla. La excepción, la fila con solo cuatro piezas, está pensada para contar cuartos de unidad (1/4, 2/4, 3/4 y 4/4), algo que resultaba útil para el antiguo sistema monetario ruso, donde un rublo equivalía a 100 *kopeks*, y un *kopek* a su vez a cuatro *polushki* [4]. El resto de varillas tienen, como es habitual, un exponencial de 10 asociado, donde el orden ascendente va de abajo hacia arriba, con la varilla de cuatro cuentas marcando el límite entre los números fraccionarios y los enteros. Para contar, las piezas han de desplazarse a lo largo de la varilla, que se encuentra ligeramente curvada para evitar deslizamientos involuntarios, en dirección de izquierda a derecha para sumar y de derecha a izquierda para restar. Puede verse un ejemplo de *schoty* en la Figura 2.14.

El ábaco japonés o *soroban* (de hecho *soroban* es la palabra para referirse al propio concepto de ábaco en japonés) deriva del *suanpan*, introducido en Japón alrededor del año 1600, probablemente tras la invasión japonesa de Corea [2]. Parece ser que conservó su forma original hasta mediados del siglo XIX, cuando perdió una de las dos cuentas de la parte superior. Hacia 1920 se suprimió también una de las piezas de abajo, con lo que el *soroban* obtuvo su forma actual [7]. Estos cambios eliminaron las redundancias presentes en el *suanpan*, lo que lleva a autores como Ifrah a considerar el *soroban* como «el estado de perfección total del instrumento y marca el cierre a la evolución de las técnicas de cálculo derivadas del manejo aritmético de piedras» [6]. En efecto, el ábaco japonés es un instrumento más simple y menos redundante que el chino, y presenta la ventaja frente al *schoty* de ser más compacto y con menos cuentas, lo que permite hacer cálculos con movimientos más cortos, rápidos y precisos. Debido a todo ello, no es de extrañar que en una competición organizada en Japón en 1945 entre un maestro

japonés del ábaco y un oficial del ejército estadounidense experto en el uso de la calculadora electromecánica, el primero resultara vencedor por cuatro a uno, siendo las cinco pruebas las siguientes: suma, resta, multiplicación, división, y una ronda de operaciones mixtas (en la obra de Kojima [7] aparecen detalladas las pruebas y los tiempos, que resultan bastante sorprendentes por la gran ventaja del *soroban* en la suma y la resta, así como por el número de errores cometidos, significativamente menor). Aún a día de hoy se enseña el uso del ábaco en las escuelas de Japón y se organizan competiciones que son verdaderos prodigios de rapidez y precisión en el cálculo [1].

El *soroban*, como ya se ha adelantado, es muy similar al *suanpan* y, sorprendentemente, al ábaco romano de bolsillo. Consta de un número variable de varillas, separadas entre una parte superior y otra inferior. Normalmente suele haber una marca en el tablero cada tres varillas para que el operador sitúe las distintas magnitudes de un modo más visual, y también se suele usar uno de estos indicadores para señalar la varilla de las unidades, de tal modo que a su izquierda queden las decenas, centenas, etc. y a su derecha los decimales (cabe indicar que el *soroban* se utiliza en posición horizontal). El valor de las cuentas es igual que el indicado para el *suanpan*: uno para las piezas inferiores y cinco para las superiores. De esta forma, en cada varilla pueden representarse valores desde el cero, cuando todas las cuentas están separadas de la madera central, hasta el nueve, cuando todas están deslizadas hacia el centro. Puede verse un ejemplo de *soroban* en la Figura 2.15.

Para finalizar con este repaso a la historia del ábaco, cabe señalar que, aunque en el siglo XX el ábaco fue perdiendo protagonismo debido al auge de las calculadoras electrónicas, siguieron surgiendo muchos ejemplos de ábacos modernos que pueden resultar más o menos curiosos o útiles, aunque su éxito haya sido limitado. Por ejemplo, el ábaco francés inventado por Fernand Nathan fue bastante popular como instrumento didáctico en Europa occidental, aunque en realidad no fuese más que un ábaco de tipo ruso sin la varilla de las cuatro cuentas (es decir, que todas las varillas tenían 10 cuentas). En 1958 un hombre llamado Lee Kai-chen «diseñó» un ábaco que llevaba su nombre y publicó un manual explicando su uso. Aunque en realidad no se trataba más que de la unión de un *soroban* en la parte superior y de un *suanpan* en la inferior, su autor defendía que el nuevo sistema facilitaba el cálculo de operaciones complejas. No menos curioso es el ábaco «de transición» desarrollado en Japón y que integraba un *soroban* con una calculadora digital, y que recibió el no muy original nombre de Soro-cal [2]. También es digno de mención el ábaco de Cranmer, que no es más que un *soroban* diseñado para su uso por personas ciegas, ya que las piezas, hechas de goma, permiten al usuario palparlas sin que se deslicen de forma involuntaria. Un último ábaco que merece ser citado es el binario, en el que cada varilla posee una única cuenta con dos posiciones, 0 y 1, diseñado con el objetivo de hacer comprender a los estudiantes el funcionamiento interno de las computadoras. Véanse las Figuras 2.16 y 2.17 donde se muestran imágenes de algunas de estas curiosas invenciones.

[1] Muy interesante el siguiente vídeo (en francés) para observar la habilidad que pueden llegar a desarrollar en el uso de este instrumento: https://www.youtube.com/watch?v=lpg_UEvocE4.

2.3 Cómo realizar operaciones aritméticas

En esta sección se explica cómo ejecutar las cuatro operaciones de la aritmética básica mediante el uso de los ábacos. Para los ejemplos se utiliza el *soroban* o ábaco japonés, pero los algoritmos y formas de operar son fácilmente trasladables al *suanpan* y al *schoty*. De hecho, la traducción al ábaco chino es inmediata, pues no haría falta más que dejar sin utilizar una de las cuentas de la parte superior y otra de las de la parte inferior y operar como si tuviésemos un ábaco japonés de 4/1. Para operar con el ábaco ruso, por su parte, simplemente habría que actuar pensando que en lugar de una cuenta de valor cinco se poseen cinco cuentas de valor uno. Por lo demás, el comportamiento es idéntico, siempre y cuando no se tenga en cuenta la varilla de los cuartos de *kopek* del *schoty*.

Para el lector interesado en una descripción más detallada de las operaciones o en realizar ejercicios para ganar soltura en el manejo del ábaco, el libro de referencia en el uso del *soroban* es el de Kojima [7], extremadamente detallado y con multitud de ejercicios para la práctica. Aparte de éste, los textos [13, 1] contienen una explicación rápida pero clara sobre el manejo del *soroban*, mientras que [8, 15] contienen una explicación más detallada y con más ejemplos y diferentes algoritmos. Por último, el texto publicado por la Dirección de Educación Especial de México [16] resulta muy interesante por contener ejemplos tanto en *soroban* como en *suanpan* y en un ábaco de diez piezas, equivalente al *schoty*, mientras que la página web [14] contiene ejemplos sobre cómo realizar cálculos con *suanpan*.

Antes de empezar con las operaciones hay que destacar las normas que Kojima establece como esenciales para operar rápida y correctamente con un ábaco:

- En primer lugar, poner siempre a 0 el ábaco antes de empezar a operar (las cuentas de la parte superior empujadas hacia arriba y las de la parte inferior, hacia abajo).

- Operar siempre de izquierda a derecha, es decir, empezando por las cifras de mayor peso y continuar en orden descendente.

- Tener siempre en cuenta el número complementario en base 10 y también en base 5. Esto quiere decir que si tenemos un 4 representado con las cuatro cuentas de la primera varilla y queremos sumarle un 2, habrá que hacer el complementario de 2 con 5, es decir, $5 - 2 = 3$, y restar ese número de las cuentas de abajo a la vez que deslizamos la cuenta superior para añadir el 5. Lo mismo sucede cuando trabajamos con acarreo entre distintas columnas: supongamos que tenemos un 9 en las unidades y queremos sumarle 7. Puesto que el resultado excede de 10, deberemos calcular el complementario $10 - 7 = 3$, y retirar 3 cuentas de la columna de las unidades a la vez que añadimos 1 a las decenas (se puede ver este ejemplo en la Figura 2.18).

- Utilizar dos dedos para operar, el índice y el pulgar, pues está comprobado que es la forma más rápida de calcular.

2.3.1. Suma

Aunque en la breve explicación sobre complementarios ya se ha hecho una suma, aquí se realizará una descripción más detallada del proceso y se explicará cómo realizar paso a paso la suma de dos números de cierto tamaño.

El primer paso al realizar una suma en el *soroban* es representar uno de los dos números en el ábaco. Para el ejemplo que vamos a realizar aquí usaremos el 89 249 y el 234 772. Es indiferente cuál de los dos números decidamos representar en el tablero, así que elegimos por ejemplo el primero de ellos. Esta operación utiliza solo números enteros, pero la forma de proceder sería exactamente igual con decimales.

Los dígitos se suman de uno en uno según su posición. Al hacerlo, nos podemos encontrar con dos casos básicos: que haya acarreo (es decir, que la suma de ambos dígitos sea 10 o mayor) o que no lo haya. Si no lo hay, simplemente habrá que añadir a la columna en cuestión el valor necesario, tal vez restando en la parte inferior el complementario del valor sumado con respecto a 5 si el dígito inicial era menor que 5 y el resultante es mayor. Si hay acarreo, se restará a la columna donde se realizaba la suma el número complementario al sumado con respecto a 10 y a continuación se sumará una unidad a la columna inmediatamente a la izquierda .

Veámoslo con el ejemplo propuesto. En primer lugar se ha representado el número 89 249 en el *soroban* (ver la parte izquierda de la Figura 2.19). A continuación, comenzamos a sumar pares de dígitos desde la posición más a la izquierda, como dicta la regla. El primer número a sumar es un 2 en la posición de las centenas de millar, donde el número inicial tenía un 0 pues era de menor magnitud. Por lo tanto, lo único que hay que hacer es subir dos cuentas de la parte inferior. Continuamos con las decenas de millar, posición en la que tenemos un 8, y nos encontramos con que debemos sumar un 3, lo cual da un resultado superior a diez y por lo tanto conlleva acarreo, por lo que quitamos el complementario de 3 con respecto a 10 de la columna actual, es decir, restamos 7, subiendo la cuenta de la parte superior y bajando dos de la parte inferior, y a continuación sumamos uno a la varilla de la izquierda representando el acarreo.

El resto de la suma sigue los mismos criterios: sumamos $9 + 4$, que conlleva acarreo, por lo que restamos el complementario (6) y sumamos 1 a la columna de la izquierda. Al sumar $2 + 7$, no hay acarreos, así que simplemente deslizamos cuentas por valor 9. $4 + 7$ sí que implica acarreo, por lo que restamos el valor del complementario (3) y sumamos uno a la columna de la izquierda. Aquí nos encontramos con que la columna de la izquierda valía 9, por lo que el sumar 1 conlleva a su vez un acarreo encadenado. Calculamos el complementario de 1 (9), lo restamos y sumamos 1 a la columna de la izquierda. Por último, $9 + 2$ también conlleva acarreo, así que restamos el complementario de 2 (8) y sumamos uno a la izquierda. Comprobamos en el ábaco el número obtenido, 324 021 (ver la parte derecha de la Figura 2.19). Es correcto.

2.3.2. Resta

El proceso de la resta es muy similar al de la suma, y una vez comprendido el primero resulta muy sencillo comprender el segundo. De nuevo, el primer paso es representar uno de los números en el *soroban* (en este caso necesariamente deberá ser el mayor de los dos, ya que no existe la opción de mostrar resultados negativos). Un vez hecho, de nuevo se procede de izquierda a derecha a restar los dígitos uno a uno. Cuando el dígito del minuendo sea mayor que el del sustraendo, simplemente se retiran las cuentas necesarias en la varilla en cuestión. Cuando suceda lo contrario, significará que hay un acarreo negativo. En tal caso se resta uno a la columna inmediatamente a la izquierda y a continuación se calculará el complementario en la columna actual y se le sumará dicho valor. Como puede observarse, el proceso es el inverso al de la suma, como se podía suponer.

Probemos con la resta $16\,532 - 8\,455$. En primer lugar representamos el minuendo en el *soroban* (ver la parte izquierda de la Figura 2.20), y procedemos a aplicar el algoritmo descrito. La columna de las decenas de millar solo está presente en el primer número, pero no en el segundo, por lo que no actuamos sobre ella. A continuación, en los millares encontramos un $6 - 8$ que implica un acarreo negativo, por lo que restamos 1 de la columna a la izquierda, que se queda a 0, mientras que en la que estábamos trabajando hallamos el complementario de 8, que es 2, y lo sumamos al valor anterior, 6, de tal modo que nos queda un 8. Continuamos por la siguiente varilla, en la que nos encontramos con la resta $5 - 4$, que no conlleva acarreo, así que simplemente representamos el resultado (1) en la columna. La siguiente operación es $3 - 5$, así que restamos 1 de la columna a la izquierda y en la varilla actual sumamos el complementario de 5 (5). Por último, la operación $2 - 5$ también conlleva acarreo, así que sumamos de nuevo el complementario (5) y restamos uno de la columna a la izquierda. Comprobamos el resultado, 8 077 (ver la parte derecha de la Figura 2.20). Es correcto.

2.3.3. Multiplicación

Para afrontar la multiplicación lo primero que hay que señalar es que el *soroban* es una herramienta de ayuda al cálculo pero, al contrario que la calculadora digital, no puede ayudar a hacer multiplicaciones a alguien que desconozca las reglas básicas de la operación, es decir, la tabla de multiplicar.

El proceso de realizar una multiplicación requiere alguna preparación más que las operaciones de suma y resta. En este caso deberemos representar en el *soroban* los dos factores. En primer lugar se realiza una suma del número de dígitos del multiplicador y del multiplicando y se sitúa el multiplicando a esa distancia de la columna de las unidades y el multiplicador a la izquierda del multiplicando, dejando un par de columnas vacías entre ellos de modo que se distingan con claridad. Esta preparación se realiza para dejar un espacio suficiente para el resultado en la varilla de las unidades y las columnas contiguas. A continuación se procede a realizar la multiplicación dígito a dígito.

Realicemos por ejemplo la operación 437×28. La configuración inicial sería la de la Figura 2.21. Puesto que hay que seguir siempre el orden de izquierda a derecha, multiplicamos el 2 del multiplicador por el 7 del multiplicando, y situa-

mos el resultado en las columnas de las centenas y las decenas. A continuación multiplicamos el mismo 7 por el 8 del multiplicador y lo situamos en las decenas y en las unidades, realizando la suma con el valor antes calculado. Pasamos al 3 del multiplicando y calculamos 3×2 en las columnas de los miles y de las centenas. Proseguimos con 3×8, cuyo resultado va a las columnas de las centenas y las decenas, y puesto que esta suma conlleva acarreo, lo realizamos tal y como se explicó en los apartados anteriores. Proseguimos con el 4×2, cuyo resultado escribimos en las columnas de las decenas de miles y los miles. Por último, calculamos 4×8 y lo escribimos entre los miles y las centenas. Una vez tenemos el resultado borramos los factores (pueden ir borrándose a medida que se va realizando la operación y no son necesarios) y leemos el resultado. El número resultante es 12 236 (se puede observar en la parte derecha de la Figura 2.21), que es el producto correcto.

2.3.4. División

En la división, como en la multiplicación, necesitamos representar el dividendo y el divisor sobre el *soroban*. En este caso situaremos el dividendo en la posición de las unidades y en las casillas contiguas a la izquierda, mientras que el divisor lo representaremos en el extremo izquierdo del ábaco (ver la disposición en la parte izquierda de la Figura 2.22).

Para el ejemplo vamos a realizar la división $552 \div 23$, por lo que organizamos el *soroban* tal y como muestra la Figura 2.22. Comenzamos a dividir, como siempre de izquierda a derecha. En primer lugar dividimos el primer 5 del dividendo por el primer 2 del divisor, y obtenemos un cociente 2, que lo anotamos dos columnas más a la izquierda de la última varilla del dividendo. A continuación, multiplicamos el cociente, 2, por el primer número del divisor, 2, y el resultado lo restamos al primer valor del dividendo, 5, lo cual resulta en $5 - 4 = 1$. Repetimos el mismo proceso multiplicando el cociente por el segundo elemento del divisor y restándolo a la columna de las decenas, es decir, $5 - (3 \times 2)$, lo cual conlleva un acarreo negativo que, tras realizar la resta, termina con un 92 como dividendo. Volvemos a comenzar el proceso, dividiendo $9 \div 2$, con lo cual ponemos el cociente resultante (4) a la derecha del que obtuvimos antes. Multiplicamos el 2 del divisor por el 4 del cociente y el resultado lo restamos del 9 de las decenas del dividendo. Para finalizar, calculamos 4×3 (cociente por segundo elemento del divisor), y lo restamos de las unidades del dividendo. Donde fuimos colocando los cocientes tenemos el resultado, 24, que es correcto (ver parte derecha de la Figura 2.22). Si la división no hubiera sido exacta, nos hubiera quedado en el lugar del dividendo un número menor que el divisor que se correspondería con el resto.

Figura 2.11: Ilustración de autor desconocido de la obra *Margarita philosophica* (1503) de Gregor Reisch. El grabado muestra a la perfección el conflicto que tuvo lugar a finales de la Edad Media entre el nuevo sistema de cálculo escrito y el antiguo de la tabla de cálculo. Además, sirve como ilustración de cómo era y cómo se operaba con una tabla de cálculo medieval

Figura 2.12: Ilustración de la obra *El primer nueva crónica y buen gobierno*, crónica escrita por un noble indígena peruano hacia el año 1615. El grabado muestra como figura principal a un indígena sosteniendo un *quipu* formado por multitud de cuerdas, mientras que en la esquina inferior izquierda aparece una *yupana*, el ábaco de tipo tabla de cálculo utilizado en la cultura inca

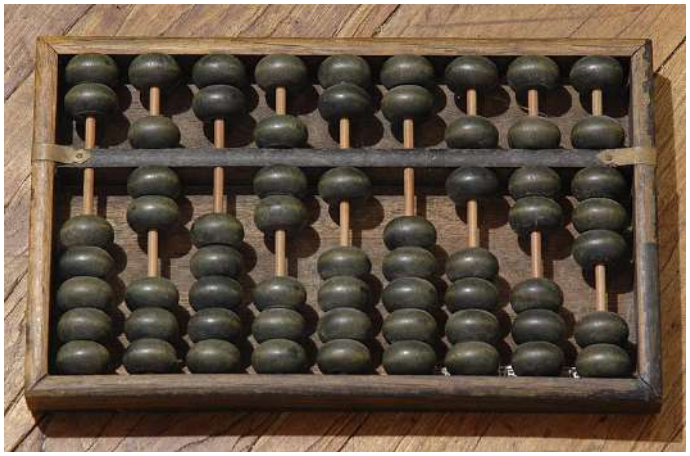

Figura 2.13: Ejemplo de *suanpan* o ábaco chino tradicional. En este caso, suponiendo que se usen las dos últimas cuentas para uso de la parte decimal, el número representado sería el 707 106,78

Figura 2.14: Ejemplo de *schoty* o ábaco ruso tradicional. En la imagen puede apreciarse la distribución y el color típico de las cuentas de un ábaco ruso. Las cuentas deslizadas completamente a la derecha indican que el ábaco está a cero. También puede apreciarse la ligera curvatura de las varillas, que dificulta los desplazamientos involuntarios de cuentas y los errores de cálculo que ello implicaría

Figura 2.15: Ejemplo de *soroban* o ábaco japonés. En este caso se trata de una fabricación de bajo coste en plástico, realizada para la venta masiva, lo cual es significativo de la popularidad de la que gozan aún la actualidad estos instrumentos. En este caso, y suponiendo que las dos últimas varillas se usen para representación de la parte decimal, el número representado es el 9 118,23

Figura 2.16: Ábacos binarios creados de forma artesanal a partir de ábacos chinos por el dr. Robert C. Good, Jr. para la enseñanza de conceptos informáticos.

Figura 2.17: Un *soroban*/calculadora digital (Soro-cal) comercializado en Japón

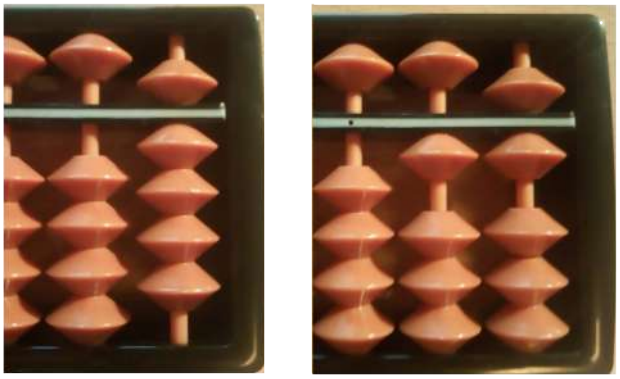

Figura 2.18: Operación de suma $9 + 7 = 16$. El proceso, que aquí se muestra de izquierda a derecha, consiste en lo siguiente: si comprobamos que la suma va a resultar en un número mayor de 10, restamos el complementario de la suma con respecto a 10 de la columna actual, en este caso $10 - 7 = 3$, por lo que retiramos 3 cuentas de la parte inferior en la columna de las unidades, e inmediatamente a continuación procedemos a sumar 1 a la columna inmediatamente a la izquierda, representando el acarreo

Figura 2.19: Operación de suma $89\,249 + 234\,772 = 324\,021$. En la imagen izquierda se ve el número inicial, y en la derecha el resultado tras aplicar el algoritmo de suma descrito en el texto

Figura 2.20: Operación de resta $16\,532 - 8\,455 = 8\,077$. En la imagen izquierda se ve el número inicial, y en la derecha el resultado tras aplicar el algoritmo de resta descrito en el texto

Figura 2.21: Operación de multiplicación $437 \times 28 = 12\,236$. En la imagen izquierda se ve la disposición inicial de multiplicando y multiplicador, y en la derecha el resultado tras aplicar el algoritmo del producto descrito en el texto

Figura 2.22: Operación de división $552 \div 23 = 24$. En la imagen izquierda se ve la disposición inicial de dividendo y divisor, y en la derecha el resultado tras aplicar el algoritmo de división descrito en el texto

CAPÍTULO 3
El lenguaje de programación Scratch

En primer lugar, y ya que Scratch ha sido el lenguaje (y entorno, puesto que como se verá más adelante, los dos son indisociables) elegido para el desarrollo del apartado de programación de este trabajo, cabe preguntarse, ¿qué es exactamente Scratch? Y además, ¿por qué elegirlo frente a otros cientos de lenguajes? Estas son las cuestiones que se intentarán abordar en el primer punto de este capítulo, para a continuación hacer una breve introducción al funcionamiento y manejo básicos del lenguaje y entorno Scratch.

