

UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR



Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO FIN DE GRADO

Interfaz Android para la herramienta Chalkpy

Rodrigo de Blas García
Tutor: Luis Fernando Lago Fernández

Julio 2016

Interfaz Android para la herramienta Chalkpy

AUTOR: Rodrigo de Blas García
TUTOR: Luis Fernando Lago Fernández

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2016

Resumen

Desde la aparición del álgebra computacional, la enseñanza ha utilizado las nuevas tecnologías para presentar contenido matemático de manera más intuitiva y dinámica que los métodos clásicos como la pizarra o el papel y bolígrafo. Bien es cierto que lo más extendido a día de hoy siguen siendo dichos métodos, pero se empieza a vislumbrar un cambio progresivo hacia la incorporación de dispositivos tecnológicos como apoyo al docente en materias de índole matemático, especialmente cuando se refiere al álgebra.

Este Trabajo de Fin de Grado da un paso más en esa dirección, proporcionando un nuevo modo de manipular expresiones matemáticas desde un dispositivo Android, ya sea Tablet o Smartphone. El producto desarrollado permite la aplicación de reducciones o equivalencias matemáticas a expresiones algebraicas de primer orden o lineales, de manera dinámica y sencilla.

La funcionalidad aportada incluye la aplicación de las propiedades matemáticas básicas como la conmutativa, asociativa, distributiva y factor común, además de realizar operaciones en expresiones con signos matemáticos y cambio de lado en igualdades algebraicas. Para permitir tales acciones se apoya en un motor algebraico que provee las equivalencias necesarias.

El proyecto ha sido desarrollado en Android bajo una arquitectura Modelo-Vista-Controlador. La capa Modelo contiene una interfaz de comunicación con el motor algebraico, así como las clases que lo implementan y los datos básicos de la aplicación: las expresiones matemáticas. La capa Vista se ha implementado como un componente visual de Android el cual permite el control de la selección de términos en una ecuación. Por su parte, la capa Controlador se encarga de actualizar los datos de la capa Modelo, transformando una expresión visualizada en la capa Vista mediante acciones de usuario.

La aplicación ha cumplido con éxito los objetivos marcados, permitiendo manipular fácilmente una expresión matemática mediante acciones de usuario que realizan las transformaciones arriba descritas.

Palabras clave

Android, aplicación, motor algebraico computacional, álgebra, tecnología educativa

Abstract

Teaching has been using new technologies since the discovering of Computer Algebra. This allowed to present mathematical workload in an easier, more dynamic way than current blackboards or paper and pen. Although these are the most extended methods nowadays, the use of tech devices is arising when it comes to Mathematics, specially Algebra.

This bachelor thesis provides a new point of view for mathematical expression manipulation, using an Android device (Tablet or Smartphone). The developed product allows rewriting rules to be applied to algebraic linear equations in an intuitive and natural way.

The provided functionality includes applying the Mathematics basic properties like commuting, associating, distributing or extracting a common factor, besides calculating results by operating terms, and moving them to the other side of a given equation. To achieve this, an external algebraic engine is used as a rewriting rules provider whenever is needed.

The project uses the Android technology, developing a Model-View-Controller Architecture to implement the project functionality. Model layer contains the algebraic engines structures and the basic data: mathematical expressions. View layer provides a way of controlling the user term selection by extending an Android widget. Finally, Controller layer updates the model data according to the actions the user performs in View layer.

This application success resides in the achievement of its objectives: manipulation of an algebraic expression so that the previous transforms can be applied at user's desire.

Keywords

Android, application, computer algebra system, algebra, educational technology

Agradecimientos

El documento que sostienes en tus manos es fruto de bastantes años en la que se ha convertido en mi casa. Noches sin dormir, días sin descansar... Todo ello me ha conducido al punto donde me encuentro ahora.

Sin embargo, nada habría sido posible si lo hubiera tenido que hacer yo solo. Detrás de mí quedan compañeros, profesores, asignaturas y, sobre todo, experiencias. Demasiadas para contarlas en esta humilde página. Así que, como buen ingeniero, al grano y resumiendo:

En primer lugar, gracias a mi tutor por darme la oportunidad de realizar este trabajo, y no rechazarlo cuando planteamos la idea.

Por supuesto, gracias a mi novia por aguantarme durante todo el tiempo que he estado incordiando con el TFG (y lo que no es el TFG). Si no fuera por ella me habría hundido hace muchísimo tiempo.

Gracias a mis padres por incordiarne con el tema estudios, si no lo hubieran hecho, probablemente habría hecho menos; gracias a mi hermana por estar ahí cuando la necesité.

Gracias a todos los compañeros de la carrera (si no digo nombres es porque no acabaría nunca) que me han ayudado a lo largo de ella, ya que sin ellos no habría aprobado más de alguna asignatura. De hecho, gracias a ellos pude desconectar de los estudios cada vez que me quemaba. Por muchas salidas más y muchas más vivencias.

Por supuesto, mencionar también a mis compañeros/as de la Biblioteca Politécnica de la EPS. Risas, llantos y muchas quejas soportadas.

Mención especial para los compañeros de juegos nocturnos en el ordenador, echando horas de juego, risas y frustraciones hasta las tantas de la madrugada.

Y cómo olvidarme de Diego y Lorenzo, eternos proveedores de ánimos desde detrás de la barra de la cafetería.

Y, por último, pero no menos importante: gracias a la comunidad de Internet: StackOverflow, Wikipedia, Yahoo Respuestas... por esa fuente inagotable de sabiduría y que me saco de algunos aprietos importantes.

Creo que no me dejo a nadie.... ¡Ah, sí! Gracias a ti, por tomarte el tiempo y la molestia de leerte este trabajo, ¡espero que no te resulte demasiado tedioso!

INDICE DE CONTENIDOS

1	Introducción.....	1
1.1	Motivación.....	1
1.2	Objetivos.....	2
1.3	Organización de la memoria.....	3
2	Estado del arte	5
2.1	Introducción.....	5
2.2	Álgebra Computacional y manipulación de expresiones	5
2.3	Dispositivos móviles y aplicaciones matemáticas.....	6
2.4	Conclusiones.....	7
3	Definición del Proyecto	9
3.1	Alcance	9
3.2	Metodología.....	9
3.3	Tecnologías y herramientas utilizadas.....	10
3.3.1	Tecnologías.....	10
3.3.2	Herramientas.....	10
4	Análisis	13
4.1	Requisitos funcionales.....	13
4.2	Requisitos no funcionales.....	14
5	Diseño.....	15
5.1	Arquitectura de la aplicación.....	15
5.1.1	Modelo.....	15
5.1.2	Controlador.....	16
5.1.3	Vista.....	17
5.2	Maquetas.....	17
6	Desarrollo	21
6.1	Estructura del proyecto.....	21
6.1.1	Fichero <i>AndroidManifest.xml</i>	21
6.1.2	Código Java	21
6.1.3	Recursos	22
6.2	Modelo/Model	22
6.2.1	Motor Algebraico Externo.....	22
6.2.2	Historial de expresiones.....	25
6.3	Controlador/Controller	25
6.3.1	Navegación	26
6.3.2	Autogestión de pantallas.....	27
6.3.3	Acciones y selección del usuario.....	30
6.4	Vista/View	31
6.4.1	<i>DrawableExpression</i> , clases de apoyo	31
6.4.2	<i>ExpressionView</i> , <i>SelectionGestureListener</i> : Selección.....	32
6.5	Utilidades.....	33
6.5.1	<i>CASUtils</i>	34
6.5.2	<i>PreferenceUtils</i>	34
6.5.3	<i>Utils</i>	34
7	Pruebas y resultados	35
7.1	Pruebas internas.....	35
7.1.1	Inspección, funcionamiento y refactorización.....	35
7.1.2	Caja negra: componentes.....	35
7.2	Pruebas de interfaz.....	36
7.3	Resultados.....	37

8 Conclusiones y trabajo futuro.....	39
8.1 Conclusiones.....	39
8.2 Trabajo futuro y posibles mejoras	39
Referencias	41
Glosario	43
Anexos	I
Anexo A. Manual de uso.....	I
Anexo B. Diagramas de clases.....	XIII
B.1 Modelo.....	XIII
B.2 Vista.....	XIV
B.3 Controlador.....	XV
Anexo C. Fichero de configuración	XVII
Anexo D. Funciones de CASAdapter	XIX
Anexo E. DrawableExpression y clases de apoyo.....	XXI
E.1 DrawableExpression	XXI
E.2 DrawableSingleExpression.....	XXI
E.3 DrawableExpressionList.....	XXI
E.4 DrawableOperator, DrawableParenthesis.....	XXII

INDICE DE FIGURAS

FIGURA 5.1: PATRÓN MODELO-VISTA-CONTROLADOR.....	15
FIGURA 5.2: NAVEGACIÓN EN LA APLICACIÓN	17
FIGURA 5.3: PANTALLA <i>TABLERO/BOARD</i>	18
FIGURA 5.4: PANTALLA <i>HISTORIAL/HISTORY</i>	19
FIGURA 5.5: PANTALLA <i>AJUSTES/SETTINGS</i>	19
FIGURA 6.1: NAVEGACIÓN POR MENÚ.....	26
FIGURA 6.2: NAVEGACIÓN POR ACCIÓN.....	27
FIGURA 6.3: FUNCIONAMIENTO DE <i>EXPRESSIONFRAGMENT</i>	28
FIGURA 6.4: REPRESENTACIÓN VISUAL DE <i>DRAWABLEEXPRESSIONLIST</i>	31
FIGURA 7.1: PANTALLA <i>TABLERO</i> EN LA APLICACIÓN FINAL	37
FIGURA 7.2: PANTALLA <i>HISTORIAL DE EXPRESIONES</i> EN LA APLICACIÓN FINAL	38
FIGURA A.1: LANZAMIENTO DE LA APLICACIÓN	I
FIGURA A.2: PANTALLA PRINCIPAL	II
FIGURA A.3: PANTALLA DE AYUDA	II
FIGURA A.4: MENÚ DESPLEGABLE DE NAVEGACIÓN	III
FIGURA A.5: AJUSTES DE LA APLICACIÓN.....	III
FIGURA A.6: AJUSTES DE COLOR DE LA SELECCIÓN DE USUARIO	IV
FIGURA A.7: EXPRESIONES DE EJEMPLO PREDEFINIDAS	V
FIGURA A.8: EXPRESIÓN SELECCIONADA PARA SU MANIPULACIÓN	V
FIGURA A.9: SELECCIÓN MÚLTIPLE DE TÉRMINOS PARA PROPIEDAD DISTRIBUTIVA	VI
FIGURA A.10:EXPRESIÓN RESULTANTE DE APLICAR LA PROP. DISTRIBUTIVA	VII
FIGURA A.11: SELECCIÓN SIMPLE DE UNA EXPRESIÓN PARA OPERAR.....	VII
FIGURA A.12: SELECCIÓN PARA CAMBIO DE LADO EN UNA ECUACIÓN	VIII
FIGURA A.13: RESULTADO DEL CAMBIO DE LADO EN UNA EXPRESIÓN	VIII
FIGURA A.14: SELECCIÓN PARA APLICAR LA PROP. CONMUTATIVA.....	IX

FIGURA A.15: RESULTADO DE APLICAR LA PROP. CONMUTATIVA A UNA EXPRESIÓN.....	X
FIGURA A.16: SOLUCIÓN DE LA ECUACIÓN.....	X
FIGURA A.17: HISTORIAL DE EXPRESIONES (I)	XI
FIGURA A.18: HISTORIAL DE EXPRESIONES (II)	XII
FIGURA A.19: PANTALLA <i>TABlero</i> CON COLOR DE FONDO OSCURO	XII
FIGURA B.1: DIAGRAMA DE CLASES DE LA CAPA MODELO	XIII
FIGURA B.2: DIAGRAMA DE CLASES DE LA CAPA VISTA	XIV
FIGURA B.3: DIAGRAMA DE CLASES DE LA CAPA CONTROLADOR	XV

INDICE DE TABLAS

TABLA 6.1: PROPIEDADES Y REDUCCIONES PARA EL FACTOR COMÚN	24
TABLA 6.2: FUNCIONES DE <i>CASADAPTER</i> Y PROPIEDAD QUE IMPLEMENTAN	25
TABLA 6.3: PANTALLAS Y FRAGMENTOS ASOCIADOS	27
TABLA 7.1: RESULTADOS DE PRUEBAS DE INTERFAZ.....	36

1 Introducción

Este Trabajo de Fin de Grado, realizado en la Escuela Politécnica Superior de la Universidad Autónoma de Madrid, tiene como propósito el desarrollo de una aplicación para dispositivos móviles Android que sirva como interfaz para un motor algebraico de manipulación de expresiones simbólicas.

La aplicación puede considerarse una nueva versión de la herramienta *Chalkpy* [1], usando como motor algebraico interno el obtenido gracias a Laura Salcedo Valderrama en su Trabajo de Fin de Grado *Desarrollo de un Motor de Matemática Simbólica para la Herramienta Chalkpy* [2]. Ésta herramienta a partir de ahora será referenciada como *G-CAS*.

A continuación, se explica la motivación y los objetivos principales del mismo, seguidos de una descripción de la estructura del presente documento.

1.1 Motivación

Es común a la hora de dar clase el utilizar una herramienta de apoyo que facilite el manejo de expresiones simbólicas, sobre todo en asignaturas de carga matemática. Normalmente se utiliza la pizarra como medio para tal fin.

Esta herramienta provee un método activo de explicación de problemas, permitiendo al docente ilustrar al momento los pasos seguidos en la resolución de los mismos. El más claro ejemplo se tiene con las ecuaciones matemáticas, donde se realizan unas manipulaciones sobre una misma expresión, cambiando su forma a medida que se va resolviendo.