3.1 Scratch, un lenguaje diferente

Podría definirse Scratch como un lenguaje de programación visual y educativo. La característica «visual» hace referencia tanto a la forma de escribir el código (de hecho tal vez sería más correcto decir montar o ensamblar) como a los resultados generados en la escena. Mientras en un lado de la pantalla el usuario puede ir encajando su código de un modo visual e intuitivo gracias a los patrones de colores y formas de los distintos tipos de instrucciones, en la otra mitad puede observar en la escena los resultados producidos por las órdenes en tiempo real. En la Figura 3.1 puede observarse esta forma de trabajar propia del entorno.

La idea básica de Scratch es presentar toda la sintaxis de un lenguaje de programación (estructuras de control, operadores, funciones de atención a eventos...) de un modo visual y eliminando las dificultades del proceso de escritura de código propiamente dicho, como podrían ser los errores léxicos, sintácticos o semánticos que suelen producirse durante el mismo. De este modo se evita uno de los aspectos que podrían resultar más desagradables durante el primer aprendizaje de la programación.

Por otra parte, el escenario en la que se combinan los objetos (*sprites*), las escenas (*stages*) y los programas (*scripts*) asociados a ambos proporciona en todo momento una retroalimentación visual de lo que se está haciendo. Además, esta orientación hacia la producción de resultados visuales también resulta mucho más atractiva a los niños, niñas y adolescentes que dan sus primeros pasos en la programación. Como argumentaban los propios desarrolladores de Scratch, la

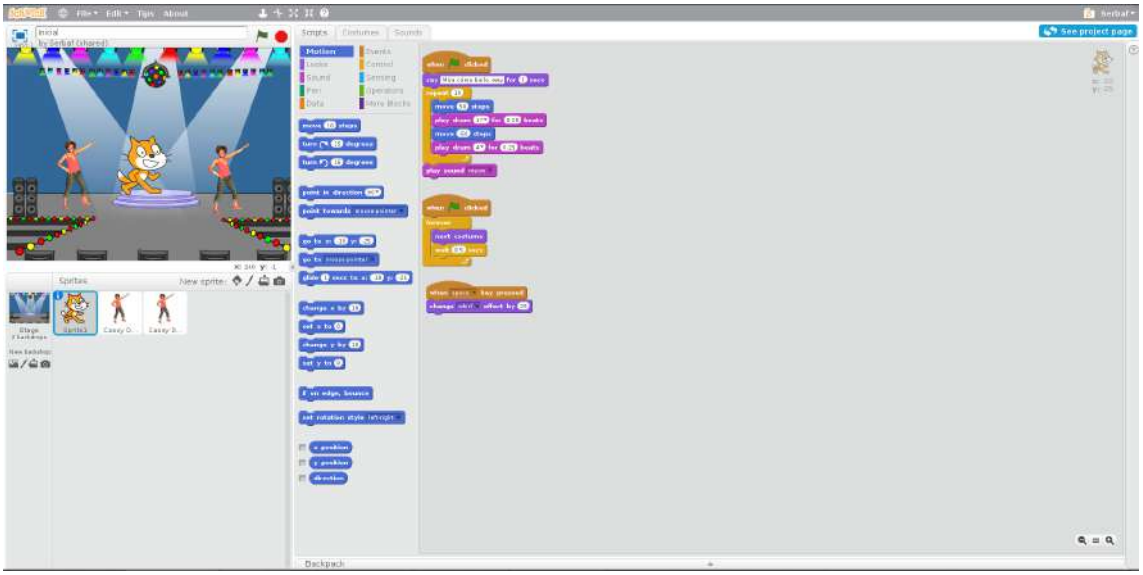

Figura 3.1: Entorno Scratch con una escena representada en la parte izquierda y con varias lineas de código en la parte derecha

generación de números primos y otros algoritmos abstractos están muy alejados de los intereses de los niños a los que se intenta introducir en el mundo de la programación.

Por supuesto, resultaría imposible usar Scratch para crear software de envergadura como podría ser un juego comercial moderno, una aplicación de oficina o no digamos ya un compilador o el *kernel* de un sistema operativo, ya que carece de características tales como herramientas para la gestión de memoria, funciones de entrada y salida de ficheros, gráficos avanzados, etcétera. No obstante, esto tampoco debe considerarse como una deficiencia, ya que todo ello jamás estuvo entre los objetivos del proyecto.

Para comprender mejor qué es Scratch hay que conocer cómo fue concebido. El proyecto tiene su origen en el año 2003, cuando el grupo Lifelong Kindergarten Group (LLK) del Massachusetts Institute of Technology (MIT[1]) desarrolló la primera versión. El LLK, liderado por Mitchell Resnick, desarrolló este primer Scratch utilizando una implementación de Smalltalk llamada Squeak. No obstante, no fue hasta 2007 cuando Scratch 1.0 fue lanzado para su uso por el gran público. La serie de versiones 1.X se extendió hasta la 1.4, que fue sustituida por Scratch 2.0 en 2013. Esta nueva versión pasó a estar escrita en Flash, aunque aún es compatible con cualquier proyecto desarrollado en Scratch 1.X. Además, con esta nueva versión se introdujo un entorno de programación en la propia web además del *offline* que era el único disponible hasta entonces. Esta es la versión que sigue vigente a día de hoy, aunque Scratch 3.0, desarrollado en JavaScript, está ya disponible en su versión alfa [2] y su lanzamiento está previsto para agosto de 2018.

Mitchell Resnick y su grupo LLK crearon Scratch con el objetivo de, según sus propias palabras, «nutrir una nueva generación de pensadores sistemáticos

[1] De hecho aún a día de hoy el sitio web oficial de Scratch se encuentra alojado dentro de la web del MIT: `http://scratch.mit.edu/`.

[2] Disponible en: `https://preview.scratch.mit.edu/`.

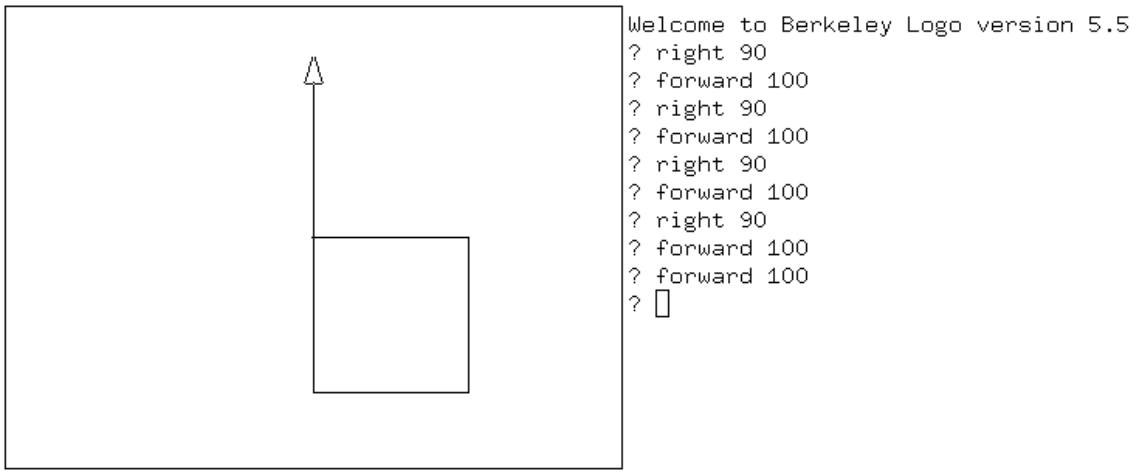

Figura 3.2: Ejemplo de código de UCBLogo para el dibujo de una silla

y creativos que usen la programación para expresar sus ideas». También pretendían hacer su contribución al conocimiento digital de las nuevas generaciones, a las que, aunque a menudo se les considera como expertas en nuevas tecnologías (los famosos nativos digitales), en realidad «pocos son capaces de crear sus propios juegos, animaciones o simulaciones. Es como si pudiesen leer pero no escribir» [12]. Por lo tanto, Scratch surgió con el objetivo específico de introducir en la programación a niñas, niños y adolescentes a través de un lenguaje y un entorno sencillos, atractivos e intuitivos, y con la intención más general de promover la creatividad y el pensamiento computacional.

Según reconocen los propios desarrolladores, la inspiración para el proyecto Scratch procede de distintas fuentes, tales como otros lenguajes de programación previos con orientación educativa, como el veterano Logo (Figura 3.2) o el más reciente Alice, así como del popular juego infantil Lego, del que se tomó la idea de la construcción de un programa como proceso de ensamblaje de pequeños bloques para formar un todo más grande. De hecho, el propio nombre de Scratch procede según sus creadores de «la técnica del *scratching* usada por los *disc jockeys* de *hip-hop*, que manipulan la música girando con sus manos los vinilos atrás y adelante, o mezclando diversas pistas de forma creativa. En la programación en Scratch la actividad es similar, mezclando gráficos, animaciones, fotos, música y sonido» [12].

De acuerdo con la filosofía de Scratch, todos los proyectos compartidos son de código abierto, están sujetos a una licencia Creative Commons Share Alike y son susceptibles de ser reinventados (*remixed*), es decir, tomados libremente por otros usuarios para modificarlos o ampliarlos.

En 2014, un grupo de la Tufts University en colaboración con Mitchell Resnick y el grupo LLK desarrolló ScratchJr como aplicación para dispositivos móviles. Aunque está basado en Scratch, se trata de proyectos distintos, paralelos y complementarios. ScratchJr está orientado a niños y niñas de cinco a siete años, a los que aún no se considera preparados para la complejidad de Scratch. En esta aplicación hay un menor número de bloques/instrucciones, éstos consisten únicamente en símbolos representativos de la lógica contenida en cada bloque en lugar de tener un texto que indique su función, y se elimina toda la parte matemática y lógica de Scratch, así como las subrutinas y variables. De este modo, el

Figura 3.3: Interfaz de la aplicación ScratchJr

resultado está muy acotado a la parte de representación visual de la escena y al movimiento de los personajes dentro de la misma, pero no es posible desarrollar una lógica interna de mayor profundidad. La Figura 3.3 permite hacerse una idea general del funcionamiento de ScratchJr.

Por último, una introducción a Scratch no podría acabar sin hacer referencia a la amplia comunidad que se ha sumado con entusiasmo compartiendo sus proyectos, participando en los foros [3], o usando plataformas como YouTube para mostrar sus creaciones o ayudar a iniciarse en Scratch a los nuevos usuarios. El sitio web posee además varias características propias de las redes sociales, como la posibilidad de marcar un proyecto como favorito, calificarlo con un «lo amo», dejar comentarios, etc. Sin olvidar acontecimientos como el Scratch Day, que suele celebrarse en mayo, y durante el cual se organizan múltiples eventos relacionados con Scratch en todo el globo y durante el cual *scratchers* de todo el mundo colaboran en la realización de proyectos [4]. A día de hoy la comunidad de Scratch cuenta con más de 30 millones de proyectos compartidos y 26 millones de usuarios, como se muestra en las estadísticas del sitio (Figura 3.4).

[3] Accesibles desde: https://scratch.mit.edu/discuss/.
[4] Más información en: https://day.scratch.mit.edu/.

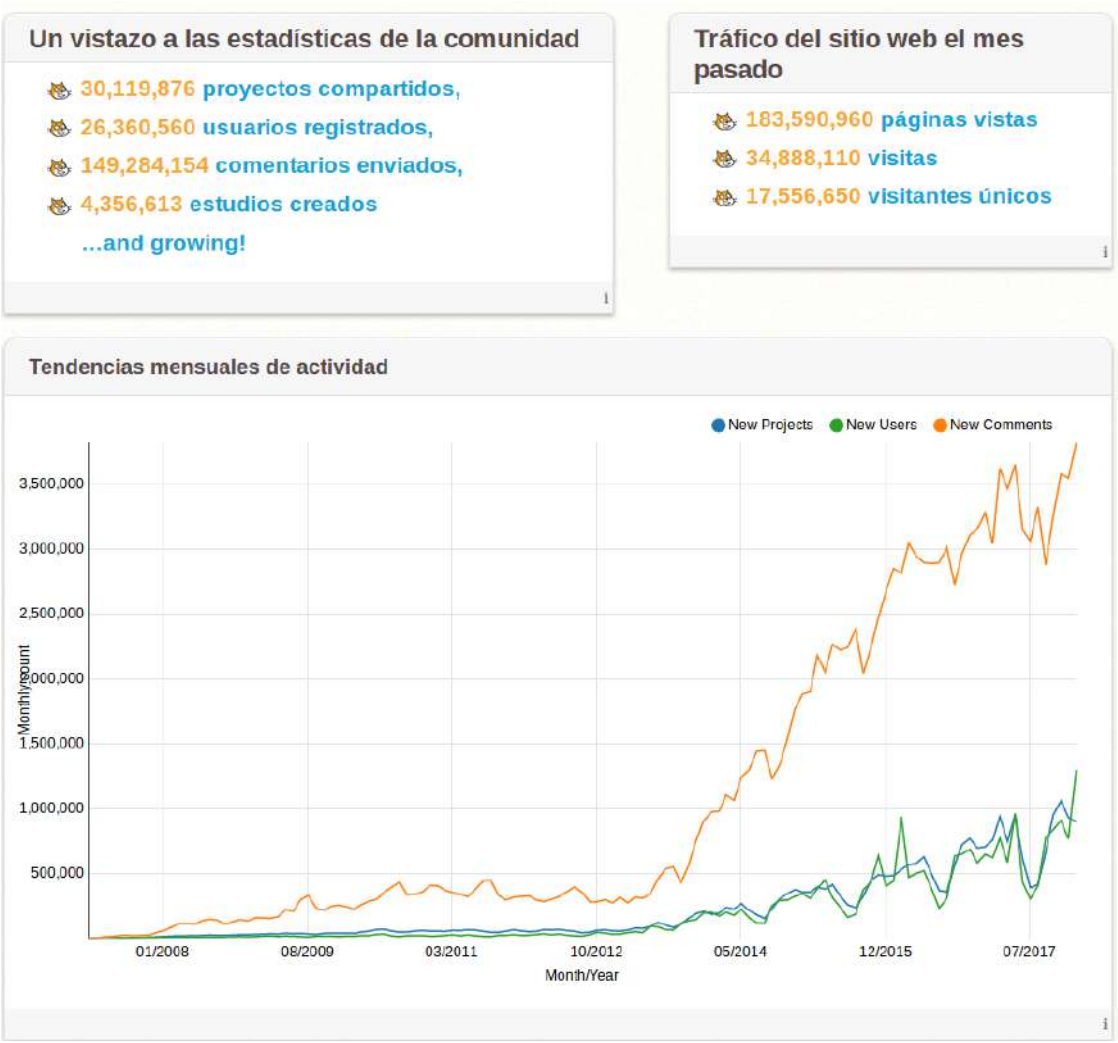

Figura 3.4: Estadísticas sobre Scratch

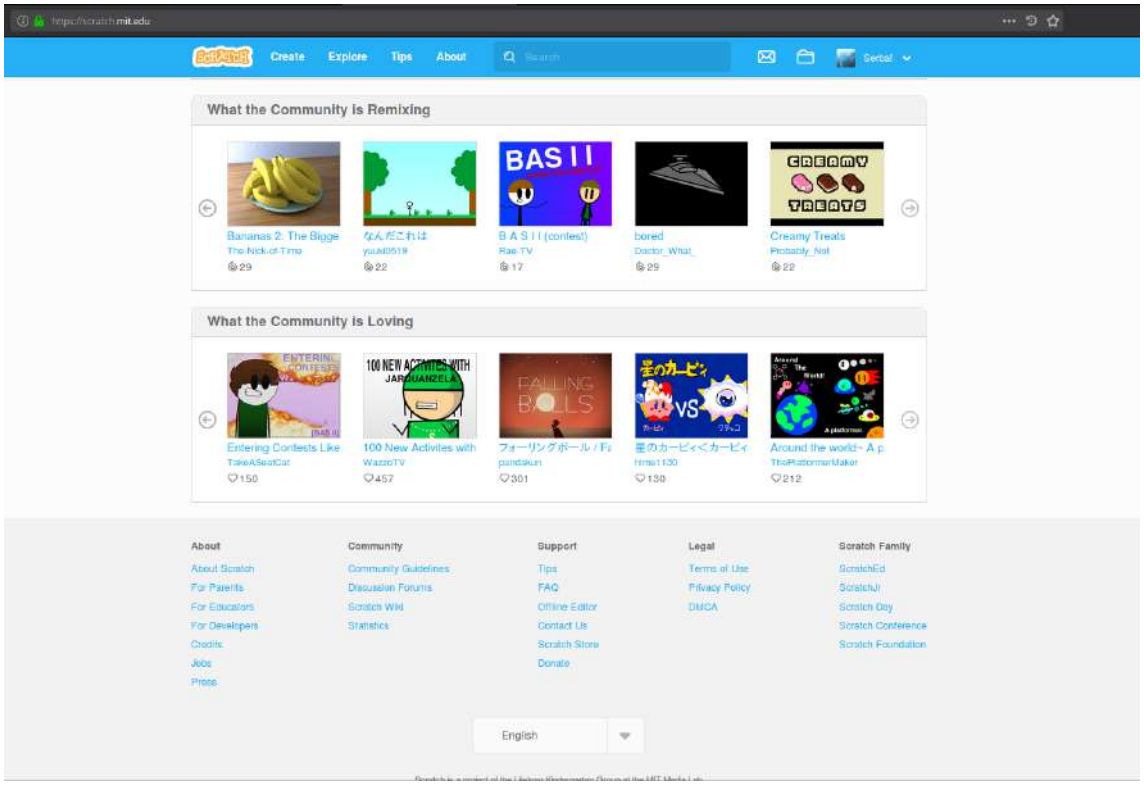

Figura 3.5: Página principal de la web de Scratch

3.2 Usar Scratch

Al acceder a Scratch, lo primero que se encuentra el usuario es una página principal con menús de navegación a distintas páginas interiores, un tablón con las últimas noticias, así como con varias secciones dedicadas a los proyectos y estudios (luego definiremos qué es cada uno) más destacados, o a los que han recibido un mayor número de «lo amo» o han sido más reeditados por terceros usuarios. Desde aquí cualquier persona interesada puede explorar la extensísima red de proyectos compartidos por la comunidad, ejecutarlos y estudiar su código. En el extremo inferior de la página también se pueden encontrar múltiples enlaces a recursos útiles, tales como la Wiki oficial de Scratch [17], una página con estadísticas más que completas sobre la comunidad *scratcher*, o los foros de discusión donde los usuarios plantean y resuelven sus dudas, entre otros.

Cabe señalar que a partir de este punto del trabajo se tratarán los detalles del uso del entorno de programación Scratch haciendo siempre referencia al editor integrado en la web oficial. Como se dijo anteriormente existe también un editor *offline* multiplataforma cuya interfaz y funcionalidad es prácticamente idéntica. También se usará a partir de ahora la terminología en castellano para los diferentes elementos del entorno, aunque se intentará dar para los más importantes también el término inglés en su primera aparición para facilitar su identificación, ya que para el desarrollo de los proyectos se ha usado la interfaz en inglés del entorno, como puede verse en las capturas de pantalla realizadas.

3.2.1. Proyectos y estudios

Una vez se ha accedido a la carpeta personal del usuario (para ello hace falta haberse registrado antes, evidentemente), llamada «Mis cosas» (*My Stuff*), se puede observar que existen varias carpetas y opciones. En la parte superior se muestran las acciones «Nuevo proyecto» (*New Project*) y «Nuevo estudio» (*New Studio*), y a la izquierda aparece una clasificación por directorios:

- **Directorio de proyectos:** dividido en compartidos y no compartidos. Aquí se almacenan todos los proyectos realizados por el usuario, siempre y cuando no hayan sido eliminados. El proyecto es la base de Scratch; podría decirse que son los programas, pero en un sentido amplio y no solo limitado al código. Incluyen el código, los objetos (*sprites*) y sus disfraces (*costumes*), los fondos (*backdrops*) que componen el escenario (*stage*), y también el ejecutable, es decir, la escena resultante con la que el usuario puede interactuar. Los proyectos pueden compartirse o dejar de compartirse en cualquier momento. Eso sí, desde el instante en el que se publican pasan a estar bajo licencia Creative Commons y cualquiera podrá ver el código fuente e incluso copiarlo para hacer una reinvención (*remix*). En Scratch no es posible publicar el ejecutable manteniendo oculto el interior.

- **Estudios:** un estudio no es más que una categoría a la que es posible asociar una serie de proyectos. Por ejemplo, en la realización de este trabajo se han desarrollado tres ábacos y se han añadido a un estudio llamado Abacus. Cualquiera puede ver un estudio, pero depende del propietario el abrirlo o no para que terceros usuarios puedan añadir sus propios proyectos.