Sin embargo, a veces es fácil malinterpretar un símbolo en la pizarra, ya sea por la letra del docente o por errores a la hora de realizar un paso: un signo incorrecto, una constante olvidada, etc. Estos errores llevan a una resolución incorrecta del problema, y por consiguiente, un resultado erróneo. Incluso muchas veces no se es consciente del error cometido, lo que puede ocasionar pérdidas de tiempo importantes en detectar y corregir el error.

Por otro lado, al no poder utilizarse la misma expresión, sino tener que ir copiándola paso tras paso, puede desembocar en una pizarra caótica, sin una ordenación clara que indique el proceso seguido.

Además, muchas veces surgen dudas en tutorías “no programadas”, lo cual quiere decir que no se tiene a mano una herramienta similar (por ejemplo papeles u ordenador) para resolver los problemas que surjan en ese momento.

La herramienta *Chalkpy* [1] surgió a raíz de esto, proveyendo una nueva forma de presentar los contenidos matemáticos con la ayuda de un navegador web, y permitiendo su manipulación de manera sencilla y dinámica.

Sin embargo, se puede considerar cerrada ateniéndose a su enfoque (por ejemplo, sólo puede usarse en navegadores), por lo que usando uno nuevo, puede ampliarse su funcionalidad en otros dispositivos.

Por otro lado, el trabajo de Laura Salcedo [2] entra en acción como un generador de motores algebraicos versátil y manejable, ya que permite obtener un motor de manipulaciones algebraicas mediante la declaración de un fichero de propiedades que contenga las equivalencias deseadas.

No obstante, no provee una interfaz gráfica para su uso, únicamente una interfaz de consola. De este modo aparece la oportunidad de dotar de una a la funcionalidad final del citado proyecto.

Debido a todo ello surge la idea central de este Trabajo de Fin de Grado: dotar al docente de una aplicación contenida en un dispositivo móvil (ya sea Tablet o Smartphone) que permita la fácil manipulación de expresiones simbólicas matemáticas, y más específicamente, de ecuaciones lineales. En cierto modo esta aplicación puede considerarse como una “nueva versión” o un nuevo enfoque de *Chalkpy*, trabajando sobre su idea principal y ampliando horizontes.

1.2 Objetivos

Este Trabajo de Fin de Grado tiene como objetivo el diseño y desarrollo de una aplicación Android para dispositivos móviles (Smartphone y Tablet) que permita la manipulación en tiempo real de expresiones matemáticas de primer orden. Esto es, ecuaciones lineales en las que aparece una única incógnita.

Para ello se usará el motor algebraico obtenido como resultado de aplicar a un fichero de configuración la herramienta *G-CAS*. Por tanto, es necesaria la creación del mencionado fichero para garantizar los objetivos abajo listados.

Este motor algebraico se conoce como CAS (Sistema Algebraico Computacional). Consiste en un conjunto de reglas de equivalencia o reducciones que permiten la manipulación algebraica de expresiones matemáticas, de modo que se puede convertir una en otra al aplicar dichas reducciones.

De este modo la principal función de la aplicación es servir de intermediario entre el usuario y el motor interno, utilizando las órdenes del usuario sobre la representación visual de una expresión matemática para transmitir las a dicho motor. Una vez transmitidas y habiendo obtenido respuesta del CAS, se ocupará de representar dichos cambios en la interfaz, para que el usuario tenga noción de que ha sido él quien ha ocasionado dichos cambios en la expresión.

Esta función se puede considerar exitosa siempre que se cumplan unas ciertas condiciones necesarias, a saber:

- ❖ Las expresiones usadas en la aplicación son de primer orden (véase Alcance, sección 3.1).
- ❖ Las expresiones son manipuladas mediante el uso de acciones simples que vendrán dadas por botones activos o no en función de la selección en ese momento.
- ❖ Las manipulaciones son guardadas en un histórico de expresiones, permitiendo deshacer una acción y volver a la expresión anterior.

1.3 Organización de la memoria

Este documento se compone de las siguientes secciones:

- **Estado del arte**

Aquí se revisa la situación actual de este tipo de aplicaciones con el fin de poner al lector en situación respecto a la extensión de uso de las mismas. Así mismo se exploran algunas aproximaciones similares al problema planteado.

- **Definición del proyecto**

En esta sección se especifica el alcance del proyecto, se explica la metodología seguida para el planteamiento y desarrollo del mismo y se describen las herramientas utilizadas para tal fin.

- **Análisis**

Este apartado detalla los requisitos de la aplicación, tanto funcionales como no funcionales, que darán el comportamiento y apariencia de la misma.

- **Diseño**

El capítulo 5 versa sobre cómo se ha diseñado el producto final, su arquitectura y componentes.

- **Desarrollo**

El contenido de esta sección trata sobre el proceso de implementación de la aplicación, entrando en detalle en la funcionalidad ofrecida, y cómo se ha conseguido.

- **Pruebas y resultados**

Las pruebas realizadas durante el desarrollo de la aplicación, así como las posteriores a su implementación, se encuentran definidas y explicadas en esta sección. Además, se incluyen los resultados obtenidos.

- **Conclusiones y trabajo futuro**

En el último capítulo se encuentran las conclusiones obtenidas al finalizar el proyecto y las posibles tareas a realizar en un futuro a raíz de los resultados obtenidos.

2 Estado del arte

Desde la aparición del ordenador, y su posterior uso en la educación, los docentes han intentado adaptar dicha tecnología a sus explicaciones en las aulas. Gracias a ello, y junto a las herramientas que iban apareciendo, se facilitó la explicación de los contenidos, sobre todo en lo referente a matemáticas. De esta forma, los profesores de hoy día son capaces de exponer sus ideas de manera más clara y visual a un alumnado cada día más familiarizado con las tecnologías de la información.

En esta sección se repasan las herramientas utilizadas en la docencia para la presentación de contenidos matemáticos y de apoyo a la hora de tratar con ellos. Se hace especial hincapié en aquellas variantes que soportan dispositivos móviles, y más específicamente, para la plataforma Android.

2.1 Introducción

A lo largo de los años se ha intentado exponer contenido matemático en las aulas de una manera lo más visual posible, es decir, intuitiva. Para ello, los docentes se apoyaban en las pizarras, dibujos que explicaban el concepto, e incluso juegos con elementos externos (por ejemplo al aprender a sumar). Sin embargo, de nada sirve todo eso si un alumno no entiende la idea subyacente de lo que se está explicando.

De este modo, gracias a la inestimable ayuda del ordenador, y posteriormente, de los sistemas algebraicos computacionales (a partir de ahora, CAS: Computer Algebra System), los profesores han sido capaces de expresar con mayor claridad los contenidos matemáticos de sus asignaturas, permitiendo a sus alumnos comprender mejor los conceptos que se les planteaban.

Sin embargo, sigue existiendo una cierta deficiencia a la hora de manipular expresiones en tiempo real. Una pizarra (tanto de tiza como de rotuladores) no permite una manipulación dinámica de las expresiones matemáticas planteadas, ni trabajar con sus términos de manera intuitiva. Se entiende por manipulación dinámica e intuitiva aquella que es “más natural” para un ser humano. Es decir, poder manipular un elemento sin tener que replicarlo para variar el contenido, y de manera fácil y visual. Al mismo tiempo, una pizarra es más sensible a fallos por parte del docente: signos cambiados, variables perdidas, etc.