- **Papelera:** aquí se conservan los proyectos eliminados para ofrecer una oportunidad de recuperarlos. También pueden eliminarse definitivamente si el propietario lo desea.

Desde esta perspectiva pueden observarse también las distintas estadísticas sociales de los proyectos compartidos: número de visitas de cada uno de ellos, comentarios, personas que los han marcado como favoritos o con un «lo amo» y el número de reinvenciones que se han hecho sobre ellos.

3.2.2. Escena y objetos

Al seleccionar un proyecto del propio usuario o de un tercero, se accede en primer lugar a una vista pública del mismo (Figura 3.6). En ella se puede «jugar» con el ejecutable resultante, además de contar con un apartado de instrucciones para explicar brevemente al público en qué consiste el programa y enseñarles cómo utilizarlo y otro de notas y agradecimientos. Además de ello se incluye aquí el resto de opciones dedicadas a lo social: sección de comentarios, botón de favorito, etc. Por último, en la parte superior derecha hay un botón llamado «Ver dentro» (*See Inside*) que permite acceder a los mecanismos interiores que conforman el proyecto. Esta es la parte que nos resulta más interesante para este trabajo y la que trataremos a partir de ahora.

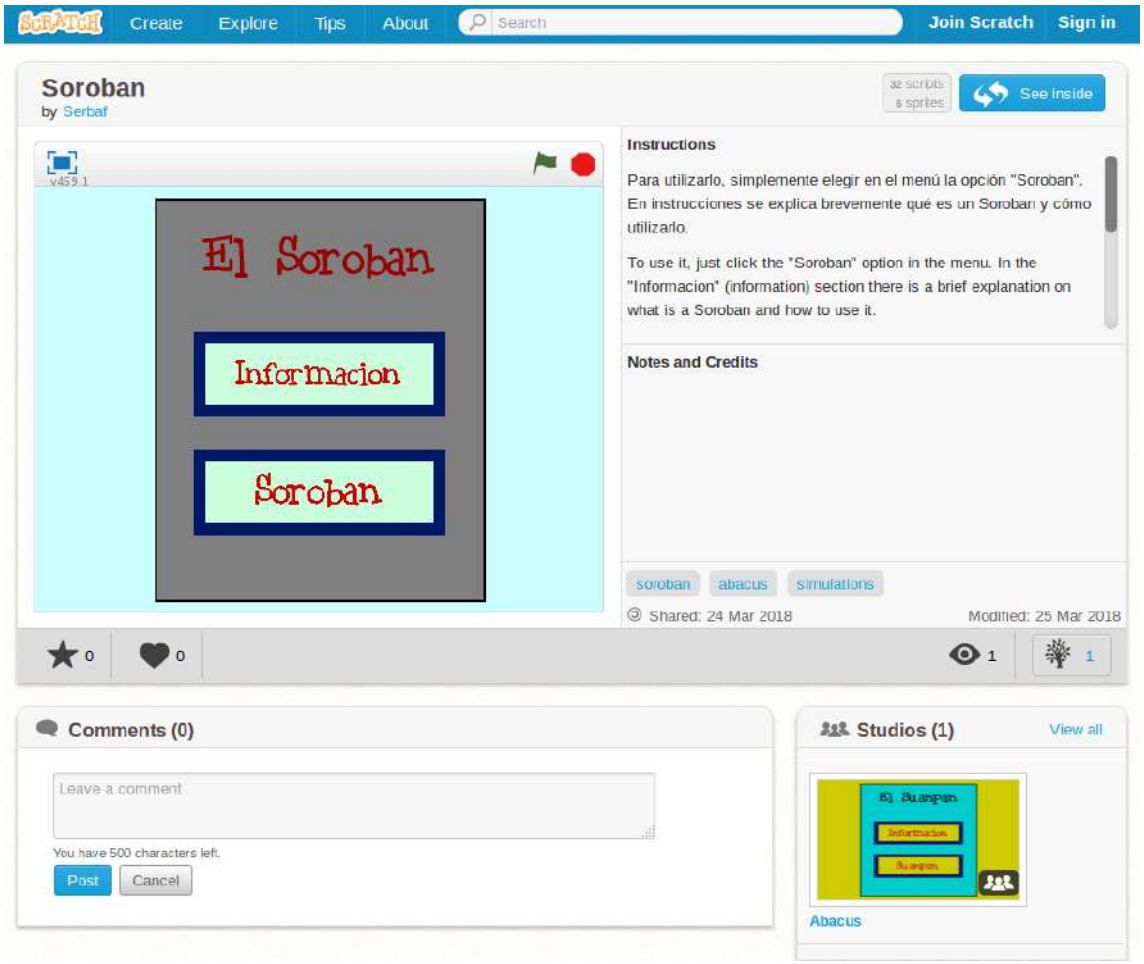

Figura 3.6: Visión de un proyecto Scratch desde la perspectiva de presentación

Al abrir la perspectiva interna de un proyecto, ya sea nuevo, en desarrollo, o completado, se accede al entorno de trabajo de Scratch. Éste está dividido en tres áreas principales: la escena, en la parte superior derecha; la zona de los objetos en la parte inferior izquierda, y por último el área de trabajo ocupando toda la mitad derecha de la ventana (y aún más, de hecho). Aparte de ello existen menús con utilidades diversas en la parte superior y un apartado oculto por defecto en el extremo derecho con ayuda, enlaces a tutoriales, etcétera.

Se comenzará por definir qué son los objetos y el escenario que aparecen en la parte inferior izquierda:

- **Escenario (*stage*):** formado por los «fondos de pantalla» usados en el proyecto durante la ejecución. El escenario se compone de uno o más fondos (*backdrops*) en principio inmóviles y menos interactivos que los objetos, aunque también es posible aplicar algunos efectos visuales o implementar sensores de captura de eventos. Esto se debe a que el escenario puede tener un código asociado que es común a todos los fondos. Los bloques de código utilizables aquí solo coinciden en parte con los de los objetos: una gran parte de bloques son comunes, otros muchos son solo propios de los objetos y unos pocos son exclusivos del escenario. Un proyecto Scratch tendrá siempre al menos un escenario (mientras que no es necesario que tenga objetos), aunque sea con un único fondo en blanco y sin código. Es posible asociarle efectos de sonido.

- **Objetos (*sprites*):** los objetos, por su parte, son las figuras que se mueven y realizan acciones de forma independiente sobre el escenario. Cada objeto tiene asociadas una o más imágenes llamadas disfraces (*costumes*), y que normalmente suelen ser variantes de una misma cosa, como un mismo rostro enfadado y alegre, por ejemplo. De igual modo que en el caso de los escenarios, también es posible asociar sonidos a un objeto. Por último, se puede asociar código de programación a cada objeto, e incluso asociarle variables locales, crear varios clones inicializados con distintos disfraces o valores en sus variables, etc. Esta característica de Scratch se acerca mucho a los conceptos propios del paradigma de la programación orientada a objetos.

Estos son todos los componentes de un proyecto Scratch, que estará compuesto siempre de un escenario con uno o más fondos y un código y unos sonidos asociados (opcionalmente), y sobre él cualquier número de objetos distintos y sus potenciales clones, también cada uno de ellos con unos programas y sonidos propios.

Con respecto a la parte audiovisual, Scratch permite una gran libertad. En el apartado de sonido, se proporciona una amplia librería de sonidos predefinidos a elegir, así como las opciones de cargar uno desde la propia máquina del usuario o de realizar una grabación desde el propio entorno web. En cuanto a las imágenes, ya sean para un escenario o para un objeto, se proporcionan unas alternativas similares: optar por imágenes ya dadas en la librería, cargarlas desde el ordenador local, realizar una foto en el mismo instante, o que el propio usuario cree la imagen en el momento con un sencillo editor gráfico proporcionado por Scratch.

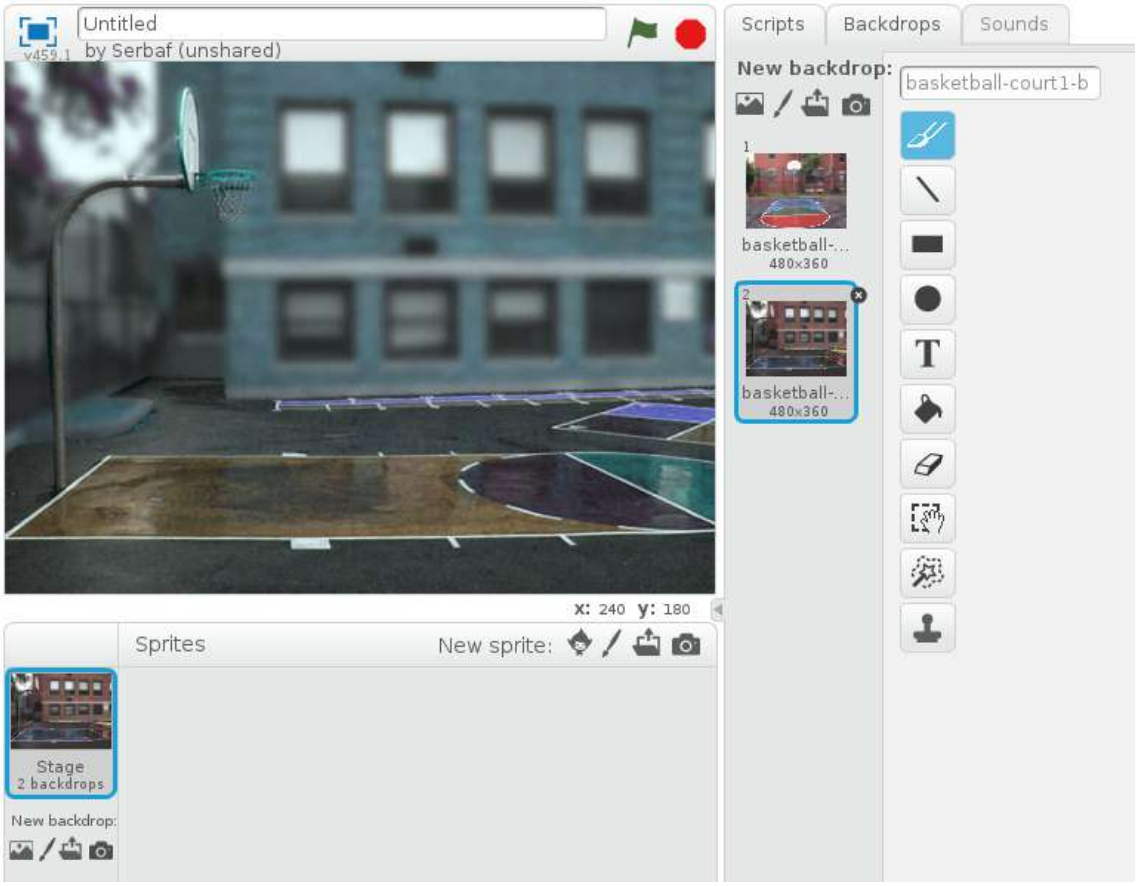

Figura 3.7: Escenario Scratch con dos fondos y al que se le ha aplicado un código que realiza una pequeña alteración en los colores

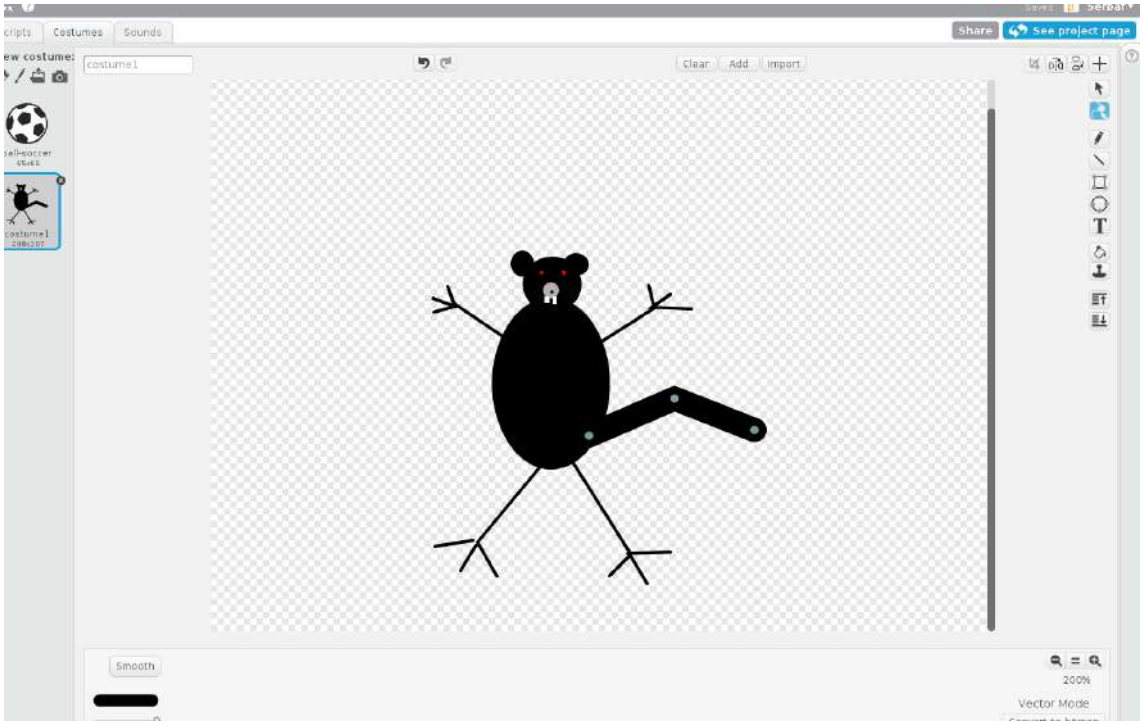

Figura 3.8: Editando uno de los disfraces para un objeto Scratch en modo vectorial

Éste permite el dibujo tanto en modo vectorial (Figura 3.8) como en mapa de bits (*bitmap*).

3.2.3. Montar el código

La parte de Scratch que seguramente resulte más relevante desde la perspectiva de un proyecto informático sea la de construcción del código. Como ya se ha adelantado, y esto es posiblemente la característica más distintiva del lenguaje, esto no se hace mediante la escritura, como es habitual en prácticamente todos los lenguajes de programación, sino a través de un proceso de «ensamblaje» de bloques, como si del juego de Lego (en el que los autores confirmaban haberse inspirado [12]) se tratase. A continuación se explican los distintos tipos de bloques que tenemos disponibles en Scratch.

Se pueden clasificar las piezas de instrucciones de Scratch de acuerdo con dos criterios. El primero de ellos sería según la forma del bloque (ver Figura 3.9):

- **Bloques de inicio (*Hat blocks*):** reciben un estímulo externo o interno ante el cual reaccionan y se comienza a ejecutar el bloque de código que encabezan. Por ello no se les puede encajar ninguna pieza en la parte superior, pero si en la inferior. Por ejemplo, pertenecen a esta categoría la instrucción `«al presionar (bandera verde)»`, que se empieza a ejecutar al arrancar el programa; o `«al presionar este objeto»`, que detecta cuando el usuario ha hecho clic en un objeto concreto y a partir de ahí inicia una serie de acciones de atención al evento. También pertenece a esta categoría la definición de funciones o subrutinas, también llamadas bloques (*blocks*) en Scratch, ya que inician un bloque de código independiente.

- **Bloques intermedios (*Stack blocks*):** conforman la mayoría de las instrucciones. Son aquellas que pueden tener código previo (de hecho necesitan tenerlo o jamás llegarían a ejecutarse) y pueden o no tener otros bloques a continuación. Si a continuación hay más instrucciones, el código seguirá su flujo secuencial. Si no, simplemente ese conjunto de instrucciones acabará hasta que vuelva a ser activado. Ejemplos de bloques intermedios serían la instrucción de mostrar un mensaje, `«decir x»`; o la llamada a una función o subrutina.

- **Bloques lógicos o booleanos (*Boolean blocks*):** tienen forma hexagonal y se caracterizan por devolver un valor booleano, es decir, un «sí» o un «no». Por ello se usan siempre dentro de otros bloques, como pueden ser las instrucciones de control o los operadores. Ejemplos de estos bloques serían los bloques `«¿ratón presionado?»` o los mismos operadores lógicos, como `«a o b»`, que necesita a su vez que los parámetros a y b sean bloques booleanos.

- **Bloques de valor (*Reporter blocks*):** tienen forma redondeada y contienen valores numéricos o *strings*. También se usan siempre dentro de otros bloques para realizar cálculos o como parámetro para realizar alguna acción determinada. Algunos ejemplos serían `«posición x del ratón»`, que devuelve las coordenadas en x del cursor en cada momento; o una variable cualquiera de las definidas por el usuario.

- **Bloques «C» (*'C' blocks*):** llamados así por su forma de C (aunque el `«si... si no»` es más bien una E mayúscula). Son todos ellos bloques de control del flujo del programa y tienen esta forma porque es posible anidar otros bloques en su interior para repetirlos en bucle o ejecutarlos condicionalmente. Ejemplos serían `«repetir»`, que no es más que el clásico bucle *for*, o el `«si... si no»`, que equivale a la estructura *if-else*.

- **Bloques de cierre (*Cap blocks*):** son aquellos que pueden tener código previo pero que siempre cierran una secuencia de código sin dar lugar a que pueda continuar, como su propia forma da a entender intuitivamente. Sólo existen dos de ellos y ambos son bloques de control: `«detener»`, que finaliza alguna ejecución, y `«borrar este clon»`, que elimina un clon (instancia) de un objeto.

La otra alternativa a la hora de realizar una clasificación de los bloques de Scratch sería según su funcionalidad y color asociado (ver Figura 3.10):

- **Bloques de movimiento (*Motion blocks*):** se les asocia el color azul oscuro. Son aquellos encargados de hacer que un objeto se desplace dentro de la escena. Contiene órdenes tanto de desplazamiento como de rotación, así como bloques de valor que contienen el valor de las coordenadas «x» e «y» y la dirección del objeto. Algunos ejemplos de bloques de esta categoría serían `«girar hacia la derecha x grados»` o el bloque de valor `«posición en x»`. Estas instrucciones son exclusivas de los objetos y no existen para los escenarios.

- **Bloques de apariencia (*Looks blocks*):** se les asocia el color lila. Están relacionados con las transformaciones en la apariencia de los objetos, como por

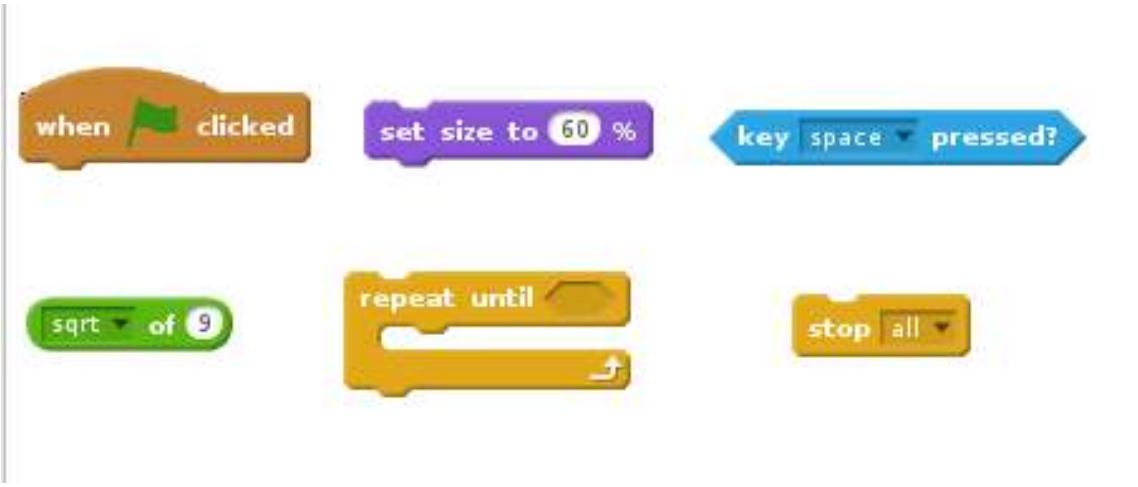

Figura 3.9: De izquierda a derecha y de arriba a abajo: bloque de inicio, intermedio, booleano, de valor, 'C' y de cierre

ejemplo cambios de color, escalado o cambio de plano. También se incluyen en esta categoría las instrucciones de escritura de mensajes por pantalla, que en el caso de Scratch aparecen con forma de bocadillos asociados a un objeto y tienen dos modalidades, «`decir`» y «`pensar`» (la representación de las cuales es similar a la de un cómic). También existen campos de valor conteniendo informaciones como el número de un disfraz, el nombre del fondo actual o el tamaño del objeto. También hay que hacer mención especial a las instrucciones «`mostrar`» (*show*) y «`esconder`» (*hide*), que hacen aparecer o desaparecer un objeto de la escena. Estas funciones son usadas en prácticamente todos los proyectos, pues los objetos son permanentes en un proyecto y no se pueden eliminar de una escena a otra (al menos los que no son clones), por lo que suele ser necesario esconderlos y mostrarlos explícitamente.

- **Bloques de sonido (*Sound blocks*):** se les asocia el color rosa. Hay diversas instrucciones para reproducir sonidos, ya sean predeterminados o personalizados. Asimismo, existen opciones para cambiar el volumen y el *tempo* de los sonidos, así como bloques que contienen el valor actual de estos dos parámetros.

- **Bloques de lápiz (*Pen blocks*):** se les asocia el color verde oscuro. Permiten que los objetos realicen dibujos a medida que se desplazan por el escenario, ya sea trazando líneas como un lápiz o estampando la propia figura del objeto. Algunas funciones de esta categoría son «`sellar`», para dejar la marca del objeto sobre el escenario, o «`fijar el color del lápiz a`».

- **Bloques de datos (*Data blocks*):** se les asocia el color naranja. En esta sección no hay ninguna instrucción al crear un proyecto, pero desde aquí el usuario puede crear dos elementos: variables y listas. Las variables pueden ser numéricas o cadenas de caracteres (*strings*), pero no hace falta definir su tipo, como ya se señaló. Una vez creada una variable se dispondrá de algunas instrucciones asociadas a la misma, «`fijar variable a`», para otorgarle un valor cualquiera, y «`cambiar variable por`», que aumenta o disminuye el contenido de la misma (x puede ser positiva o negativa). También

se dispone de funciones «`mostrar`» y «`esconder`» según se quiera o no ver el valor de la variable en el escenario. En el momento de crear una variable el usuario deberá decidir si le concede un rango global o local para el objeto en el que esté definida. También se proporciona una utilidad de creación de listas básicas. Las listas también pueden ser globales o locales y poseen las clásicas funcionalidades propias de este tipo de estructuras: añadir elemento, borrarlo, reemplazarlo, insertar un valor en una determinada posición, obtener la longitud de la lista, etc.

- **Bloques de eventos (*Event blocks*):** se les asocia el color marrón. Aquí se encuentran todos los bloques de inicio, pues permiten empezar a ejecutar un fragmento de código tras recibir un evento, ya sea el del mismo inicio de la ejecución (bandera verde), u otra señal: al hacer clic sobre el objeto, al presionar una tecla, al recibir el mensaje de una difusión (*broadcast*)... Aparte de estos bloques de inicio también hay otros dos encargados del mencionado envío de mensajes, uno de ellos con espera a la respuesta y el otro sin ella.

- **Bloques de control (*Control blocks*):** se les asocia el color amarillo. Son las instrucciones de control de flujo propias de la mayoría de lenguajes de programación. En Scratch se cuenta con el bucle «`repetir`» (equivalente al *for*), con el «`si`» y el «`si... si no`» (*if* e *if-else*), con «`repetir hasta`» (similar a un *while* con el valor de la condición invertida) y con el «`por siempre`» (equivalente a un *while* con la guarda permanentemente verdadera (*True*)). Además hay dos instrucciones de espera, una por tiempo y otra por condición booleana, y tres instrucciones para la gestión de clones, así como una para detener la ejecución de algún elemento.

- **Bloques sensores (*Sensing blocks*):** se les asocia el color azul claro. Permiten realizar acciones muy diversas, en general centradas en obtener algún dato externo, como la posición del ratón, el nombre del usuario, si se ha presionado una determinada tecla, etc. También permiten una cierta interacción con el usuario, como realizarle una pregunta y guardar el resultado obtenido en un bloque contenedor de valor llamado «`respuesta`». Aparte de estos, hay varias instrucciones relacionadas con el color, el vídeo y el sonido.

- **Bloques operadores (*Operator blocks*):** se les asocia el color verde claro. Contienen operaciones matemáticas (suma, resta, multiplicación, división, módulo y redondeo, así como una instrucción configurable con operaciones más complejas), lógicas (*and*, *or*, *not*), comparadores («`mayor que`», «`menor que`» e «`igual`»), operaciones sobre cadenas de caracteres y un generador de números aleatorios.

- **Más bloques (*More blocks*):** se les asocia el color púrpura. Aquí, al igual que en los bloques de datos, no existe ninguna función predefinida, pero permite crear los propios «`bloques`» que el usuario desee. O, lo que es lo mismo, definir subrutinas para después llamarlas cuando resulten necesarias. Además existe una opción para realizar extensiones, es decir, añadir una de las tres librerías especializadas que se ofrecen. Éstas permiten el manejo de hardware didáctico, como por ejemplo la PicoBoard, una placa con sensores de luz y sonido para implementar programas que puedan actuar de forma interactiva con el «mundo real».

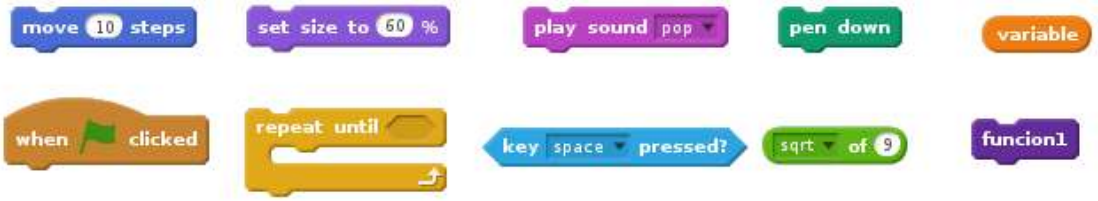

Figura 3.10: De izquierda a derecha y de arriba a abajo: bloque de movimiento, apariencia, sonido, dibujo, datos, eventos, control, sensores, operadores y 'más bloques' (funciones)

3.2.4. Scratch como lenguaje de programación

Aunque el entorno Scratch tenga una clara orientación visual para incrementar su atractivo, lo interesante del mismo no es el editor de imágenes ni la grabación de sonidos, sino el original lenguaje que permite crear infinidad de proyectos de (casi) toda índole.

En primer lugar, cabe hacer hincapié en algunos aspectos del lenguaje que ya se mencionaron anteriormente y en otros que aún no. Para empezar, en Scratch no hay que escribir ningún código, como máximo algún valor numérico o alguna cadena de texto para pasar como parámetros a una instrucción. Todo el código real que subyace está enmascarado tras una capa adicional de abstracción a las que ya tiene cualquier lenguaje de alto nivel, y la instrucción pasa a ser representada por un bloque con un color, indicativo del tipo de instrucción; una forma que señala dónde puede encajar dicha instrucción, y un texto definiendo la acción que realiza. De este modo se impide por completo el problema de los errores léxicos (pues el usuario no se encarga de escribir el código), sintácticos y semánticos (pues al montar el código el usuario no podrá cometerlos, ya que unos tipos no compatibles o una estructura incorrecta simplemente «no encajará»).

También, desde un punto de vista de la ingeniería informática resulta conveniente hacer algunos apuntes sobre Scratch. Como ya se ha dicho antes, se trata de un lenguaje que toma gran parte de la filosofía de la programación orientada a objetos, ya que cada objeto que se crea puede tener un código asociado, con sus propias funciones y variables, puede crear clones (instancias) con distintos valores y disfraces, y el resto de objetos o el usuario pueden interaccionar con él. Por otro lado, también es un lenguaje de programación dirigido por eventos, pues está totalmente orientado a la interacción del usuario con la escena, y posee gran cantidad de funciones de captura de eventos para detectar clics sobre un objeto, captar la posición del cursor, etc. Además, es débil y dinámicamente tipado, pues no se realiza ninguna declaración de tipos por parte del usuario y estos se evalúan en tiempo de ejecución. Por último, hay que decir que es un lenguaje interpretado, cuyo intérprete se encuentra en el propio entorno Scratch y carece de compilador. Todo ello lleva a que se trate de un lenguaje de muy alto nivel, de gran abstracción y, con toda probabilidad por ello altamente ineficiente en términos de uso de recursos, pero dados los modestos objetivos del lenguaje esto es poco relevante.

CAPÍTULO 4
Diseño e implementación de ábacos mediante Scratch

En el presente capítulo se muestran los detalles de la implementación de tres programas en Scratch, cada uno de ellos centrado en el caso de un ábaco diferente, concretamente en los casos del *soroban*, del *suanpan* y del *schoty*, cuyo funcionamiento «real», el cual se desea emular, ya se describió con anterioridad. Las tres aplicaciones se han desarrollado con el objetivo de ser integradas en una página web dentro del sitio del Museo de Informática para ampliar sus contenidos y, sobre todo, despertar en niñas y niños un verdadero interés por la historia de las matemáticas, el cálculo y la informática. Por ello, los tres proyectos tienen una orientación claramente visual, lúdica y de interactividad con el usuario. Y, como se verá a continuación, Scratch proporciona una serie de herramientas que lo convierten en un lenguaje ideal para el desarrollo de aplicaciones con estas características.

4.1 Diseño visual

Como ya se ha adelantado en la introducción, la parte gráfica de las aplicaciones desarrolladas es fundamental, ya que están planteadas con un objetivo didáctico a la vez que lúdico, y están en especial orientadas a un público infantil y juvenil. Por lo tanto, el objetivo inicial era crear una interfaz gráfica de usuario que resultase fácil de usar, intuitiva y, a la vez, que fuese atractiva visualmente y resultase divertida e informal.

Para su diseño se ha utilizado casi exclusivamente el sencillo editor gráfico proporcionado por el entorno de desarrollo de Scratch, pero que resulta suficiente para la creación de fondos de color plano, rótulos de texto y botones sencillos. Las pocas veces que ha resultado necesario realizar algún retoque de imagen algo más complejo, como añadir transparencias o eliminar fondos, se ha utilizado el editor GNU Image Manipulation Program (GIMP). Puesto que Scratch permite importar imágenes y sonidos desde un archivo local, resulta sencillo también manipular los elementos gráficos con herramientas externas al entorno para después simplemente importarlos e integrarlos en un fondo o un objeto, como se ha hecho en este caso con las imágenes de los tres ábacos reales, incluidas en los menús de las aplicaciones, como se puede observar en la Figura 4.1. Por último, también se

Figura 4.1: Uno de los menús principales, en este caso del proyecto sobre el ábaco chino. Los dos objetos que no forman parte del fondo y que poseen interactividad son los botones «Información» y «Suanpan»

ha hecho uso de la librería de imágenes proporcionada directamente por Scratch, en concreto para el caso del objeto **Ghoul**.

El diseño de la interfaz gráfica es muy similar en las tres aplicaciones, aunque existen algunas pequeñas diferencias en el coloreado y en la distribución de los elementos en los menús, esto último debido principalmente a las necesidades impuestas por las diferentes formas de los tres ábacos. Aparte de esos matices, la interfaz gráfica de usuario tiene la siguiente estructura en los tres casos:

- **Créditos iniciales:** primera escena del proyecto, no interactiva. Se compone del nombre del ábaco en cuestión y de cuatro objetos informativos de la autoría. Estos son un texto con los nombres del tutor y del autor, y tres logos: el de la UPV, el del Museo de informática y el de la Escuela Técnica Superior de Ingeniería Informática (ETSINF) (Figura 4.2).
- **Menús**: tras los créditos, el usuario accede a un sistema de menús que permite acceder a las distintas funcionalidades del programa. Estos menús se componen gráficamente de un fondo sencillo con una imagen del ábaco concreto del que trata la aplicación insertada en él. Sobre este fondo se superponen los botones que permiten el acceso a las otras distintas escenas del proyecto (Figura 4.1).
- **Pantallas de información:** estas escenas se componen simplemente de fondos con texto, el cual forma parte del propio fondo, aunque también se po-

Figura 4.2: Escena inicial con los créditos de autoría (izquierda) y uno de los fondos con información sobre la historia y el uso del *schoty* (derecha)

dría haber optado por convertirlo en objetos independientes, como se hizo en el caso de los créditos iniciales. Aparte del fondo, estas escenas solo cuentan con un par de botones de navegación en la parte inferior de la pantalla, que permiten al usuario avanzar a través de las distintas transparencias o volver al menú (Figura 4.2).

- **Ábacos:** en el caso de los ábacos virtuales, la presentación gráfica intenta imitar la estética más tradicional de sus equivalentes analógicos. Así pues, para el *soroban* las cuentas se dibujaron con una forma hexagonal aplastada de color madera, mientras que en el proyecto del *suanpan* son más redondeadas y de color negro, y para el caso del ábaco ruso el objeto **Cuenta** posee dos disfraces que comparten tamaño y forma, pero difieren en el color, pues uno es negro, para los clones de la parte central de cada varilla, y el otro blanco para el resto de las piezas. Aparte del propio ábaco, en la escena aparecen algunos objetos más: un botón para volver al menú, otro para reiniciar la posición de las cuentas y poner el ábaco a cero y, en algunas modalidades del programa, un personaje de la librería gráfica de Scratch llamado **Ghoul**. Además, normalmente hay una o varias variables de Scratch visibles en pantalla mientras se interactúa con los ábacos, ya sea para mostrar el número que se está representando o para otra finalidad como ejercer de cronómetro o representar una operación aritmética con una cadena de texto. Por último, para la aplicación del ábaco ruso también se creó un objeto **Reloj** de arena que representa el paso de un periodo de tiempo mediante disfraces, creando la sensación de que la arena va cayendo a medida que pasan los segundos. Puede verse el *soroban* en pleno funcionamiento en la Figura 4.3

4.2 Funcionalidades generales

A continuación se pasa a describir la programación realizada en el lenguaje Scratch para el desarrollo de las tres aplicaciones. En el presente capítulo se tratará de aquellos bloques de código que, por implementar comportamientos que son comunes a los tres programas, son prácticamente iguales en todos ellos, más allá

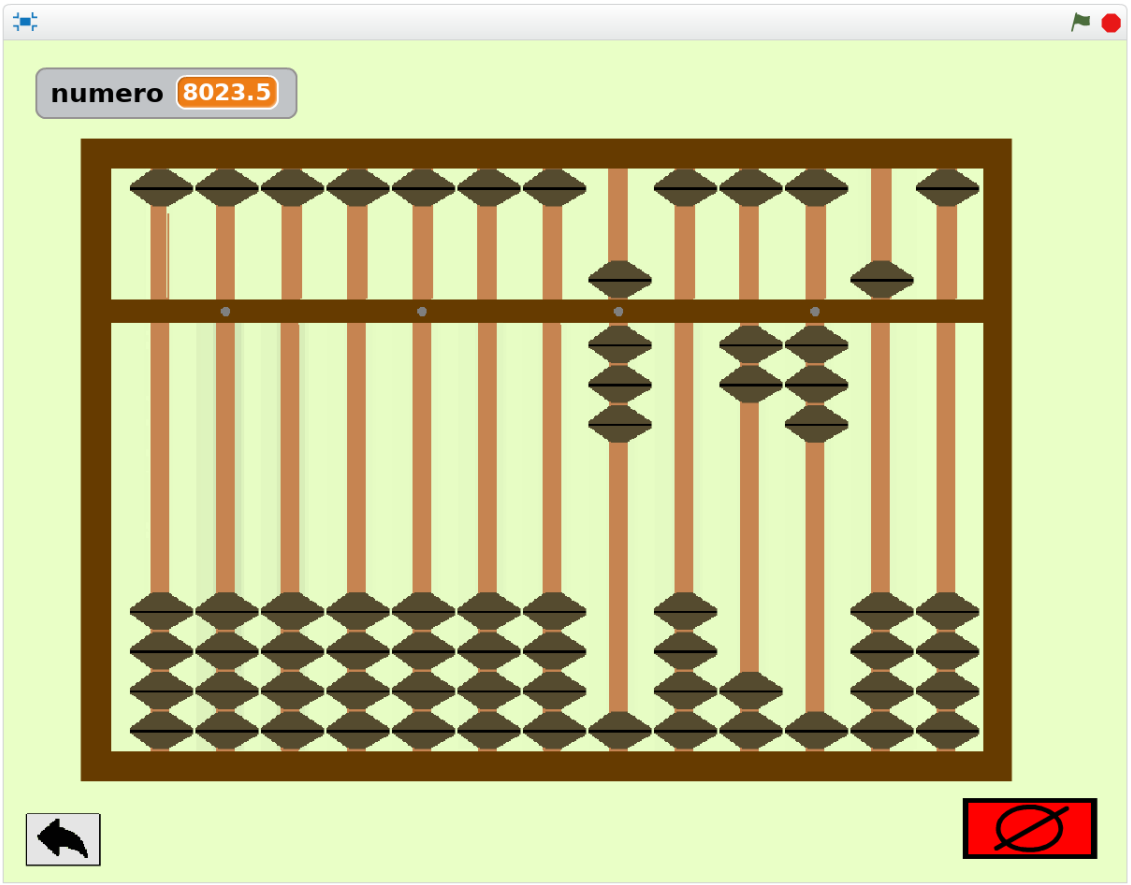

Figura 4.3: Uno de los tres ábacos virtuales implementados. En este caso se trata de un *soroban* con el que se está interactuando en «modo libre». El número representado es el 8 023, 5

de matices o variaciones triviales en las coordenadas, colores, etc. Aquellas funcionalidades o partes de la ejecución de los programas más diferenciadas entre sí se cubrirán en el siguiente apartado.

4.2.1. Navegación

La navegación dentro de los propios proyectos, entendida como el flujo que controla el paso por las distintas pantallas o fases del programa, es prácticamente idéntica en los tres casos. La forma de implementar este aspecto de la lógica se ha basado fundamentalmente en la sincronización de los objetos y del escenario mediante la difusión de mensajes, que a su vez es activada por determinados eventos.

Por ejemplo, los tres proyectos comienzan con una pantalla de créditos estática que tras un lapso de cinco segundos deja paso al menú principal con el que el usuario ya puede interactuar de forma activa. Este comportamiento se codifica con un bloque que se inicia al presionar la bandera verde (es decir, el inicio de programa), seguido de la difusión de un mensaje que indica el paso a la escena de créditos inicial, ésta a su vez sucedida por una espera incondicional de cinco segundos y, a continuación, otra difusión, en este caso de una orden que indica la transición al menú principal (se puede ver este bloque de arranque de la aplicación en la Figura 4.4). Por supuesto, este diminuto bloque de código no hace nada perceptible para el usuario por sí mismo, pero puede considerarse que emite una serie de órdenes en forma de mensajes dirigidos a todo el resto de objetos y al escenario de la aplicación. Si en los otros elementos del proyecto se define un bloque (o más) que atienda a alguno de esos eventos, se puede lograr la sincronización de los distintos elementos del programa y, de este modo, implementar la lógica de cambio de escenas.

Siguiendo con el ejemplo anterior, al pasar los cinco segundos y enviarse el mensaje de realizar el cambio de escena al menú, todos los elementos (objetos y escenario) del programa que tengan un bloque de código encabezado por la instrucción «`al recibir menú`» ejecutarán las instrucciones de dicho bloque. Puesto que lo que se está realizando en este caso concreto es un cambio de pantalla, el escenario manejará este evento cambiando el fondo, mientras que los objetos de créditos y logos se esconderán y un par de botones aparecerán en pantalla en las coordenadas indicadas. Algunos de estos bloques *listeners* que se activan con el menú han sido recogidos en la Figura 4.4 para dar una mejor idea global de cómo se puede realizar este tipo de sincronización orientada a eventos con Scratch.

El resto de la navegación entre las diferentes pantallas de las tres aplicaciones se rige por principios similares. Cada uno de los botones del menú activa un evento que lleva a una u otra funcionalidad del programa, ya sea acceder a un segundo submenú, al modo de información sobre el ábaco en cuestión o a una de las tres modalidades de interacción con el ábaco virtual: tutorial, uso libre y el modo de juego o «desafío». Por último, existen otros dos botones relacionados con el cambio de escena: la flecha de retroceso, presente en casi todas las pantallas de la aplicación y que genera un evento de regreso al menú principal en todos los casos, y permite de este modo volver al punto de partida (excluyendo los créditos iniciales) y seguir utilizando la aplicación indefinidamente hasta que

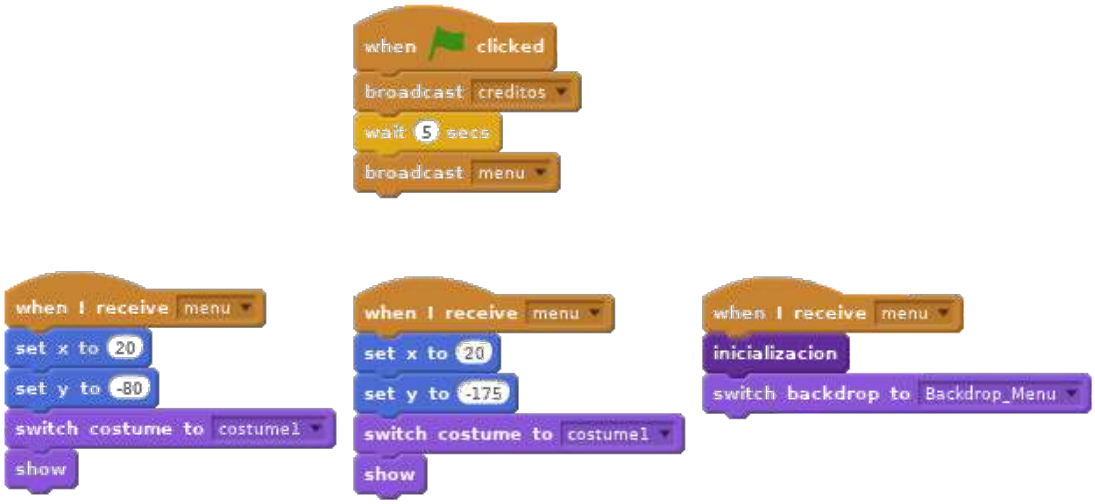

Figura 4.4: Arriba, código de arranque de programa, con dos difusiones de mensajes para indicar el cambio de escena. Abajo, el código de atención a este evento por parte de los dos botones del menú y por el propio escenario (abajo a la derecha)

el usuario lo desee. Cabe señalar aquí que este evento de vuelta al menú también provoca el reinicio de la mayor parte de las variables del programa con muy contadas excepciones, lo cual evita posibles incongruencias que pudiesen provocar comportamientos indeseados. Por último, existe una flecha de avance que solo se utiliza en las pantallas de instrucciones de texto, y que permite avanzar de una a la siguiente en orden secuencial.

En resumen, la difusión de mensajes, activada por eventos y en el caso concreto de estas tres aplicaciones principalmente a través de los eventos de tipo `«clic sobre el botón»`, teje toda una red de sincronizaciones entre los distintos elementos del proyecto que permiten que una serie de bloques de código aparentemente independientes actúen en realidad de forma coordinada. Las respuestas instantáneas a estos mensajes, sobre todo (pero no solo) mediante la ocultación y aparición de objetos y el cambio de fondos, son las que permiten que el programa avance de forma fluida y el usuario obtenga una sensación de interactividad y de «progreso» por las distintas escenas.