Aun siendo cierto que encontramos pizarras digitales en las que se puede trabajar con los dedos, el ámbito se reduce a las aulas, cuando lo ideal sería poder trabajar de esa manera en cualquier lado. Es ahí donde entran en juego los dispositivos móviles y las aplicaciones que permiten trabajar con expresiones matemáticas, ya que una pizarra digital no es una aplicación para un dispositivo de estas características.

2.2 Álgebra Computacional y manipulación de expresiones

Sin un motor algebraico que permita realizar las manipulaciones necesarias sobre las expresiones no existirían las aplicaciones con tal fin que hoy en día conocemos.

Desde sus inicios con las ideas de Leibniz y Babbage, y pasando por los genios del S.XX como Hilbert, Gödel y Turing, el álgebra computacional estuvo enfocada a la manipulación

automática de expresiones. Estas ideas se vieron cumplidas con la aparición del primer CAS en 1963 por Martinus Veltman (*Schoonschip*) [3].

Así pues, para una herramienta de manipulación de expresiones es necesaria la aportación de uno de estos motores algebraicos. El CAS permite, dada una expresión, realizar una serie de acciones incluidas en él, o, mediante reducciones, pasar de esa expresión a otra equivalente. Es decir, un CAS simplemente es un conjunto de reglas de reescritura.

Estas reglas no son más que las que realiza un humano de manera normal al manipular contenidos matemáticos: conmutar términos dentro de una suma, pasar a un lado de la ecuación, simplificar un cero, etc. La idea del CAS es realizar estas manipulaciones de manera automática, aunque se pueden ir aplicando estas reducciones mediante la interacción del usuario. Basta con saber qué reducción se quiere aplicar, la expresión dada y los términos involucrados.

2.3 Dispositivos móviles y aplicaciones matemáticas

Recientemente se desarrolló una aplicación *online*, *Chalpy* [1], que permitía, mediante el uso de un navegador, presentar contenidos matemáticos de manera más sencilla y dinámica para un docente.

Con ella se pueden manipular expresiones simbólicas haciendo uso de botones que representan acciones a realizar sobre las mismas. Sin embargo, no se dispone de una aplicación móvil para dicha herramienta.

Existe una gran variedad de aplicaciones para móviles que tratan el tema de expresiones matemáticas, como bien se comenta en el sitio web *matematicascercanas* [4]. No obstante, la gran mayoría de éstas tienen un enfoque más calculístico, es decir, orientadas al cálculo de una solución en vez de aprovechar la potencia de un CAS y trabajar de manera simbólica.

Cierto es que alguna herramienta ofrece resolución paso a paso y es capaz de manejar expresiones simbólicas (el mejor ejemplo de esta clase es el conocido CAS de la herramienta *Wolfram Alpha* [5]).

Algunas de las aplicaciones para móviles que más se acercan a la solución buscada en este Trabajo de Fin de Grado son las siguientes:

PhotoMath

Photomath [6] es una aplicación Android consistente en una calculadora con cámara. Se apunta con la cámara del dispositivo móvil a una operación matemática y la aplicación mostrará el resultado instantáneamente, con la opción de mostrar los pasos seguidos en su resolución. No obstante, no cumple el requisito fundamental de la aplicación, la manipulación de expresiones por parte del usuario.

Wolfram Alpha

La aplicación móvil de *Wolfram Alpha* [7] es un CAS muy potente que permite la resolución paso a paso de problemas matemáticos y multitud de reescritura de expresiones, permitiendo pasar de una a otra de manera instantánea. De nuevo, no permite la manipulación de las mismas por el usuario, con el añadido de que hay que pagar por usarla (la versión web es gratuita, pero no ofrece toda la funcionalidad).

Mathination

Mathination [8] es una aplicación móvil para la plataforma iOS de Apple que permite realizar de forma similar la funcionalidad buscada en este Trabajo de Fin de Grado. El usuario es capaz de mover términos dentro de la expresión, y operar con ellos mediante pellizcos en la pantalla. Sin embargo, tiene asociado un coste para su uso.

Algebra Touch

Algebra Touch [9] es una aplicación móvil para todos los sistemas operativos (Android, iOS, Windows Phone) que tiene una funcionalidad muy similar a *Mathination*. Al igual que la herramienta anteriormente descrita, cumple de manera parecida la funcionalidad básica buscada: manipulación de términos de una expresión matemática. Además tiene el aliciente de tener un módulo para explicar la resolución de ejercicios, una base de datos con problemas predefinidos y la posibilidad de guardar ejercicios personalizados. Su gran inconveniente es que, de nuevo, tiene un coste asociado.

2.4 Conclusiones

Pese a que ya existen herramientas que logran una funcionalidad similar, o bien no están para la plataforma Android o bien son de pago, lo cual supone una limitación en la distribución de esta tecnología para su aplicación en las aulas. Además, el número de aplicaciones existentes es muy reducido.

Con la aplicación desarrollada en este TFG se intentará dar una primera aproximación de un sistema gratuito y para Android de este tipo de herramientas, permitiendo una fácil ampliación de funcionalidades y extensión de usuarios.

3 Definición del Proyecto

En esta parte del documento se dan respuesta a las preguntas relativas al alcance de la aplicación, la metodología aplicada para el desarrollo y las tecnologías y herramientas usadas en ella.

3.1 Alcance

Dado que este Trabajo de Fin de Grado consiste en la aplicación Android que sirva como interfaz, no entra dentro de su ámbito el implementar la lógica que trabaje internamente con las expresiones utilizadas, ni garantizar que el usuario siga unos pasos establecidos. No obstante, la aplicación debe hacer una asociación entre las acciones del usuario y las reducciones del CAS interno.

Sí entra dentro de su alcance el proporcionar una visualización simple y clara de las expresiones utilizadas por el CAS interno, así como representar el cambio en las expresiones que el usuario ha ordenado al motor. Al mismo tiempo, debe proporcionar un fichero de configuración para obtener el motor adecuado mediante el uso de *G-CAS*.

3.2 Metodología

Una vez definido el proyecto, y establecido el alcance del mismo, se planteó el problema de cómo desarrollar la aplicación.

Dado que esta aplicación depende bastante del usuario, se ha optado por un desarrollo ágil, que permitiera ofrecer funcionalidad a medida que se desarrollaba el producto. En palabras de Ian Sommerville [10]:

“Los métodos ágiles se apoyan universalmente en el enfoque incremental para la especificación, el desarrollo y la entrega del software. Son más adecuados para el diseño de aplicaciones en que los requerimientos del sistema cambian, por lo general, rápidamente durante el proceso de desarrollo.”

De esta forma, se decidió ir desarrollando funcionalidades de manera acorde con lo que se describió en una especificación informal de requisitos de software. A medida que la funcionalidad clave estaba más clara, se iban modificando los requisitos de la aplicación, con el consiguiente cambio en la implementación.

Así pues, este método permitió ir creando funcionalidades a raíz de otras, utilizando un enfoque incremental por iteraciones, pero a su vez al mismo nivel (muchas funcionalidades en paralelo que crean otra más grande). Una iteración se compondría de los siguientes pasos:

0. Requisitos iniciales de la aplicación (sólo una vez en todo el proyecto).
1. Decisión de la funcionalidad a implementar.
2. Análisis y diseño de la misma.
3. Desarrollo del código que cumple la funcionalidad deseada.
4. Prueba de aceptación de la característica implementada.
5. Refactorización de código.

Al mismo tiempo se ha utilizado la técnica de la *refactorización de código*: una vez terminada la funcionalidad deseada y probada, se reordena y “limpia” el código de la misma para una mejor integración con las funcionalidades futuras. Esto permite tener un código simple y legible que es más fácil de reutilizar y mantener.

3.3 Tecnologías y herramientas utilizadas

Una vez definido el proyecto y establecidos los pasos para desarrollarlo, es momento de decidir las tecnologías con las cuales se va a llevar a cabo.

3.3.1 Tecnologías

Dado que el proyecto es una aplicación Android, las tecnologías asociadas a este tipo de proyectos son las siguientes:

- *Android* [11] como base de trabajo. Es un sistema operativo para dispositivos móviles que se basa en Linux. Para desarrollar una aplicación para este sistema operativo se necesita:
 - *Java* [12] como lenguaje de programación.
 - *XML* [13] como lenguaje de especificación de *layouts* (interfaces gráficas) y de recursos (*resources* en la terminología Android).
- *LaTeX* [14] como fuente de texto para la representación de las expresiones.

3.3.2 Herramientas

Entorno

Se ha elegido el Android Studio [15] como entorno de desarrollo. Es el IDE oficial para programar en Android, sustituyendo a Eclipse como tal. Contiene todo lo necesario para trabajar en esta plataforma, no siendo necesaria ninguna otra herramienta para esta tarea.

Bibliotecas Externas

Se ha utilizado el CAS obtenido tras la aplicación sobre un fichero de configuración de la herramienta *G-CAS* como motor algebraico interno para la manipulación de las expresiones a tratar. Utilizando una interfaz de comunicación (que debe ser creada), la aplicación llamará a las reducciones necesarias para realizar la acción que el usuario decida.

Dichas reducciones son generadas mediante la lectura de un fichero de configuración que contiene las definiciones y equivalencias deseadas. Acto seguido, la herramienta *G-CAS* generará un motor algebraico cuyas reducciones se han asociado a las definidas en el fichero de configuración. De este modo se dispone de una librería externa con las reducciones del CAS, pudiendo crear la interfaz buscada para realizar la conexión de los componentes.

Control de Versiones

Un control de versiones permite gestionar los cambios realizados durante el desarrollo de un proyecto. Para ello se han utilizado dos herramientas complementarias:

- *GitHub* [16]: Es una plataforma web que está basada en el software Git que permite almacenar los archivos de un proyecto en la nube, y gestionar el sistema de

cambios de los mismos. Aunque se encuentra integrada en Android Studio, se ha utilizado otra herramienta adicional (siguiente elemento de esta lista).

- *SourceTree* [17]: Aplicación de escritorio que provee una interfaz gráfica simple y fácil de usar para Git. No tiene cliente para Linux, pero sí para Windows y MacOS.
 - *SmartGit* [18]: Cliente de Git para Linux que proporciona una interfaz gráfica similar a la ofrecida por SourceTree. Usada cuando se desarrollaba en entorno Linux. También tiene cliente para Windows y MacOS.

Pruebas

Durante el proceso de validación (ver apartado de Pruebas, sección 7.2) se utilizaron varios dispositivos móviles para probar el correcto funcionamiento de la aplicación:

- *Jiayu S3 Plus* (Android 5.1): En él es donde se han realizado la mayoría de pruebas de la aplicación debido a que es una versión reciente de Android y tiene un tamaño de pantalla “razonable”.
- *UMI X2 Turbo* (Android 4.2): En él se realizaron pruebas de compatibilidad para garantizar el funcionamiento en versiones anteriores de Android, así como en dispositivos con un tamaño de pantalla menor.
- *Acer Iconia One 7* (Android 5.0): Utilizada para pruebas en dispositivos con mayor pantalla, de forma que se pueda garantizar el correcto funcionamiento en éstos.

Gráficos

Para la realización de los diagramas, maquetas y demás componentes visuales presentes en este documento se han utilizado las siguientes aplicaciones.

- *draw.io* [19]: aplicación web para la creación de los diagramas UML.
- *Balsamiq Mockups* [20] para la creación de las maquetas de la aplicación. Se eligió esta herramienta ya que permitía visualizar el diseño final de la interfaz y al mismo tiempo avanzar en el desarrollo de la misma, al tener parte codificada ya.
- *Gimp* [21]: aplicación de escritorio para edición de imágenes en general.

4 Análisis

En este capítulo se da una visión de las características que definen la aplicación, es decir, un análisis de lo que se le pide y cómo queda recogido.

Dada la naturaleza del proyecto, se definen en un primer lugar unos requisitos “base” sobre los que más adelante se fundan el resto de funcionalidades y la manera de implementarlas. En esta sección se recoge la especificación final, tras todo el proceso de cambio. De esta forma queda definida toda la funcionalidad de la aplicación.

4.1 Requisitos funcionales

Los requisitos claves o *básicos* funcionales (aquellos que recogen la funcionalidad de la aplicación) quedan recogidos en la siguiente lista:

- RF 1.** La aplicación mostrará una única expresión para manipular. La expresión será una ecuación lineal de primer orden, esto es, con términos de grado uno (sin potencias).
- RF 2.** Las acciones a realizar sobre la misma incluirán:
 - a. Selección simple de términos.
 - b. Selección múltiple de términos.
 - c. Propiedad conmutativa izquierda y derecha (desplazar el término hacia la izquierda y derecha, respectivamente).
 - d. Propiedad asociativa (crear paréntesis alrededor de un término).
 - e. Propiedad disociativa (deshacer los paréntesis que rodean al término).
 - f. Operar términos (ver RF3).
 - g. Cambiar un término de lado en la ecuación.
- RF 3.** La acción operar puede referirse a uno de los siguientes casos:
 - a. Operar términos mediante el operador inmediato superior (por ejemplo sumar los términos de una suma).
 - b. Operar un término con una expresión suma mediante un operador producto (propiedad distributiva).
 - c. Extraer un factor común de una serie de términos en una misma expresión.
- RF 4.** Debe detectarse qué acciones pueden llevarse a cabo en el estado actual de la expresión.
- RF 5.** En caso de realizarse una acción incorrecta, la expresión no cambiará, avisando al usuario de la situación mediante un mensaje en la pantalla.
- RF 6.** Se dispondrá de un histórico de expresiones que permita ver la evolución de la actual, pudiendo volver a una de ellas en cualquier momento. En este caso, la expresión actual será cambiada por ésta y descartada.
- RF 7.** Se dispondrá de unas “expresiones de prueba” que permitan seleccionar una expresión ya definida.