4.2.2. Animaciones e interacción

En este apartado se tratará de la construcción de animaciones y de la interacción del programa con el usuario. Estos dos puntos aparecen en distintas partes de los proyectos, y por ejemplo el usuario puede interactuar con los menús, como ya se ha visto, o hacer clic sobre una cuenta de un ábaco para que esta se desplace. No obstante, en este punto se pondrá el foco sobre el objeto **Ghoul**, ya que es el que tiene con diferencia bloques de código más extensos, pero la mayoría de los cuales consisten en animaciones y atención a clics por parte del usuario, por lo que se intentará explicar estos conceptos con algún ejemplo y se omitirá un análisis del resto del código, pues sigue los mismos patrones.

Ugluk, nombre con el que se presenta el objeto **Ghoul** ante el usuario, es una especie de fantasma o espectro que cuenta con dos disfraces predefinidos y uno

más añadido por el desarrollador. Debido a su aspecto entre terrorífico y simpático se eligió para hacer de «guía» y encargarse de la interacción con el usuario en distintas partes de los proyectos, y más concretamente en las modalidades de **Tutorial** y de **Desafío** implementadas en cada proyecto. El comportamiento del **Ghoul** se ha diseñado intentando crear un personaje a mitad camino entre lo espectral y lo jocoso, de tal modo que haga más entretenido el uso de las aplicaciones, en especial entre los niños y niñas que son los potenciales usuarios de las mismas.

Aunque la funcionalidad de los tres modos **Desafío** difiere sustancialmente en su lógica interna y en su funcionamiento de cara al usuario, las líneas de código que se encargan de la animación del personaje son en realidad mayoritarias, y resultan similares en todos los proyectos. Además, los tres **Tutoriales** sobre el uso de los ábacos se basan casi por completo en animaciones, comentarios y difusiones de mensajes por parte del **Ghoul**, por lo que pueden considerarse equivalentes entre sí, obviando los mínimos matices que hay en cada caso.

¿Cómo se logra la sensación de animación de un objeto en Scratch y, en este caso concreto, del **Ghoul**? Al ser Scratch un lenguaje muy orientado al aspecto gráfico y visual, proporciona una buena cantidad de instrucciones englobadas en la categoría **Movimiento** que permiten realizar de forma sencilla animaciones que en otros lenguajes necesitarían de un bloque de instrucciones bastante complejo o del uso de librerías externas. En los proyectos el movimiento, es decir, el desplazamiento de un objeto espaciado a lo largo de un periodo de tiempo, se ha implementado con el uso en pequeños bloques de un bucle `«repetir x»` con las instrucciones `«cambiar x por»` y `«cambiar y por»` en su interior, con lo que se puede controlar la duración de una animación de desplazamiento (aumentando o reduciendo el número de iteraciones). También se puede regular la distancia desplazada, que equivaldrá al número de iteraciones multiplicado por el cambio de cada una de las coordenadas en cada vuelta del bucle. Por supuesto, esto significa que también la velocidad de la animación puede variarse (pues equivale a desplazamiento partido por tiempo), así como la fluidez de la animación, ya que bucles cortos con cambios grandes en las coordenadas en cada iteración darán la sensación de un movimiento «a saltos». Cabe señalar que en realidad Scratch permite realizar este tipo de animaciones con una única instrucción llamada `«deslizar en n segs a x, y»`, pero se ha preferido optar por la solución explicada para poder realizar desplazamientos relativos y no con coordenadas absolutas.

Además de la traslación, para lograr una sensación de animación también se hace uso de sonidos, concretamente una risa espectral, de cambios de disfraz y de instrucciones de rotación del objeto para dotar de una mayor vitalidad y naturalidad al personaje. También resulta interesante ver cómo se ha implementado la escritura de mensajes dirigidos al usuario. El comportamiento deseado era que el mensaje no desapareciese hasta recibir un clic de ratón o teclado, para que cada usuario pudiese adaptar el ritmo del programa a su velocidad de lectura. Esto se logró con un bloque de tres instrucciones que se repite con mucha asiduidad en el código del **Ghoul**: `«decir»` el mensaje, `«esperar 0,3 segundos»` y finalmente `«esperar a que se presiona el ratón o el teclado»`. El funcionamiento del código es bastante evidente salvo por la espera de tres décimas de segundo que se introdujo al comprobar que sin esa instrucción el código en ocasiones realizaba

comportamientos no deseados, como saltar varios mensajes con un solo clic, sin que diese tiempo a leerlos.

Por último, cabe señalar que el objeto **Ghoul** también hace difusiones de mensajes como las antes mencionadas para lograr que otros objetos del entorno se sincronicen con sus necesidades. Por ejemplo, si el fantasma asegura que va a representar cuatro centenas en un *soroban*, necesita a continuación dar valores a las variables columna y fila para seleccionar las cuentas del ábaco que quiere desplazar (en el siguiente apartado se explicará con detalle este y otros aspectos del funcionamiento de los ábacos) y, por último, difundir un mensaje de movimiento de piezas, que será atendido por estas, y provocará en consecuencia su movimiento y la suma de 400 unidades, así como la sensación de sincronía entre el fantasma y el propio ábaco. En la Figura 4.5, que muestra un bloque de código extraído de uno de los tutoriales, se incluyen casi todos los aspectos mencionados: el mecanismo de envío de mensajes, el movimiento y la interacción con el propio ábaco. El código mostrado está incluido en un bloque o subrutina por mera cuestión de legibilidad y limpieza, ya que todo el código de **Tutorial** es secuencial y podría haberse construido como un solo (y enorme) bloque.

Aunque en el próximo apartado se volverá al **Ghoul** para describir la lógica de las tres modalidades de **Desafío**, con los elementos hasta ahora descritos ya es posible comprender gran parte del código de este objeto, y la totalidad del funcionamiento del modo **Tutorial**, que se compone básicamente de animaciones, mensajes por pantalla y difusiones, todo ello realizado de forma secuencial y no interactiva, ya que de hecho se deseaba que los tutoriales se comportasen de modo similar al de un vídeo de animación.

4.2.3. Otros elementos comunes

Hay algunas funcionalidades que resultan bastante menores comparadas con las dos anteriores, pero que también se repiten en los tres proyectos. Una de ellas es la función de reinicio, que se activa mediante el botón rojo que aparece en todas las escenas con ábaco. Al apretar dicho botón se produce la difusión de un mensaje que es captado por las cuentas del ábaco. Éstas comprueban si han sido desplazadas de su posición de origen y, en caso afirmativo, se mueven de regreso a la situación de partida. Puesto que todas las cuentas reciben el mensaje a la vez, todas aquellas afectadas se desplazarán de forma simultánea. Véase la Figura 4.6 con el código que implementa esta funcionalidad.

Otro elemento común a la lógica de los tres proyectos es el «bloqueo» de las cuentas, es decir, el código que impide que el usuario interactúe con el ábaco en cuestión cuando esto pudiera resultar inconveniente para el correcto funcionamiento del programa. Esta característica se añadió a los proyectos tras comprobar que un usuario que operase de forma muy rápida con las cuentas podía hacer aflorar un *bug*: al tratar de desplazar una pieza del ábaco cuando la animación anterior aún no había concluido sucedía que el desplazamiento previo se detenía a mitad de su camino (en una posición intermedia, ni «subida» ni «bajada»), algo totalmente ajeno a la naturaleza de cualquiera de los ábacos. Además, esos clones de **Cuenta** afectados dejaban de responder por completo a partir de ese momen-

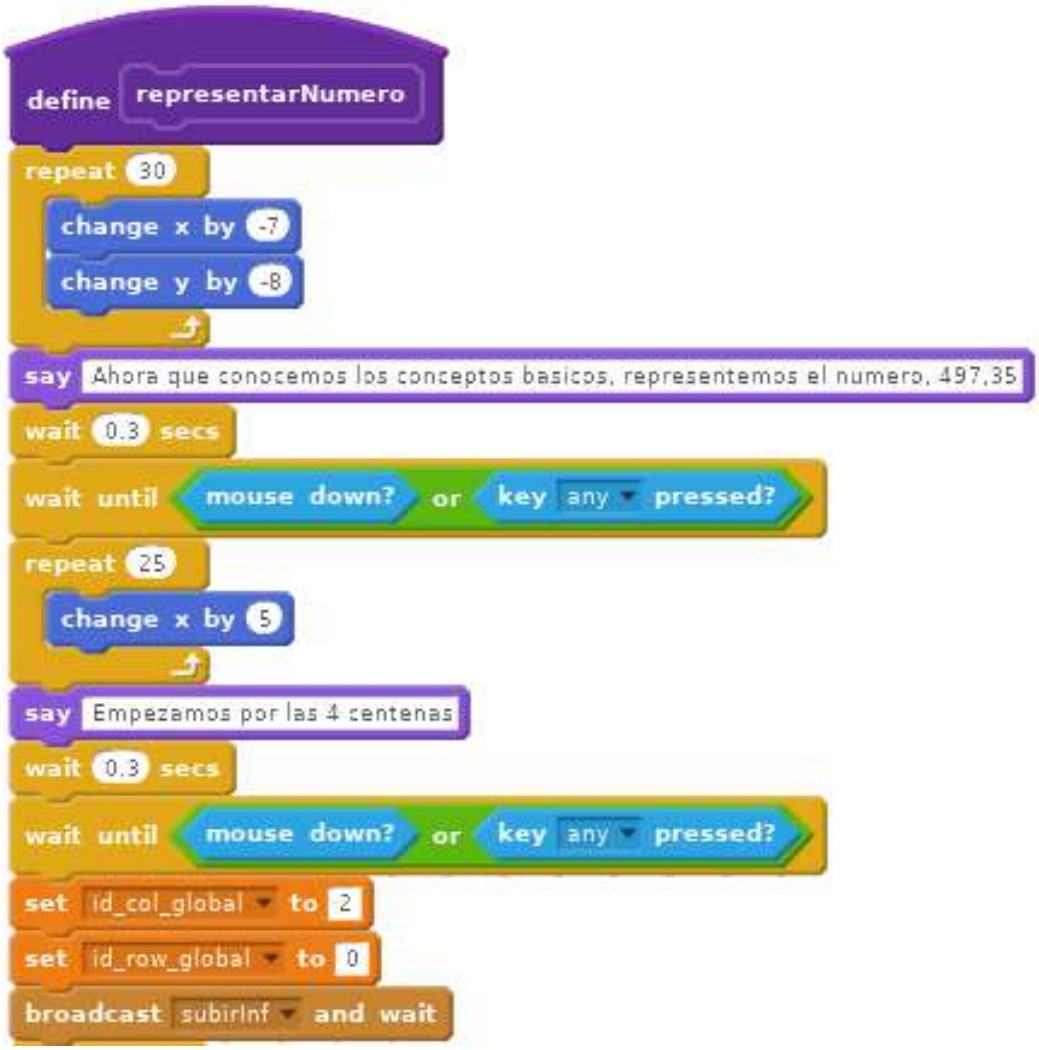

Figura 4.5: En el bloque mostrado (incompleto), extraído de uno de los tutoriales, pueden verse todas las características explicadas: comienza con una animación, prosigue con una interacción con el usuario y acaba con un envío de mensaje para interactuar con el resto de objetos del proyecto

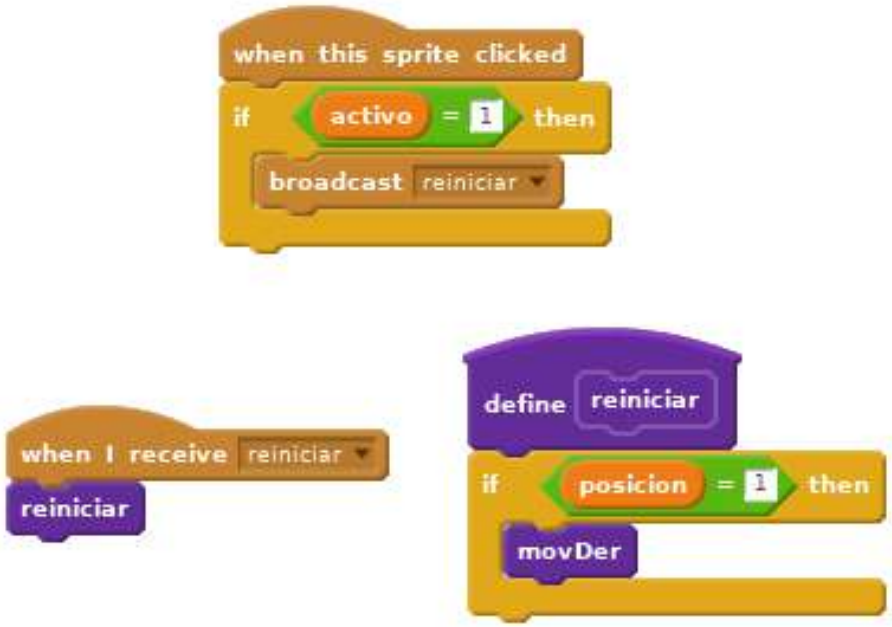

Figura 4.6: Arriba, el manejador de eventos del objeto botón **Puesta0**, que emite *broadcast* un mensaje «`reiniciar`» cuando se capta un clic sobre el botón estando el ábaco activo. Abajo, código del objeto **Cuenta** que capta el mensaje y reacciona moviendo a la derecha la cuenta si la posición actual estaba a la izquierda del ábaco, ya que este código de reinicio pertenece al *schoty*

to, ya no atendían a los clics para subir ni bajar ni podían ser utilizados para el cálculo.

Este problema se solucionó creando una variable llamada **activo** que realiza una función de *flag* de tipo booleano (metafóricamente, ya que en Scratch no hay tipos explícitos): cuando su valor es uno, el usuario puede operar con el ábaco, pero cuando está a cero, las cuentas no responden a los clics realizados por el operador. Esto se ha implementado con un simple condicional que compruebe el valor de esta variable cuando el usuario realiza un clic sobre las piezas del ábaco. La variable, por su parte, pasa a señalar cero (inactividad) en diversas situaciones: durante las décimas de segundo que dura la animación de una cuenta del ábaco, lo cual soluciona el *bug* antes descrito; durante todo el modo **Tutorial**, ya que este está diseñado para actuar como una animación no interactiva, como ya se explicó; y durante los primeros segundos tras elegir alguna de las funciones del proyecto basadas en uno de los ábacos virtuales, ya que las piezas tardan unos pocos segundos en crearse por completo y algún movimiento por parte del usuario podría resultar problemático. Véase la Figura 4.7 con algunos de los elementos que implementan esta característica de los programas.

Figura 4.7: Arriba, cuatro manejadores de mensajes en el *soroban* encargados de bloquear el ábaco durante unas milésimas de segundo cada vez que se mueve una cuenta. El bloqueo es algo menor al mover cuentas de la parte superior porque su animación es más corta. Abajo a la izquierda, bloqueo durante unos pocos segundos al comenzar el «modo libre» de uso del ábaco. Abajo a la derecha (solo el principio del bloque de código), comprobación del valor de la variable **activo** tras hacer el usuario clic sobre ella. Si vale cero, no sucederá nada. Todos estos fragmentos de código y algunos otros que no se muestran aquí se encuentran en el objeto **Cuenta**

4.3 Funcionalidades específicas

Mientras que en el apartado anterior se trató de aquellas partes del código que eran iguales o extremadamente similares en los tres proyectos, en este se cubrirán los bloques de programa que implementan una lógica claramente diferenciada entre sí. Estas instrucciones se pueden clasificar en dos categorías: aquellas que simulan la funcionalidad del *soroban*, el *suanpan* y el *schoty*, y que lógicamente son distintas entre sí; y los bloques que implementan los tres modos **Desafío**, en los que se ha intentado ofrecer unos retos bastante distintos entre sí que hagan el aprendizaje de estas herramientas más ameno para el usuario.

4.3.1. Lógica de los ábacos

Como ya se describió en el capítulo dedicado a la historia de los ábacos, el *soroban*, el *suanpan* y el *schoty* son ábacos del mismo estilo, es decir, basados en pequeñas piezas engarzadas en un tablero de madera, donde el movimiento a una posición u otra de estas piezas (arriba-abajo o derecha-izquierda) permite añadir o sustraer una cantidad y de este modo representar números y realizar cálculos. No obstante, cada uno tiene sus características y forma de operar propias. A continuación se estudiará cómo se ha implementado cada uno de sus comportamientos en código Scratch. En los tres casos este código definitorio de

la lógica del ábaco se encuentra en el objeto **Cuenta**, que representa a las piezas móviles usadas sobre el tablero para realizar los cálculos.

- **El *soroban*** es un ábaco con dos tipos de piezas, las inferiores, de valor base igual a uno, y las superiores, que equivalen a cinco inferiores (siempre dentro de la misma columna, por supuesto). La situación inicial de las cuentas se realiza con un bloque de instrucciones que realiza lo siguiente: se inicializan cuatro variables, dos para las coordenadas «x» e «y», que toman el valor del píxel en el que se quiere dibujar la primera pieza, y otras dos que representan la posición de ese primer dibujo en columnas y filas. La variable de la fila toma valor cero, representando que corresponde a la fila inferior, mientras que la variable columna toma el valor 10, lo cual puede parecer absurdo dado que representa a la varilla más a la izquierda (y por lo tanto la de más a la derecha tiene asignado un valor -2 en un ábaco con trece columnas), pero esto toma sentido posteriormente ya que nos facilitará la realización de las operaciones matemáticas necesarias para el cálculo del número representado.

  Una vez inicializadas las variables se recorren dos bucles, uno más externo para las cuatro filas inferiores y otro interior para las 13 columnas de cada fila. En cada iteración se actualizan las coordenadas, tanto del píxel como de la columna y la fila según corresponda, y se crea un clon del objeto **Cuenta**. Además, a cada uno de estos clones se le otorga otra variable de carácter local, o lo que es lo mismo, propia de ese clon en concreto: **posicion**, que representa si la pieza está bajada (**0**) o subida (**1**), lo cual determina si el número asociado a esa cuenta debe añadirse o no al total de la cantidad representada. En estos primeros bucles esta variable se inicializa a cero, indicando que todos los clones de la región inferior comienzan en posición baja. Una vez construidas las cuatro filas de abajo, un único bucle de 13 iteraciones crea las cuentas de la zona superior, siguiendo los mismos principios pero esta vez con la variable **posicion** con valor **1**. En la Figura 4.8 puede verse el bloque que inicializa el *soroban* a su situación de partida.

  Una vez realizada la inicialización, todos los clones de **Cuenta** quedan a la espera de un evento. Aunque realmente el código del objeto contiene manejadores para gran cantidad de eventos (mensajes de cambio de escena, principalmente), el que más interesante resulta es el que se activa al detectar un clic de ratón sobre cualquier instanciación del objeto. Cuando este clic se produce por parte del usuario, el clon concreto afectado comienza a ejecutar el bloque de programa de la Figura 4.9. En primer lugar, se comprueba si la variable global **activo** tiene un valor igual a uno, pues en caso contrario el programa está bloqueando la interactividad y no sucederá nada. Una vez superada esta condición, el comportamiento de la cuenta dependerá de cuatro casos posibles: si la fila del objeto sobre el que se realizó el clic es la número cuatro (o sea, la de la parte superior) y la variable **posicion** es igual a uno (la cuenta está subida), se hace una difusión de un mensaje que indica que hay que realizar un movimiento de bajada de cuentas de la parte superior. Si el clon está en la cuarta fila pero bajado, se indicará que hay que subir cuentas superiores. Por último, en el caso de que la fila no sea la número cuatro, esto significará que se trata de una cuenta de la parte inferior,

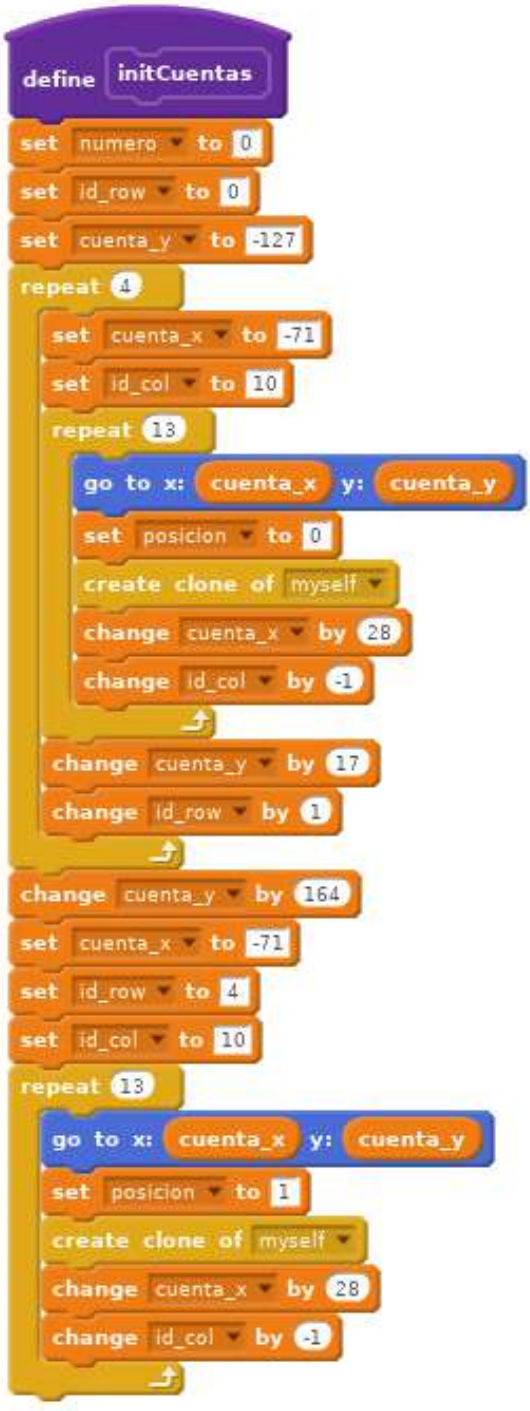

Figura 4.8: Fragmento de código del *soroban*, construido en un bloque o subrutina separado llamado **initCuentas**, y encargado de la inicialización de todas las cuentas del ábaco en sus posiciones iniciales y con valores correctos en sus variables locales (**id_row**, **id_col** y **posicion**), que serán necesarias posteriormente

por lo que se siguen los mismos principios pero adaptados a esta región: si **posicion** es igual a cero se difunde que hay que subir cuentas de la parte inferior, y si está en posición subida (igual a uno), que hay que bajarlas. En cualquiera de los casos se copian la fila y columna del objeto con el que se ha realizado la interacción a unas variables globales que resultan necesarias para los siguientes procesos.