RF 8. Se podrá deshacer una acción sobre una expresión, volviendo a la anterior, hasta un máximo permitido por el historial.

RF 9. No se podrá resolver la ecuación de manera automática; quedan a cargo del usuario todas las manipulaciones sobre ella.

4.2 Requisitos no funcionales

A su vez, se definieron unos requisitos básicos no funcionales para modelizar la apariencia de la interfaz, así como su usabilidad. Con el tiempo, los requisitos fueron modificados, hasta desembocar en los aquí definidos:

RNF 1. La expresión mostrada sólo podrá manipularse mediante el uso de botones que realicen las acciones indicadas en el RFB2.

RNF 2. Las acciones que puedan realizarse se mostrarán como un botón activado, mientras que las que no se puedan realizar deberán aparecer como dicho botón desactivado.

RNF 3. Los botones de acción deben poder mostrar la descripción de la acción.

RNF 4. La aplicación podrá ser usada desde cualquier Smartphone o Tablet Android, con una versión superior a la API 16 (Android 4.1 JellyBean), aunque está pensada para un dispositivo con API 22 (Android 5.1 Lollipop).

RNF 5. La aplicación permitirá la selección de los términos de la expresión mediante pulsaciones en la pantalla (*taps*) para la selección única, y mediante una pulsación larga para la selección múltiple de los mismos.

RNF 6. La selección que el usuario haga debe mostrarse en un color que resalte sobre el resto de la expresión y sobre el fondo. El usuario puede elegir el color de la misma dentro de un rango de colores definidos por la aplicación (actualmente tres: anaranjado, azul claro y rojo).

RNF 7. La aplicación mostrará una pantalla de ayuda que explique el funcionamiento de la pantalla de manipulación (de aquí en adelante *Board* o *Tablero*).

RNF 8. La aplicación podrá adaptar el color del tablero de expresiones acorde con los deseos del usuario. Éste podrá elegir un tablero claro (blanco) u oscuro (verde pizarra), para adecuar la vista a uno u otro.

RNF 9. La interfaz gráfica permitirá adaptarse al idioma del teléfono. Esto indica que la aplicación detectará el idioma del teléfono, usando inglés o español según corresponda.

RNF 10. Las expresiones se representarán usando la fuente de texto *Latin Modern Roman*, utilizada por LaTeX, dada su mejor apariencia en este tipo de expresiones.

RNF 11. Se dispondrá de un Manual de Usuario (Anexo A).

5 Diseño

En este capítulo se detallan las decisiones de diseño que se han tomado para la creación de la aplicación Android cuya especificación se ha visto en la sección 4. Para ello se habla del patrón general elegido, de los componentes que lo forman y de la manera en la que se comunican entre ellos.

5.1 Arquitectura de la aplicación

Android proporciona un *framework* que permite de manera muy sencilla implementar un modelo de arquitectura Modelo-Vista-Controlador (*Model-View-Controller*, o MVC) [22]. Este patrón se caracteriza por separar todos los componentes en tres capas:

- *Modelo*: encargada de manejar los datos básicos (por ejemplo las entidades de una base de datos o las estructuras que modelizan las expresiones matemáticas utilizadas en este proyecto) de una aplicación. En el caso de Android esto corresponde a los datos que manejan las aplicaciones.
- *Vista*: capa cuya tarea consiste en proveer una visualización de los datos de la capa de Modelo. Para una aplicación Android, la capa Vista la componen las definiciones de interfaz gráficas (*layouts*) en ficheros XML, y las clases Java que heredan de la clase *View* de Android (más sobre esto en la sección 6.4).
- *Controlador*: este nivel se ocupa de, como su nombre indica, controlar y manejar la lógica de la aplicación, poniendo en contacto los datos de la capa de modelo con la de vista. Para Android, esta capa viene representada (principalmente) por las clases *Activity* y *Fragment*, así como por los demás componentes que permiten el control interno de la aplicación.

La figura 5.1 representa de manera gráfica esta arquitectura y la interacción entre sus capas:

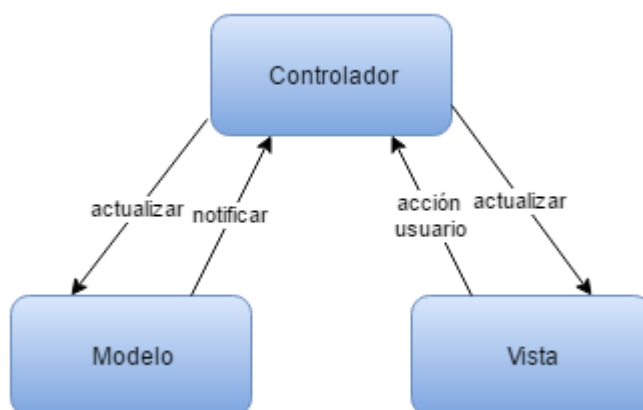


Figura 5.1: Patrón Modelo-Vista-Controlador

5.1.1 Modelo

Como se ha mencionado en la sección anterior, en una aplicación Android la capa de modelo contiene los datos que se manejan a lo largo de su ciclo de vida.

En este caso los datos utilizados no son más que las clases que modelizan las expresiones matemáticas a utilizar en la aplicación y los componentes que las manipulan internamente, sin interferir en el flujo de procesos de la aplicación.

Este proyecto utiliza el CAS obtenido como aplicación de *G-CAS* a un fichero de configuración como base para la manipulación de las mismas, pasando las acciones a ejecutar desde la capa Controlador (ver secciones 6.2 y 6.3).

De esta forma, es necesario un adaptador que conecte el componente de control de la aplicación con el CAS externo, permitiendo enviar y recibir los datos que sean necesarios para el correcto funcionamiento de ambas partes.

5.1.2 Controlador

En una aplicación Android, quien hace el papel de controlador y se encarga de manejar la lógica interna de la misma son las Actividades, que comunican las vistas (*Widgets* en los ficheros de definición de interfaces gráficas) con los datos, permitiendo la manipulación de los segundos en función de las acciones que el usuario realice. Así pues, el controlador del proyecto está formado por las pantallas de la aplicación.

Este proyecto usa una única actividad con un widget de navegación que permite mostrar en un único espacio de contenido las diversas pantallas desarrolladas. Se tomó esta decisión de diseño dado que era la forma más actual de implementarla, además de permitir una rápida transición entre las diferentes funcionalidades que ofrece la aplicación. Esto convierte a la pantalla *Tablero* en la principal.