Pero, ¿por qué realizar estas cuatro difusiones de mensajes en lugar de directamente ejecutar las instrucciones para mover el clon de **Cuenta** sobre el que se hizo clic? Esto se debe a la política de desplazamiento múltiple de piezas con un solo clic que se ha implementado. Lo cual significa que si por ejemplo las cuatro cuentas que forman la parte inferior de una varilla están en posición baja y se hace clic en la tercera de ellas (contando desde arriba hacia abajo), el comportamiento deseado será que suban las tres a la vez como se haría en un ábaco real si se empujasen con un dedo. Para ello es necesario realizar una difusión de mensaje a todos los elementos del ábaco y que cada uno de ellos compruebe si cumple las condiciones para moverse. ¿Cuáles son esos condicionales necesarios? Para la acción consistente en subir cuentas de la región inferior, por ejemplo, todo objeto que se vaya a desplazar debería cumplir las siguientes condiciones: estar en posición bajada, que su columna sea la misma que la de la cuenta sobre la que se realizó el clic (aquí es donde resultan necesarias las variables globales de columna y fila antes mencionadas, donde se guardaron estas coordenadas del clon activado por el usuario); y que su fila sea menor que cuatro, es decir, que pertenezca a la parte inferior, pero mayor o igual que la del objeto activado por el usuario. Los otros tres casos de posibles movimientos son similares, aunque algo más sencillos en el caso de las condiciones para las cuentas de la región superior, ya que cuentan con una única fila.

Una vez superados estos condicionales, implementados con instrucciones de control y comparadores booleanos, se activa el correspondiente de los cuatro bloques (**subirInferior**, **subirSuperior**, **bajarInferior** y **bajarSuperior**) que gestionan la animación del desplazamiento, reproducen un pequeño sonido, realizan el cálculo matemático y cambian la variable **posicion** de los clones afectados. La animación se realiza del mismo modo explicado en el capítulo anterior, y simplemente desplaza un número fijo de píxeles sobre el eje «y» para la subida, y el mismo número pero con signo negativo para la bajada. Ese número es menor para las piezas de la parte superior, ya que sus desplazamientos son más cortos. Por último, el cálculo del número representado se realiza del siguiente modo: existe una variable llamada **numero** que se inicializa a cero con el ábaco, con lo que se corresponde con la disposición inicial de las cuentas del *soroban*. Cuando uno de los clones se desplaza hacia el centro, incrementa esa variable en 10 elevado al identificador de la columna (10^{id_col}), o en 5×10^{id_col} si se trata de una cuenta de la parte superior. En el caso de que las cuentas se estén alejando de la parte central, se resta una cantidad equivalente del total guardado en **numero**. En este punto nos resulta muy útil la decisión previamente comentada de numerar de 10 a -2 las columnas del *soroban*, de tal forma que ahora resulta más sencillo calcular el valor de cada cuenta, el cual variará entre 10^{10} en la varilla más a la izquierda y 10^{-2}, es decir, una centésima, en la varilla más

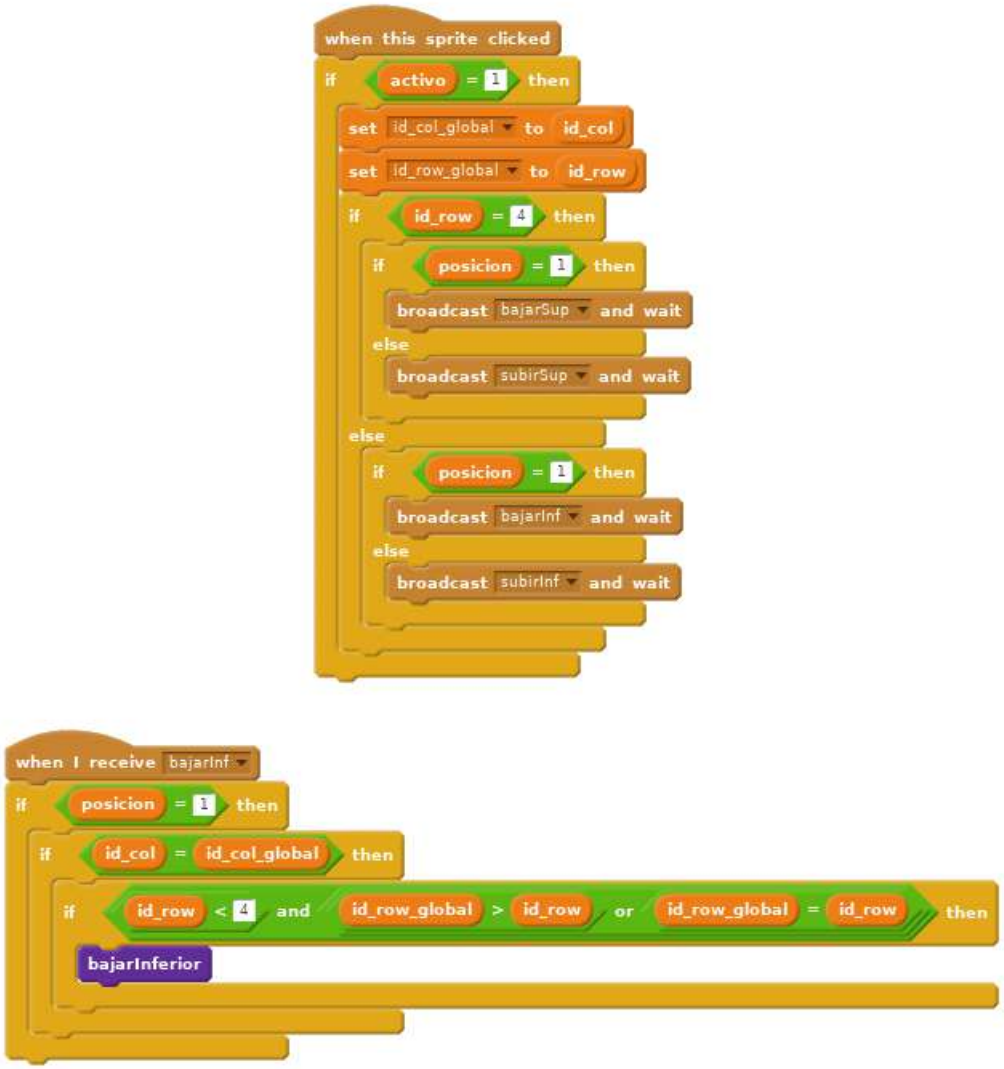

Figura 4.9: El código mostrado maneja los clics realizados por el usuario sobre un objeto **Cuenta** en el programa sobre el *soroban*. El bloque de arriba decide el mensaje a enviar con cuatro posibles órdenes según los valores de **posicion** y de **id_row**. El bloque de abajo es uno de los cuatro que atienden a las órdenes enviadas por el de arriba. Comprueba mediante condicionales y comparaciones booleanas que el objeto **Cuenta** que lo reciba cumpla unos determinados requisitos antes de efectuar el movimiento

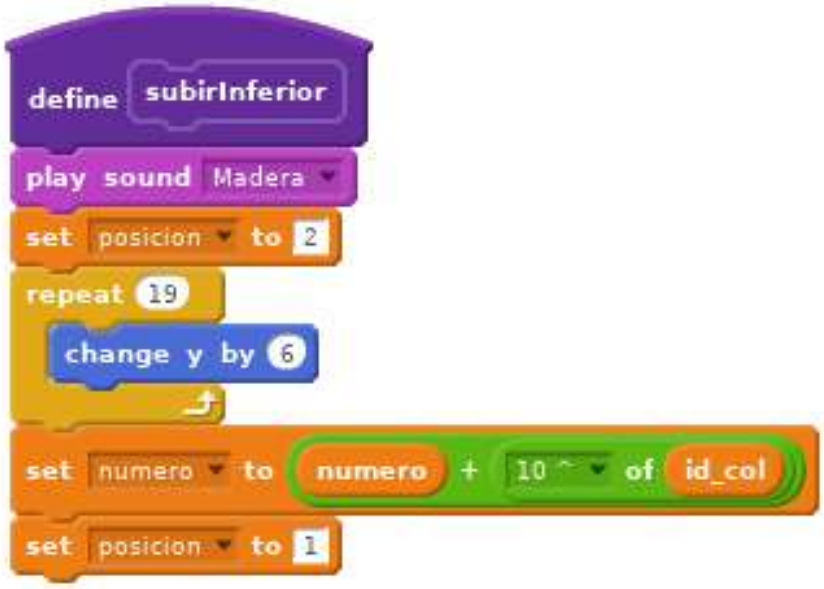

Figura 4.10: Este bloque de código se ejecuta cuando se decide que una cuenta de la región inferior debe realizar un movimiento ascendente. En primer lugar reproduce un sonido similar al chasquido de una madera, a continuación sitúa la variable **posicion** a **2**, un valor no definido que evita que nadie interactúe con esta cuenta mientras tanto. Por último, realiza una animación de ascenso sobre el eje «y» y aumenta el número representado en 10^{id_col}, además de indicar en **posicion** que ahora la cuenta está subida

a la derecha. En la Figura 4.10 se presenta como ejemplo uno de estos bloques encargado de implementar el movimiento de las cuentas y el cálculo asociado a él. Los otros tres casos son muy similares.

- El caso del ***suanpan*** es extremadamente similar al del *soroban*, ya que la única diferencia sustancial reside en que el ábaco chino posee dos cuentas más por cada varilla, una en la zona inferior y otra en la superior. Además, en este caso el tablero posee 12 columnas en lugar de las 13 del *soroban*, ya que aquel suele ser más amplio, pero ninguno de los dos tiene en realidad un número fijo de varillas, por lo que la elección de estos números ha sido algo arbitraria. La inicialización del ábaco chino se realiza, por lo tanto, igual que se hizo con el *soroban*, pero cambiando el número de iteraciones de los bucles, que en este caso serán de tamaño 5×12 para las filas inferiores y 2×12 para las superiores. El resto del código se compone de las mismas partes: manejo del evento de clic sobre el objeto, difusión de mensajes indicando qué tipo de movimiento se debe realizar, comprobación de cuáles son los clones que deben ser desplazados y, finalmente, la ejecución de la animación y del correspondiente cálculo. Todo ello es equivalente a lo explicado para el caso del ábaco japonés.

- Más interesante resulta el caso del ***schoty***, donde las diferencias son más profundas. En primer lugar, el ábaco ruso se coloca en posición vertical y las cuentas se desplazan sobre sus varillas en horizontal y de derecha a izquierda, lo cual, aunque parezca una diferencia nimia, obliga a plantear de otro modo el diseño, los nombres de variables y funciones (filas por colum-

nas y viceversa), o a desplazar las cuentas sobre el eje «x» en lugar del «y». El proceso de inicialización también cambia bastante debido a la fila que solo consta de cuatro cuentas y a que las dos piezas centrales en cada fila son negras, mientras que todas las demás son de color blanco. Esto complica bastante el código y, para que el bloque no resultase demasiado largo y farragoso y mejorar la legibilidad se crearon tres bloques (subrutinas) para funcionalidades recurrentes: **crearCuentaBlanca** y **crearCuentaNegra**, que generan clones con el disfraz correspondiente, blanco o negro, del objeto **Cuenta** en la posición dada por unas coordenadas, y al finalizar actualizan dichas coordenadas para el siguiente clon. Los clones blancos y negros son equivalentes por completo salvo en el aspecto visual. Por otra parte, el bloque de código **crearFilaLarga** crea toda una fila de cuentas llamando primero cuatro veces a **crearCuentaBlanca**, a continuación dos veces a **crearCuentaNegra**, y por último genera otros cuatro clones con disfraz blanco.

Haciendo uso de estas subrutinas, el bloque **initCuentas** se encarga de inicializar las variables para, acto seguido, crear siete filas largas mediante un bucle simple. A continuación se genera la fila de cuatro cuentas fijando sus coordenadas y llamando directamente a las funciones de creación de cuentas. Lo más llamativo de esta fila es que a su identificador de columna se le asigna el valor $\log_{10} 0,25$, ya que ese número, usado después como exponente, permitirá el cálculo sencillo con cuartos de unidad, ya que $1 \times 10^{\log_{10} 0,25} = 0,25$. Por último, con un nuevo bucle, en este caso de tres iteraciones, se crean las últimas tres filas largas, de 10 piezas cada una. Esta inicialización del *schoty* puede observarse en la Figura 4.11.

La atención a eventos, y más concretamente al clic realizado por el usuario sobre el objeto, sigue el mismo patrón que en los casos del *soroban* y del *suanpan*. Sin embargo, al tener el tablero una única «región», a diferencia de los otros dos, solo puede haber dos casos posibles de desplazamiento: mover a la izquierda si la **Cuenta** está en posición cero (estado inicial, a la derecha del tablero), o mover a la derecha si la posición fuese igual a uno.

El proceso por el que una cuenta comprueba si debe desplazarse o no tras recibir la correspondiente difusión es también similar: una cuenta se deberá desplazar a la derecha, por ejemplo, si su variable **posicion** es igual a uno (izquierda), si su varilla es la misma que la del clon con el que el usuario interactuó y si su columna es mayor o igual que la de aquel. Por último, solo hay dos bloques de código para desplazar una cuenta: de nuevo el de movimiento a la derecha y el de movimiento a la izquierda. El comportamiento es en esencia el mismo que el descrito para las cuentas inferiores de los ábacos chino y japonés, con la importante excepción de que se realiza una comprobación condicional para reconocer a la varilla especial de cuatro cuentas, ya que en tal caso la animación recorre más espacio. Por último, el cálculo matemático utiliza la misma fórmula en todos los casos al no haber cuentas de valor base diferente, y en todos los casos se suma o se resta 10^{id_row} al resultado. Esta fórmula sirve para todos los casos, incluido el de la fila especial, ya que se tuvo la precaución de asignar los exponentes adecuados (**id_row**) a cada fila durante la inicialización. En la Figura 4.12 se muestra el código que gestiona estas funciones para el *schoty*.

Figura 4.11: Código de inicialización del ábaco ruso. A la izquierda, bloque de código que controla el flujo principal de la inicialización, creando (de arriba a abajo) primero siete filas largas, después la más corta de cuatro cuentas, y finalmente otras tres largas, además de encargarse de la asignación de valores a algunas variables. A la derecha arriba, código del bloque auxiliar **crearFilaLarga**, que genera una fila normal del *schoty*, formada por cuatro cuentas blancas, dos negras y otras cuatro blancas. Por último, abajo a la derecha, código para generar un clon de una cuenta blanca en las coordenadas «x» e «y» con su posición bajada. El código de **crearCuentaNegra** es idéntico pero poniendo el disfraz (`«switch costume to»`) a **CuentaNegra**

Figura 4.12: Movimiento hacia la derecha (de vuelta a la posición original) de una **Cuenta** en el *schoty*. Realiza las mismas funciones vistas para el caso del *soroban* en la Figura 4.10, pero con una comprobación específica para descubrir si la fila actual es la que solo posee cuatro cuentas, específica del ábaco ruso

4.3.2. Tres modos de juego

Para introducir algo más de variedad en los proyectos y hacerlos más amenos y didácticos para el público infantil, se decidió desarrollar tres modos de juego totalmente diferentes entre sí, cada uno asociado a uno de los ábacos tratados. La lógica de estas funcionalidades se encuentra en todos los casos en el código del objeto **Ghoul**. Gran parte de las instrucciones están dedicadas a la animación de este objeto o a la comunicación con el usuario para transmitirle el objetivo y funcionamiento del juego y que pueda elegir entre las opciones planteadas. No obstante, puesto que entrar de nuevo en detalles sobre esos temas resultaría repetitivo y poco interesante, se obviará esa parte del código y se pasará a describir directamente las partes que implementan la lógica esencial de cada modalidad.

- **Las operaciones aritméticas** básicas constituyen el desafío planteado al usuario en la aplicación del *soroban*, de tal modo que se pide que simplemente resuelva la operación planteada. Para lograr este objetivo, se le pregunta qué tipo de operación querría practicar y se entra en un bucle que bloquea el programa hasta que una de las letras sugeridas para la selección de operación ha sido elegida. Una vez seleccionada una de las opciones, una serie de «`si... si no`» («`if... if else`») comprueban el modo elegido y según ello plantean uno de los siguientes problemas: suma, resta, multiplicación o división. Puede observarse cómo se implementa esta funcionalidad de selección de opciones por parte del usuario en la Figura 4.13.

  Las cuatro operaciones aritméticas implementadas siguen un patrón similar, en el que primero se eligen dos operandos al azar, aunque en el caso de la resta y de la división se garantiza que el segundo número será siempre igual o menor que el primero. Elegidos los operandos, se visualiza por pantalla la operación a realizar utilizando una cadena de texto, construida gracias a la función «`unir`» («`join`») de Scratch, y se almacena internamente la solución al problema. Por último, se aguarda hasta que el número representado en el ábaco coincide con el resultado de la operación (confiando en que el usuario actúa de buena fe y no resuelve el problema por otros medios para después simplemente representar el número), y finalmente el **Ghoul** da la enhorabuena al jugador y le ofrece la posibilidad de seguir resolviendo operaciones. De nuevo se acepta un *input* por parte del usuario de tipo «S/N», y si decide seguir adelante con otro problema, se activa una variable *flag* utilizada como guarda en el bucle del programa y se vuelve a repetir todo el proceso a excepción de la introducción animada. El bloque de instrucciones que implementa la estructura principal y el flujo de este modo de juego se puede estudiar en la Figura 4.14.

- **El modo cronometrado**. En el caso del *suanpan*, el reto que se plantea es la resolución de una suma o una resta, o incluso de la simple representación de un número usando el ábaco chino. Sin embargo, lo que hace interesante a este juego es la presencia en pantalla de un cronómetro que indica en todo momento los segundos transcurridos desde el instante inicial en el que se empezó a operar. Una vez alcanzada la solución, Ugluk, el **Ghoul**, informa al usuario de si ha logrado o no batir su mejor marca anterior. En caso afirmativo, el contador que indica la mejor marca hasta el momento (hay tres

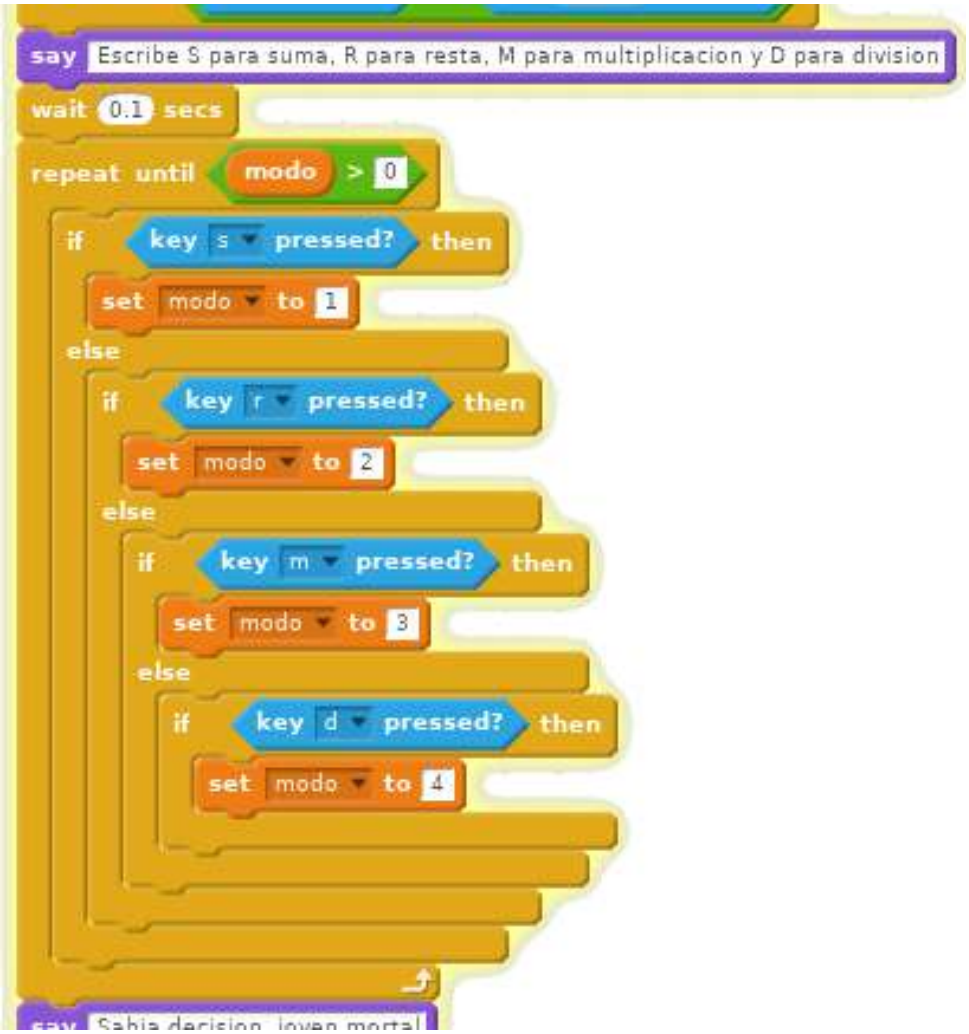

Figura 4.13: Fragmento de código dentro del objeto **Ghoul** del proyecto sobre el *soroban*. Este bloque, tras pedir un *input* por teclado repite indefinidamente un bucle «`repeat until`» hasta que se detecta que una de las teclas demandadas ha sido apretada. En ese momento se elige el modo correspondiente y se sale del bucle para continuar con el problema seleccionado

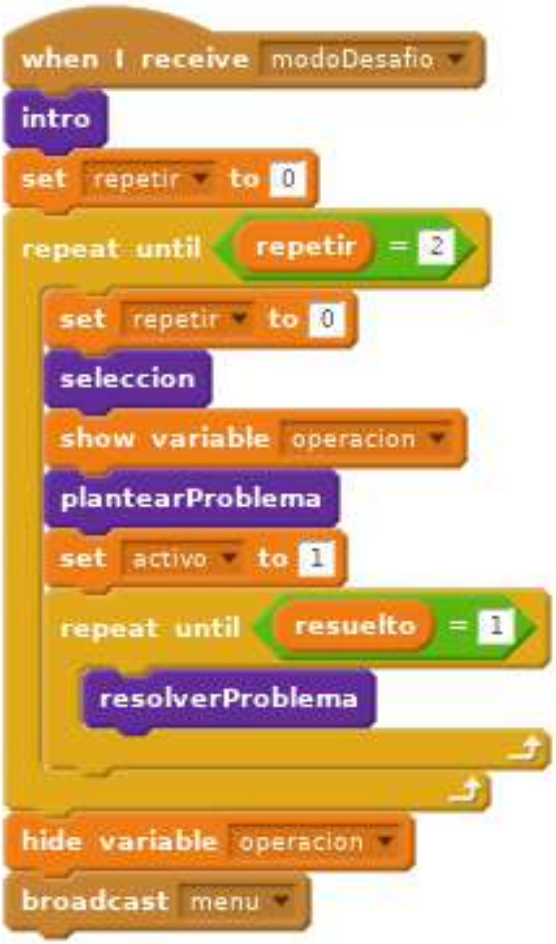

Figura 4.14: Bloque controlador del flujo de programa del «modo desafío» del *soroban*. Tras recibir el mensaje de activación, ejecuta un bloque de animaciones introductorias (**intro**), y a continuación repite en bucle (hasta que el usuario seleccione que no quiere continuar) la estructura del problema: **seleccion** de la operación aritmética a realizar, planteamiento del problema (**plantearProblema**) y, hasta que se detecte que se ha resuelto, un bloque **resolverProblema**. Finalmente, cuando el usuario decide que no quiere seguir jugando, se difunde una orden («`broadcast`») de regreso al menú

variables independientes entre sí, una para cada operación) se actualiza al nuevo valor. Por último, se ofrece al usuario la posibilidad de intentarlo otra vez si quiere seguir jugando. La definición de esta estructura de programa se encuentra en el código de la Figura 4.15.

En este modo, como en los otros dos, la mayor parte del código consiste en animaciones, mensajes por pantalla y obtención de respuestas desde teclado por parte del usuario para tomar algunas decisiones. La parte más original de este **Desafío** reside en lo relacionado con el cronómetro y las marcas de mejor tiempo. Este sistema se implementa mediante la inicialización a un valor arbitrariamente alto de los mejores tiempos en cada operación, para que así la primera marca realizada por el usuario los sobreescriba en cualquier caso. Esta inicialización se produce tan solo una vez al comenzar la ejecución del programa, por lo que los nuevos mejores tiempos logrados por el usuario no son borrados y resultan persistentes durante toda la ejecución (pero no después).

El cronómetro, por su parte, se inicia con una difusión de mensaje justo antes del bucle que se recorre continuamente a la espera de que se encuentre la solución, y se detiene con la difusión de otro mensaje, realizada tras salir del bucle una vez el usuario ha dado con la solución al problema planteado. El bloque de código que implementa el cronómetro es extremadamente sencillo, y consiste en un pequeño bucle que no se detiene hasta que es informado de que el problema ha sido resuelto. Este bucle realiza a cada iteración una espera de exactamente un segundo para acto seguido incrementar en una unidad el cronómetro. Por último, se realiza una comparación sencilla entre lo indicado por el reloj y la mejor marca anterior para el tipo de operación realizada y, si el nuevo tiempo es menor, sobreescribe el anterior y el **Ghoul** felicita al usuario por su gesta. Los fragmentos más interesantes de este código se presentan en la Figura 4.16.

- **El modo «a contrarreloj»**. Por último, en el *schoty* se plantea al usuario la posibilidad de realizar una serie de operaciones de suma y resta nuevamente, pero esta vez con un límite máximo de tiempo. El usuario elige un nivel de dificultad entre tres posibilidades, el cual afecta a la magnitud de los operandos planteados en los problemas, pero no al número y tipo de operaciones ni al tiempo total disponible. Una vez seleccionada la dificultad se reinicia el tiempo y se envía un mensaje para inicializar el **Reloj** (planteado de forma diferente a la del *suanpan*, como a continuación se describirá). Por último, se entra en el bucle de planteamiento y resolución del problema, del que se sale al resolver un total de tres operaciones, en cuyo caso se felicita al usuario por superar la prueba; o, en caso contrario, si no se han resulto las operaciones al haber transcurrido los 60 segundos que se concedían como tiempo máximo, se sale del bucle y el personaje **Ghoul** realiza algunas mofas jocosas hacia el jugador por su fracaso. En este caso no hay bucle para repetir el juego, por lo que el flujo de este modo de juego acaba siempre en una difusión de mensaje que direcciona al menú principal. Puede verse esta estructura en la Figura 4.17.

  Los dos elementos de mayor interés en este modo de juego son el generador de operaciones aleatorias y el reloj de arena. La función o bloque encargado

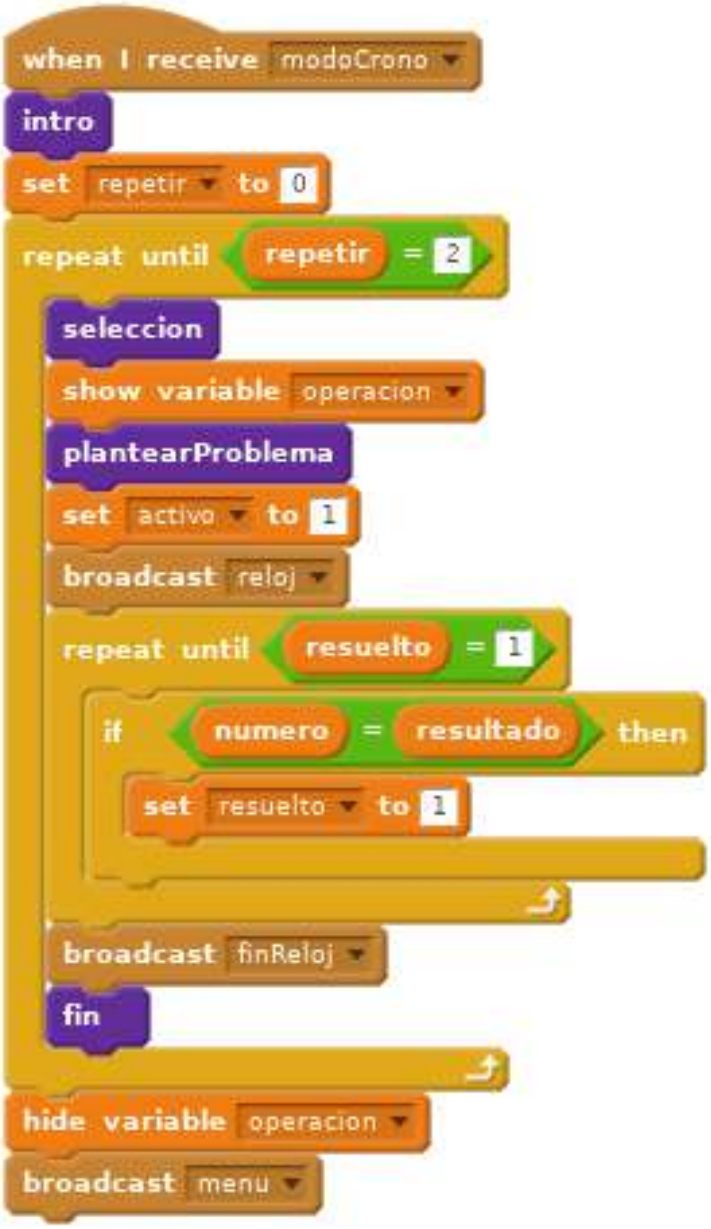

Figura 4.15: Bloque controlador del flujo de programa en el modo «cronometrado» del *suanpan*. Tras una **intro** animada, se entra en un bucle del que no se saldrá hasta que el usuario lo decida. En este bucle el usuario realiza una **seleccion** de la operación que desea resolver, se le realiza el planteamiento del problema en cuestión (**plantearProblema**) y, tras difundir una orden de inicio del reloj, se entra en el bucle de resolución del problema, del que no se sale hasta que el número representado por el usuario sea igual a la solución del problema. Por último, una vez hallada la solución se envía una orden para terminar el reloj y se ejecuta un bloque de **fin**, que muestra algunos mensajes por pantalla de felicitación y pregunta al usuario si desea seguir jugando. Si la respuesta es negativa, se sale definitivamente del bucle externo y se regresa al menú principal

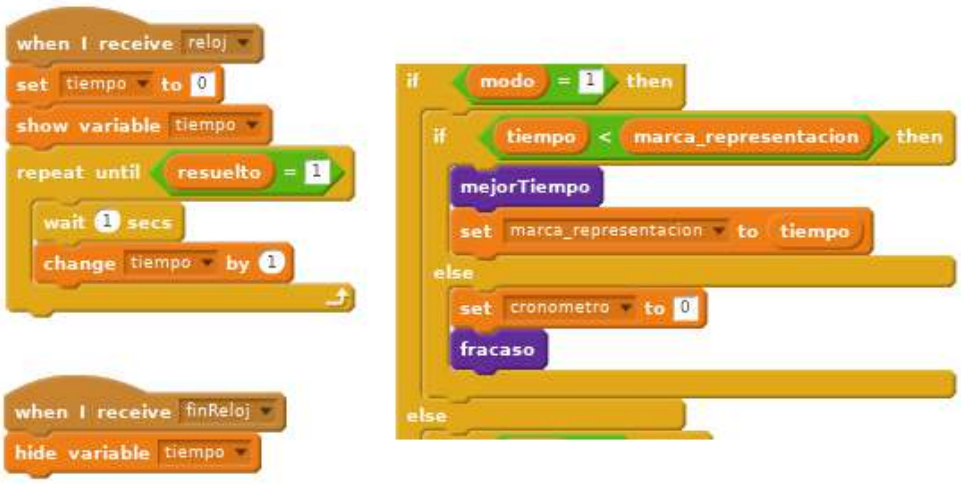

Figura 4.16: A la izquierda, código encargado de implementar la función de cronómetro, dentro del objeto **Ghoul**, así como el bloque para esconderlo una vez resuelto el problema. A la derecha, un fragmento de la serie de condicionales «`if`» e «`if...else`» que comprueban en primer lugar la operación que se estaba realizando, y a continuación si se ha superado la mejor marca, en cuyo caso se felicita al usuario y se sobreescribe el nuevo tiempo en la variable

de crear operaciones al azar funciona del siguiente modo: elige en primer lugar un operando aleatorio de uno u otro tamaño según la dificultad elegida por el usuario. A continuación se selecciona un segundo operando que será siempre menor que el primero, para evitar potenciales números negativos en el caso de las restas. Por último, se elige el tipo de operación a realizar, también al azar. Esto se consigue haciendo un sorteo entre dos números, el cero y el uno. En función del contenido de la variable que guarda ese resultado se selecciona la operación, una suma o una resta en cada uno de los respectivos casos. En la Figura 4.18 se puede estudiar cómo implementar el sencillo generador aleatorio de operaciones planteado.

Para acabar, el reloj que se propone en la aplicación del ábaco ruso consiste en un objeto independiente (**Reloj**) que posee un total de seis disfraces. Al recibir el mensaje que activa el reloj, y justo después de reiniciar el cronómetro (en este caso se ha utilizado el facilitado por Scratch), se pone en marcha un bloque de código que hace visible el objeto con su primer disfraz. Este primer «fotograma» del objeto consiste en un reloj de arena con todo su contenido en la parte superior del mismo. No obstante, mientras no se reciba el mensaje indicando que el juego ha acabado y el usuario ha resuelto las tres operaciones, el **Reloj** entra en un bucle en el que se comprueba continuamente el valor del cronómetro y, cada vez que se detecta que se ha superado un múltiplo de 10 (10, 20, 30, 40 y 50), se pasa al siguiente disfraz. Cada uno de estos disfraces contiene menos arena en la parte superior y más en la inferior, proporcionando al usuario una sensación de que la arena va cayendo, así como una idea bastante aproximada por métodos gráficos del tiempo que le resta para resolver las operaciones. En la Figura 4.19 se muestran los distintos disfraces del **Reloj** y en la Figura 4.20 puede estudiarse el código que implementa esta funcionalidad.

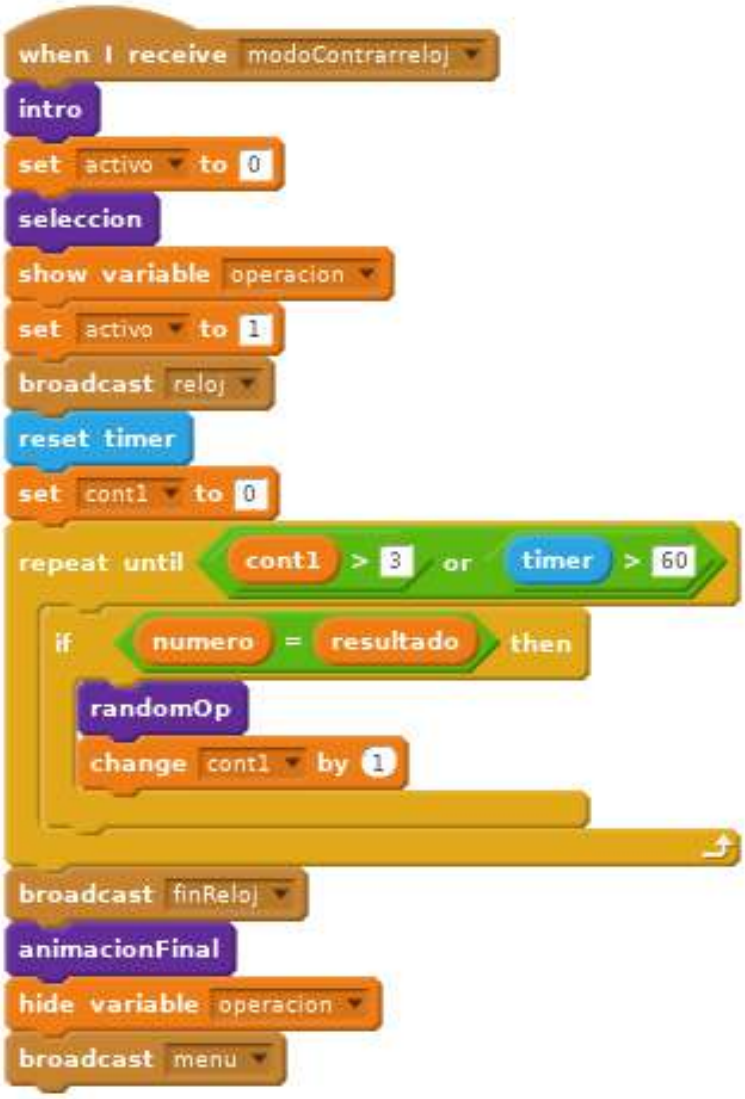

Figura 4.17: Bloque de código encargado de implementar el flujo principal en el «modo de desafío a contrarreloj» del *schoty*. El esquema es similar a los otros dos casos, aunque con algunas diferencias: tras una **intro** y una **seleccion** del problema, se realiza una inicialización del reloj y se entra en el bucle de resolución del problema, del que se sale cuando el usuario halle la solución a tres operaciones o cuando se agote el tiempo. Mientras tanto, cada vez que el usuario halla la solución a un problema, se le plantea otro mediante un generador de operaciones aleatorias (**randomOp**). Por último, se hace una llamada para terminar el reloj, se muestra una **animacionFinal** y se regresa al menú

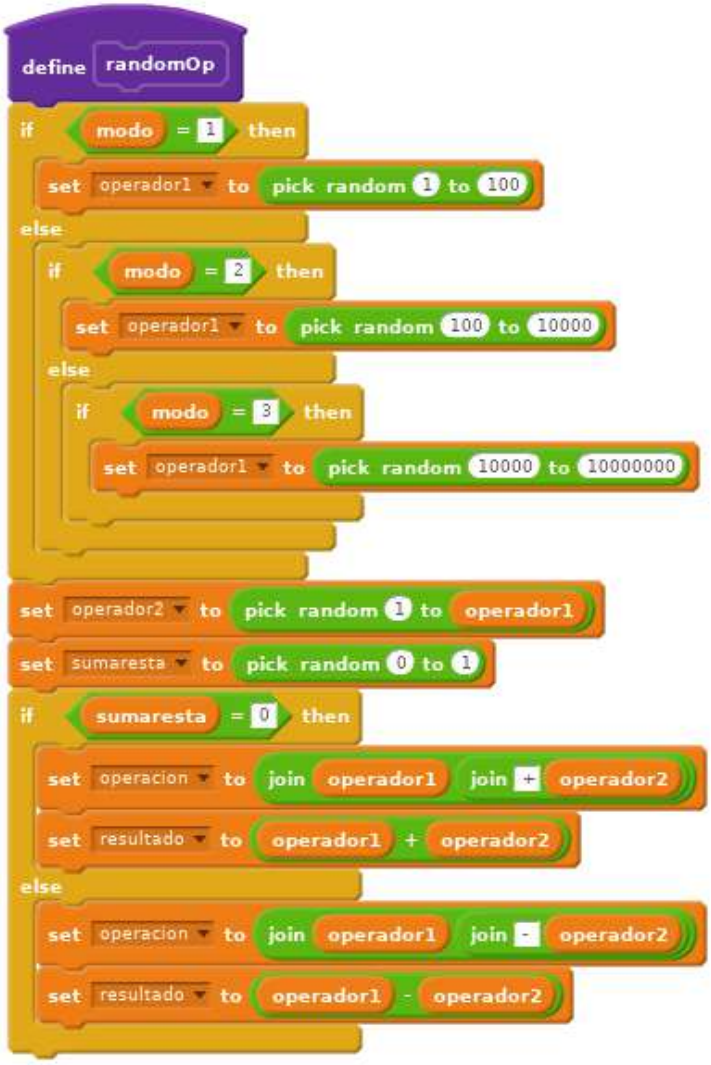

Figura 4.18: Función generadora de operaciones (de suma y resta) aleatorias. Elige un número aleatorio que puede ser de tres magnitudes distintas según la dificultad, y a continuación elige un segundo operando menor que el primero y una de las dos operaciones posibles, todo ello al azar. Por último, lo une todo en una cadena de texto para mostrársela al usuario y calcula el resultado de la operación

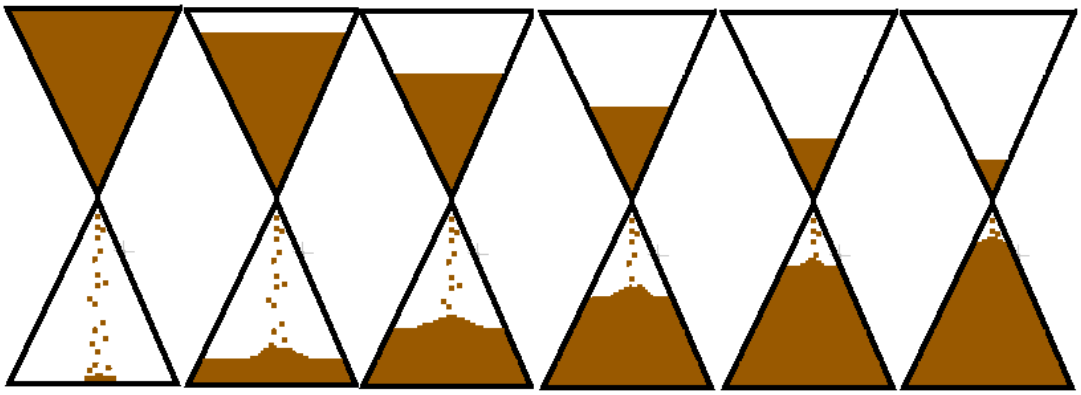

Figura 4.19: Los seis disfraces del objeto **Reloj**. El cambio de disfraces de acuerdo con el paso de los segundos implementa otro tipo de animación, dotando a un proyecto de más posibilidades en el aspecto visual. En este caso se logra dar al usuario una noción del tiempo transcurrido a través de un elemento gráfico

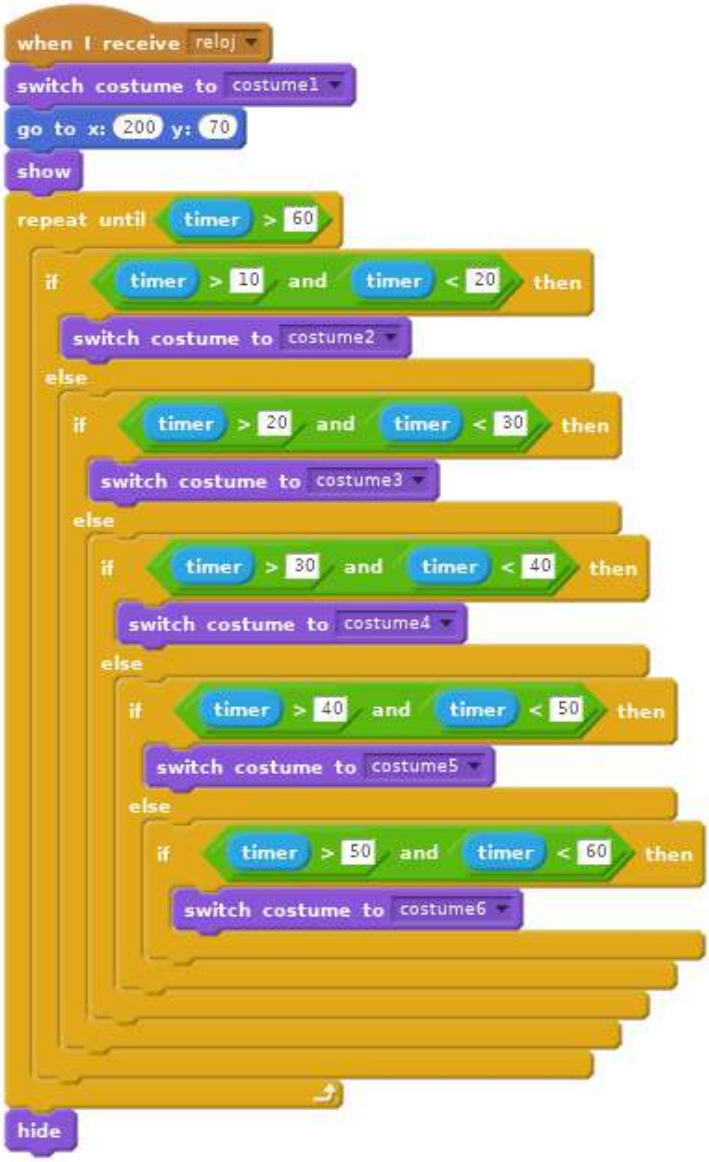

Figura 4.20: Implementación del funcionamiento de reloj de arena. Se trata básicamente de una animación gráfica, pero en este caso no hay desplazamiento sino cambios de disfraces. Hasta que el cronómetro supera los 60 segundos, se recorre continuamente un bucle en el que múltiples condicionales comprueban el número de segundos transcurridos y en función de ello cambian a uno u otro disfraz. El resultado es que cada 10 segundos se produce un cambio de disfraz, dando la sensación de que el objeto va mutando

CAPÍTULO 5
Conclusiones

En este último capítulo de la memoria se realiza una mirada retrospectiva al trabajo desarrollado y se hace una evaluación con respecto a los objetivos que se fijaron al comienzo del mismo, así como una reflexión *a posteriori* de todo lo realizado y de las dificultades superadas. Además, se plantean algunas ideas sobre cómo se podrían expandir en un futuro las líneas desarrolladas.