De esta forma, cada pantalla hace referencia a ciertos requisitos funcionales, ya que cada una tiene ciertas responsabilidades. Las pantallas diseñadas para su uso en la aplicación son:

- *Tablero/Board*: Es la pantalla principal y primera en ser vista al iniciar la aplicación. Muestra la expresión actual y las acciones permitidas sobre ella. Es la encargada de reaccionar a las pulsaciones del usuario y obrar en consecuencia, modificando la expresión acorde con la selección actual y la acción pulsada. También muestra una referencia hacia la pantalla *Ayuda* y permite deshacer una acción inmediata sobre la expresión. Hace referencia a los requisitos funcionales RF1-5 y parte del RF-8, además de los requisitos no funcionales RNF1-3 y RNF5.
- *Ayuda/Help*: Es la encargada de explicar el funcionamiento de la pantalla *Tablero*. Sólo es accesible mediante el botón con el mismo nombre en dicha pantalla. Se muestran sendos mensajes de información directamente en la posición de los elementos a los que hacen referencia (expresión actual, botones de acción, botón de deshacer última acción). Hace referencia al requisito no funcional RNF-7.
- *Expresiones de ejemplo/Showcase*: Muestra una lista con unas expresiones predefinidas que se encuentran siempre en la aplicación. Seleccionar una resetea la expresión actual, pasando a ser ésta la seleccionada. Acto seguido, retorna a la pantalla *Tablero*. Hace referencia al requisito funcional RF-7.
- *Historial de expresiones/History*: Encargada de mostrar todos los pasos que se han ido realizando sobre las expresiones actuales. Al seleccionar una entrada del mismo, se cambia la expresión actual por la seleccionada, eliminando todos los cambios posteriores desde la selección y volviendo a la pantalla *Tablero*. Hace referencia a los requisitos funcionales RF-6 y RF-8.
- *Ajustes/Settings*: Permite controlar los ajustes de la aplicación. Hace referencia a los requisitos no funcionales RNF-6 y RNF-8, además de parte del requisito funcional RF-8.

- *Color de tablero*: permite la selección de un color de tablero claro u oscuro, acorde con lo que decida el usuario. Los colores varían entre verde pizarra y blanco.
- *Color de la selección*: permite la selección de un color de contraste para resaltar el término de la expresión actual que seleccione el usuario. Puede elegirse entre anaranjado, azul claro o rojo.
- *Navegación o Navigation Drawer*: No es una pantalla en sí, sino que se trata del elemento que gestiona la transición entre ellas. Muestra las diversas pantallas accesibles y permite ir a cualquier pantalla con excepción de *Ayuda*.

El esquema de navegación queda reflejado en la Figura 5.2:

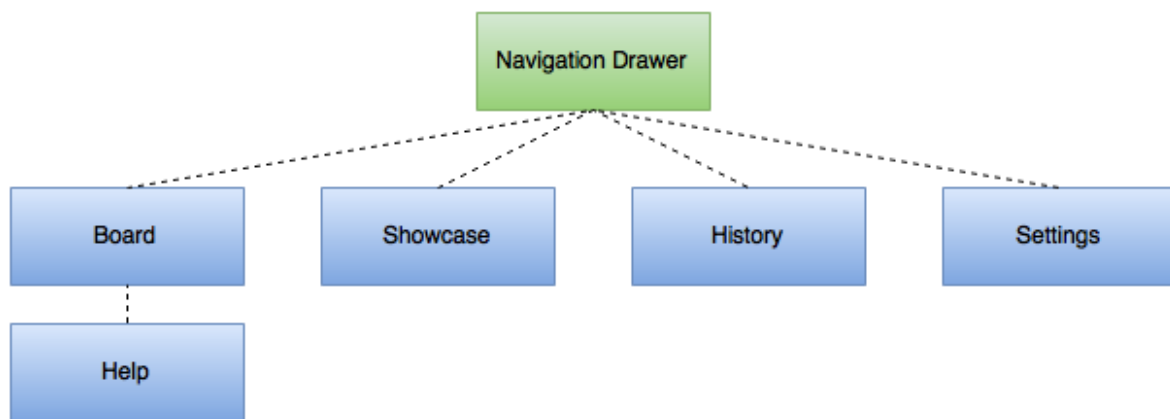


Figura 5.2: Navegación en la aplicación

5.1.3 Vista

La capa Vista en una aplicación Android viene dada por la representación visual de los elementos que forman un *layout*, una cuadrícula imaginaria que distribuye elementos visuales dentro de un contenedor. Es decir, la capa de Vista en Android viene dada por los ficheros XML usados para la definición de los *layouts*, y por las clases que extiendan la clase *View* (sección 6.4).

En este proyecto se ha utilizado un *layout* para cada pantalla que incluye elementos propios de Android y un componente propio, que extiende la clase *View* de Android, con el fin de poder manejar los datos visuales a voluntad. Este componente se usa en el *layout* de la pantalla *Tablero*.

La representación visual de cada pantalla se muestra en la sección 5.2 Maquetas (a continuación).

5.2 Maquetas

Para una mejor comprensión del diseño visual final de la aplicación se realizaron unas maquetas que representaran el aspecto de las pantallas diseñadas. A continuación se muestran las maquetas más importantes, junto con una descripción de sus elementos. No se incluye la representación gráfica del widget de navegación, ya que únicamente consiste en un menú desplegable con una lista de las pantallas accesibles.

La Figura 5.3 muestra la pantalla *Tablero* o *Board*:

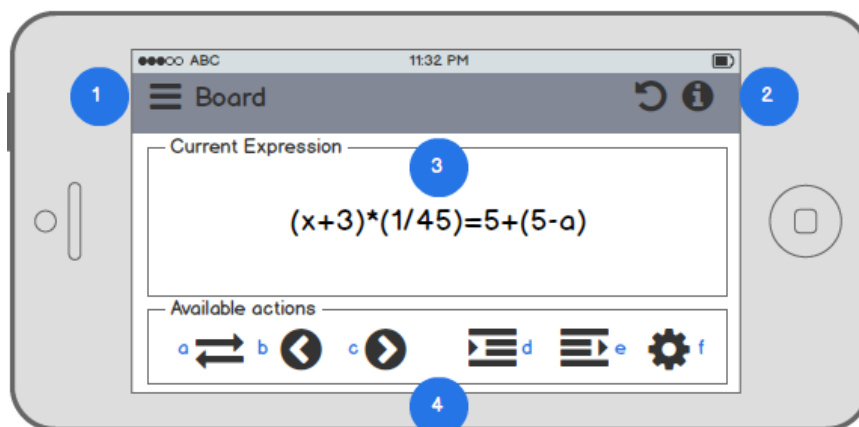


Figura 5.3: Pantalla *Tablero/Board*

1. *Botón de activación de la navegación.* Tanto pulsando el icono como deslizando el dedo desde el borde exterior izquierdo de la pantalla hacia dentro de la misma, aparecerá el menú de navegación. Este botón es común en todas las pantallas.
2. *Iconos de “deshacer” y “ayuda” (respectivamente).* El primero deshace una acción anterior inmediata realizada sobre la expresión actual, mientras que el segundo muestra una pantalla con información sobre el uso del tablero y los botones de acción.
3. *Expresión actual.* Muestra el estado actual de la expresión que está siendo manipulada en ese instante. Pulsando o manteniendo pulsado sobre sus elementos se activan las acciones posibles y se destaca la selección sobre el resto de la expresión. El tamaño de la letra de la expresión se ha fijado de manera que sea compatible con la mayor cantidad de dispositivos (tomando como referencia una pantalla de 4 pulgadas), manteniendo a su vez una proporción adecuada entre los términos para facilitar la selección de los mismos. Si la letra fuera más pequeña entorpecería esta función; si fuera más grande no sería posible utilizar expresiones más grandes.
4. *Acciones posibles.* Botones que permiten al usuario interactuar con la expresión una vez ha realizado su selección. Cambian de color acorde con si se puede realizar o no la acción que representan. De izquierda a derecha las acciones son:
 - a. *Cambiar de lado.* Cambia un término de lado de una igualdad.
 - b. *Conmutar hacia la izquierda.* Conmuta un término con el inmediato izquierdo dentro de una subexpresión.
 - c. *Conmutar hacia la derecha.* Conmuta un término con el inmediato derecho dentro de una subexpresión.
 - d. *Asociar.* Permite asociar dos o más términos de una expresión. Deben elegirse dos, quedando asociados todos los términos que haya entre medias.
 - e. *Desasociar.* Permite eliminar los paréntesis de una subexpresión cuyo padre tenga el mismo operador.
 - f. *Operar.* Realiza la operación acorde con la selección actual: realizar la operación con los términos, propiedad distributiva, factor común.