5.1 Consideraciones finales

Los dos objetivos principales de este trabajo eran dar una visión global del ábaco que incluyese aspectos tanto históricos como técnicos y, por otra parte, desarrollar varias aplicaciones informáticas sobre este tema utilizando el lenguaje de programación Scratch, aparentemente sencillo, pero con potencia suficiente para realizar programas de relativa complejidad como los presentados.

Más concretamente, en el aspecto histórico se ha proporcionado en primer lugar una visión general del progreso de las máquinas calculadoras desde la Prehistoria hasta nuestros días. Con ello se pretende que el lector sitúe las computadoras actuales en un contexto mucho más amplio, llegando a considerarlas simplemente como el último (de momento) y más complejo paso en la larga historia del ser humano por facilitar o incluso automatizar el cálculo matemático. Durante esta revisión histórica se han presentado y explicado el funcionamiento de numerosos artefactos complejos, ingeniosos y más o menos provechosos desarrollados durante miles de años.

Siendo el ábaco el núcleo alrededor del cual se ha realizado todo el estudio, la parte principal de la investigación histórica se ha centrado también en este utensilio. Partiendo de la tabla o mesa de cálculo como antecesor del ábaco actual, se ha realizado un repaso a gran cantidad de ábacos y utensilios similares (muchos de los cuales sería cuestión de debate si podrían ser clasificados dentro de la categoría del «ábaco», como el *quipu* o la *yupana*) de distintos lugares y épocas. Sin profundizar en la mayoría de ellos, se ha pretendido ofrecer una perspectiva completa de la historia de este utensilio que pervive hasta nuestros días.

Los tres ábacos en los que sin duda se ha incidido más por su popularidad y continuidad hasta la actualidad han sido los ábacos japonés, chino y ruso, llamados en sus idiomas *soroban*, *suanpan* y *schoty*, respectivamente. Usando uno

de ellos como ejemplo, en concreto el *soroban*, se ha realizado una explicación de cómo operar de forma correcta y eficiente. Asimismo, se han explicado algunos algoritmos y técnicas para realizar las cuatro operaciones aritméticas sobre el ábaco japonés de forma tan rápida y eficaz como en una calculadora electrónica (siempre que se adquiera la destreza manual suficiente, eso sí). Aunque no se ha realizado esta explicación para los casos del *suanpan* y del *schoty*, sí que se ha señalado cómo extrapolar estas técnicas de cálculo de un ábaco a otro, lo cual resulta extremadamente sencillo, al menos entre los que pertenecen a la categoría de «tablero con cuentas engarzadas».

En el tercer capítulo se cambia totalmente de temática y se trata la historia, concepto y funcionamiento del lenguaje de programación Scratch, así como del entorno asociado a él. Se trata pues de una pequeña introducción necesaria a este lenguaje para comprender posteriormente el otro objetivo principal del trabajo: la creación de tres proyectos que implementen en este lenguaje los tres ábacos destacados anteriormente. Para ello, se trata brevemente la historia y el propósito de este lenguaje orientado a lo visual e interactivo, y creado en sus inicios con la intención de fomentar el aprendizaje de la programación en niñas y niños. Pese a ello, y a ser en apariencia extremadamente simple, se demuestra que es posible implementar programas informáticos arbitrariamente complejos siempre que se cuente con una serie de instrucciones básicas mínimas. Ello no quita, no obstante, que ciertas funcionalidades de bajo nivel sí que resultarían imposibles de implementar en este lenguaje. Sin embargo, esas necesidades quedan fuera de las intenciones de este trabajo, como quedaban fuera de los objetivos de los creadores de Scratch.

Una vez se han adquirido ciertas nociones sobre el funcionamiento de los ábacos y del lenguaje y entorno de programación Scratch, se hace converger a los dos caminos para lograr alcanzar el segundo objetivo: la implementación de tres aplicaciones informáticas alrededor de los ábacos japonés, chino y ruso. Cada uno de estos tres proyectos Scratch posee un ábaco virtual, cada uno de ellos implementado de forma coherente tanto en lo gráfico como en su funcionamiento con sus contrapartidas analógicas. Alrededor de este ábaco virtual, núcleo de los tres programas, se implementan diversas funcionalidades adicionales como tutoriales, distintos retos planteados al usuario o una sección informativa sobre el utensilio. El resultado final son tres aplicaciones de carácter muy visual e interactivo, adecuadas sobre todo para contextos didácticos y lúdicos.

El código desarrollado se ha descrito de forma bastante detallada en el texto, haciendo hincapié en los fragmentos que pudieran resultar algo más complicados de comprender o que supusieron una mayor dificultad en su implementación. A la vez, se ha intentado evitar repeticiones innecesarias en la explicación de los programas, ya que, al no permitir Scratch el uso de código de terceros proyectos a través de funciones importadas, pero sí la copia directa, gran parte de lo desarrollado en los tres programas es exactamente igual o muy similar. En cambio, se han intentado explicar todas las partes claramente diferenciadas o genuinas de cada aplicación para así cubrir todas las funcionalidades desarrolladas.

Por último, se ha creado una sencilla página web usando el lenguaje de marcado HTML para proporcionar al público general un *front end* desde el cual poder acceder a los proyectos Scratch embebidos en la página, y también facilitar un

pequeño contexto histórico similar al desarrollado en el trabajo, pero de forma bastante más resumida y accesible.

5.2 Trabajo futuro

El trabajo desarrollado podría expandirse a través de dos líneas fundamentales: por un lado, sería posible ampliar los tres proyectos ya construidos, añadiéndoles nuevas funcionalidades en forma de modos de juego distintos o aspectos que los hiciesen más atractivos o completos: tal vez mejorar el aspecto sonoro, con más efectos o una posible música de fondo con la posibilidad de ser desactivada, tutoriales sobre operaciones aritméticas o el uso de las variables en la nube (*cloud*) que proporciona Scratch para implementar una cierta persistencia de datos entre ejecuciones aunque fuese a un nivel básico.

Por otro lado, como se comentó en el capítulo de contexto histórico, la categoría de los ábacos es muy amplia, y la implementación mediante Scratch podría extenderse a muchos otros de estos artefactos. Esta podría ser una forma de contribuir a la popularización de utensilios casi totalmente desconocidos entre el gran público, como las tablas de cálculo, los *quipu*, el ábaco romano, etcétera. El desarrollo de cualquiera de estas herramientas de cálculo sería perfectamente factible con Scratch.

Bibliografía

[1] Bernazzani, David. *Soroban Abacus Handbook*. http://sliderulemuseum.com el 19 de abril de 2018

[2] Fernandes, Luis. *The Abacus: A Brief History*. Consultado en: https://www.ee.ryerson.ca/~elf/abacus/history.html

[3] García Serrano, Jaime. *Manual para el buen uso del ábaco*. Consultado en: https://www.scribd.com/doc/17623188/manual-para-el-buen-uso-del-abaco el 19 de abril de 2018

[4] Dalakov, Georgi *The abacus*. Consultado en: http://history-computer.com/CalculatingTools/abacus.html

[5] Ifrah, Georges. *Universal history of computing*. John Wiley & Sons, Inc., Nueva York, 1ª edición, 2000.

[6] Ifrah, Georges. *Historia universal de las cifras*. Espasa Calpe, S.A., Madrid, 6ª edición, 2008.

[7] Kojima, Takashi. *The Japanese Abacus: Its Use and Theory*. Charles E. Tuttle Publishers, Rutland, 25ª edición, 1970.

[8] League for Soroban Education of Japan. *Soroban. Useful arithmetical tool*. The League for Soroban Education of Japan, Inc., 1981

[9] López García, Juan Carlos. *Guía de referencia de Scratch 2.0*. Consultado en: http://eduteka.icesi.edu.co/pdfdir/ScratchGuiaReferencia.pdf

[10] Marji, Majed. *Learn to Program with Scratch*. No Starch Press, San Francisco, 1ª edición, 2014.

[11] Pullan, J.M. *The History of the Abacus*. Frederick A. Praeger Publishers, Nueva York, 1ª edición, 1969.

[12] Resnick, Mitchell et al. Scratch: Programming for all *Communications of the ACM*. 52 (11): pp. 60-67.

[13] Samoly, Kevin. The History of the Abacus *Ohio Journal of School Mathematics*. Ohio Council of Teachers of Mathematics, no. 65 (2012), pp. 58-66

[14] Young, E.A. *The Abacus: A History*. Consultado en: http://www.fenris.net/~lizyoung/abacus.html

[15] Zúñiga Morelli, Óscar *Aritmética en el ábaco japonés* Consultado en: http://www.librosmaravillosos.com/zumor/index.html el 19 de abril de 2018

[16] Texto sobre el uso de distintos ábacos por la Dirección de Educación Especial de México. *Los ábacos: Instrumentos didácticos*. Consultado en: http://educacionespecial.sepdf.gob.mx/escuela/documentos/publicaciones/LosAbacos.pdf

[17] Wiki oficial de Scratch Consultado en: https://wiki.scratch.mit.edu/wiki/Scratch_Wiki_Home el 23 de marzo de 2018

APÉNDICE A
Web del Museo de Informática

El Museo de Informática de la UPV realiza una ardua labor por la difusión del conocimiento informático, realizando numerosas actividades como visitas guiadas a las distintas colecciones de objetos, ciclos de cine relacionados con la historia de este campo, o incluso talleres de retroinformática o del propio lenguaje Scratch. También promueve iniciativas como las jornadas «Bits en femenino», que pretenden resaltar el importante pero a veces ignorado papel de las mujeres en la historia (y la actualidad) de la ciencia informática. Es posible encontrar información sobre todas estas iniciativas y muchas más, así como numerosos artículos, vídeos divulgativos y otros muchos materiales en la página web del Museo [1]. En la Figura A.1 puede verse la página principal del sitio del Museo con muchos de sus contenidos principales destacados.

Dentro de la web del Museo es posible acceder a numerosas páginas con interesante contenido relacionado con la historia de la informática y con proyectos Scratch embebidos. A lo largo de los años y gracias al respaldo de la ETSINF hacia el lenguaje Scratch como herramienta para acercar la programación a la población externa al mundo de la informática, y en especial al público infantil y juvenil, se han desarrollado numerosas aplicaciones de distinto tipo, pero normalmente relacionadas con la historia de la informática o del cálculo y las matemáticas. Con todas estas aplicaciones se ha creado el repositorio Scratch del Museo, donde es posible encontrar gran cantidad de proyectos de todo tipo: juegos «retro» implementados de nuevo usando Scratch, simuladores de antiguas máquinas calculadoras como las vistas en este trabajo, proyectos que hacen uso de la PicoBoard, etc.

Uno de los objetivos del presente trabajo consistía en realizar una aportación al Museo y, más en general, a todo el público interesado en la programación, la informática o la historia del cálculo en general. Por ello, se ha diseñado una página web en la que se incluye un pequeño resumen de todo lo explicado en la memoria acerca de la evolución histórica de las herramientas de cálculo y en concreto de los ábacos. También se explica de forma muy sucinta el funcionamiento de los ábacos chino, japonés y ruso, y se encuentran embebidos los tres proyectos Scratch desarrollados para que se puedan utilizar desde la propia página de la Universitat Politècnica sin tener que navegar a la de Scratch.

[1] Página web del Museo de Informática de la UPV: http://museo.inf.upv.es/es/, visitada el 7 de julio de 2018

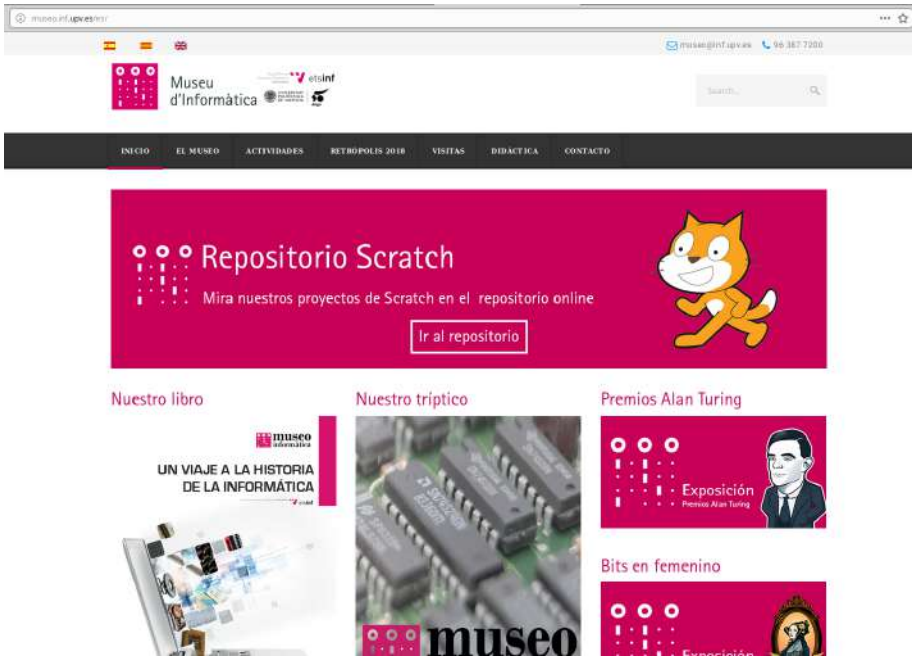

Figura A.1: Página principal en el sitio web del Museo de Informática de la UPV. En ella se pueden ver algunos de los más destacados proyectos del Museo: las jornadas «Bits en femenino», los premios Alan Turing, el libro editado por el propio Museo o el repositorio de proyectos Scratch.

El diseño de la página es muy sencillo, y se ha realizado usando *Hypertext Markup Language* (HTML), el lenguaje de marcado estándar en la web y que, junto con CSS y JavaScript, compone el esqueleto principal de los contenidos que encontramos al navegar por ella. El diseño realizado consiste en unos pocos apartados y subapartados, introducidos cada uno de ellos por un encabezado. En ellos se puede leer unos pocos párrafos sobre el tema tratado, así como ver algunas imágenes y utilizar los proyectos Scratch embebidos. Además, se han creado tres documentos HTML con el contenido en valenciano, castellano e inglés para poder adaptarla a los distintos idiomas soportados por la web del Museo. En la Figura A.2 puede estudiarse el código fuente de la página web desarrollada, mientras que en la Figura A.3 se observa la apariencia final de esta web tras añadirle estilos y al ser su código interpretado por un navegador.

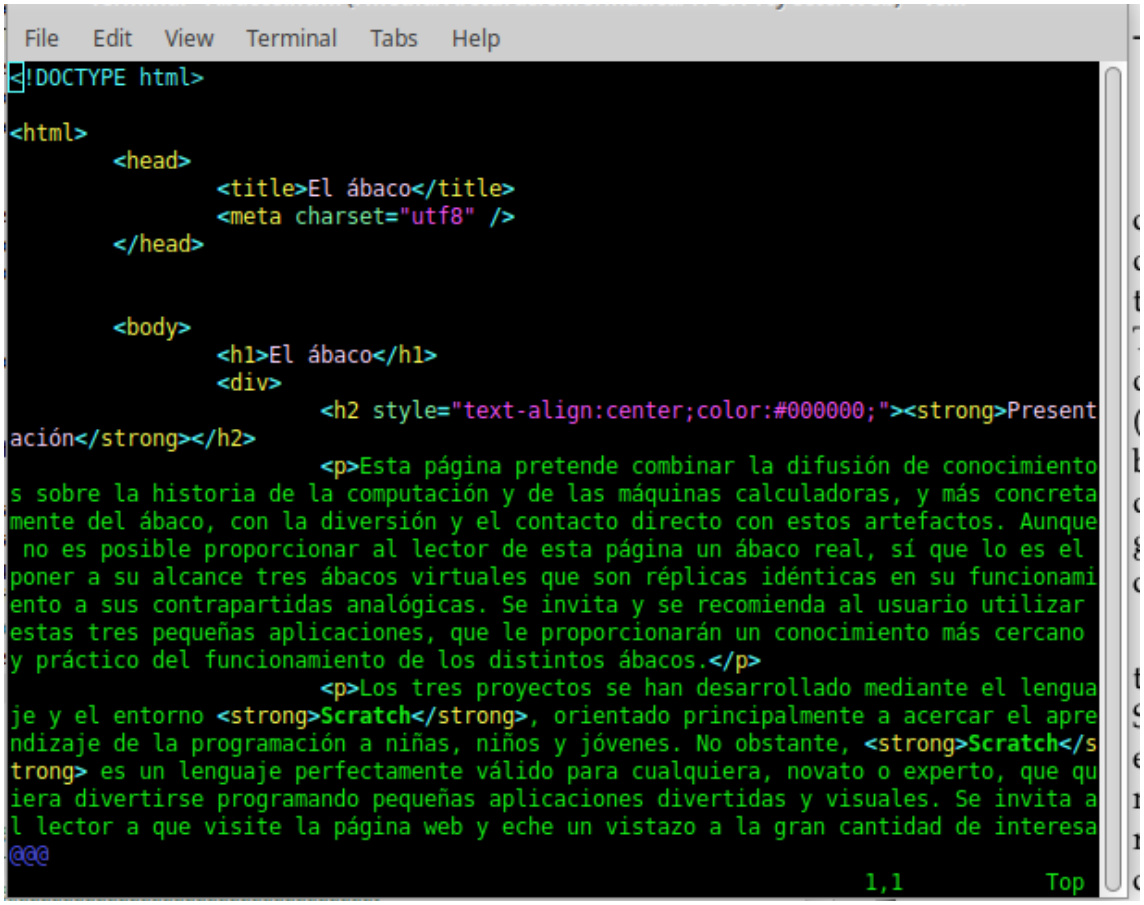

Figura A.2: Código HTML de la página web desarrollada sobre los ábacos. En este caso se ha utilizado el editor de textos Vim para escribir el código. En la imagen se observan los encabezados y los primeros párrafos de la página.

museo.inf.upv.es/es/el-abaco/

El ábaco chino o *suanpan*

Durante la Edad Media, concretamente hacia el siglo XII, se desarrolló en China un ábaco de piezas integradas desplazables, posiblemente el más antiguo que sobrevive aún en uso hasta la actualidad. Su mecanismo es muy similar al del ábaco romano, por lo que se piensa que podría haber sido creado inspirándose en aquel, tal vez a través de algún intercambio comercial entre los dos imperios.

Este ábaco se compone de dos partes, una superior y otra inferior, divididas por una madera central. Cada varilla tiene cinco cuentas en la parte inferior, con un valor de 1, y dos en la superior, con un valor asociado de 5. Estos valores base después se multiplican por el exponente de 10 asociado a esa columna (1, 10, 100...). Para añadir debemos desplazar las cuentas hacia la parte central del ábaco, es decir, subir las cuentas de la parte inferior o bajar las de la superior.

A continuación se presenta una pequeña aplicación Scratch para operar con el *suanpan*:

Figura A.3: Aspecto final de la página desarrollada al ser vista usando un navegador web (en este caso Firefox). Puede observarse en la parte inferior uno de los proyectos Scratch embebidos.