La figura 5.4 refleja la visualización de la pantalla *Historial* o *History*:

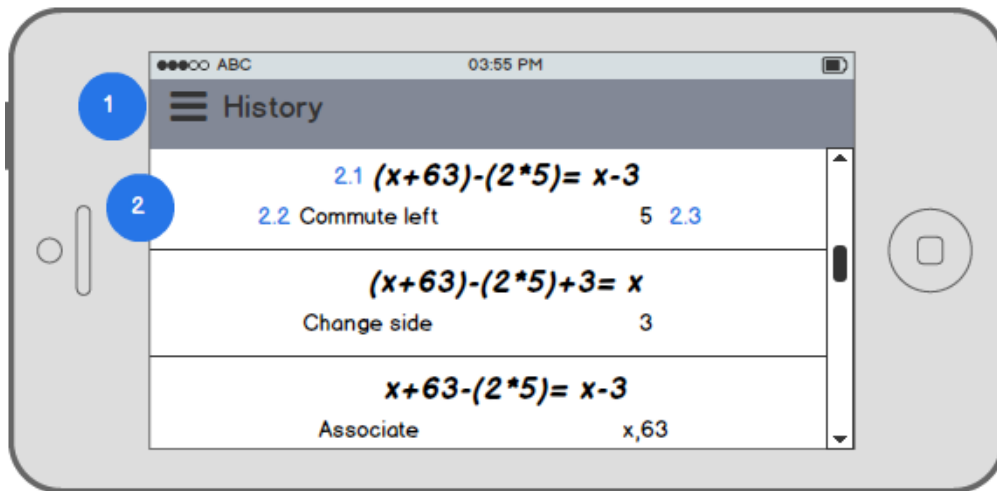


Figura 5.4: Pantalla *Historial/History*

1. *Botón de activación de la navegación.* Tanto pulsando el icono como deslizando el dedo desde el borde exterior izquierdo de la pantalla hacia dentro de la misma, aparecerá el menú de navegación. Este botón es común en todas las pantallas.
2. *Lista de acciones.* Esta lista contiene un registro de las acciones realizadas sobre la expresión actual. Por ejemplo, en este caso se indica que el 5 va a conmutar hacia la izquierda, cambiando su posición con el -2. Sus campos son:
 - 2.1. *Expresión inicial.* Expresión sobre la que se aplicó la acción.
 - 2.2. *Acción.* Acción realizada por el usuario sobre la expresión inicial.
 - 2.3. *Selección.* Subexpresión que indica la selección del usuario al realizar la acción.

La pantalla *Expresiones de ejemplo* o *Showcase* es muy similar a esta última, cuyo único cambio son los elementos de la lista: solamente muestra una expresión; si se pulsa en ella, se reinicia la expresión actual, sin perder las entradas anteriores en el historial.

Por su parte, la pantalla *Ajustes* o *Settings* se visualiza de manera similar a la mostrada en la figura 5.5:

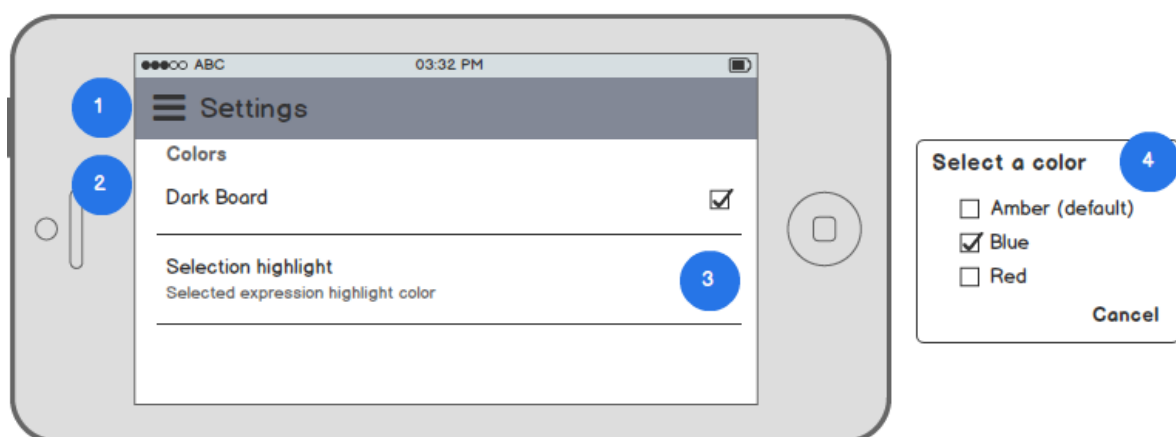


Figura 5.5: Pantalla *Ajustes/Settings*

1. *Botón de activación de la navegación.* Tanto pulsando el icono como deslizando el dedo desde el borde exterior izquierdo de la pantalla hacia dentro de la misma, aparecerá el menú de navegación. Este botón es común en todas las pantallas.

2. *Selección de color de fondo de tablero.* Cambia el color de fondo del tablero, pudiendo alternar entre verde pizarra (oscuro) o blanco (claro). Esta opción también cambia el color de la expresión a blanco o negro, respectivamente.
3. *Color de contraste de la selección.* Cambia el color de la subexpresión seleccionada, mostrando una lista de colores disponibles (4).
4. *Selector de color de contraste.* Permite elegir un color de los disponibles para el contraste: anaranjado (por defecto), azul claro o rojo.

El diseño final de la aplicación (su aspecto visual final) puede verse en la sección Anexo A: Manual de usuario. En él se muestran imágenes de la aplicación en funcionamiento explicando el funcionamiento de la misma.

6 Desarrollo

En este capítulo se tratan los aspectos de implementación de la aplicación, entrando en un mayor detalle de las funcionalidades implementadas. No es más que la explicación del código resultante del análisis y diseño descritos en los dos capítulos anteriores.

Dado que la aplicación se llevó a cabo mediante la metodología ágil descrita en la sección 3.2, los diagramas de clases se llevaron a cabo a medida que la funcionalidad iba incrementándose, añadiendo ésta al diagrama final. En esta sección se muestran los diagramas finales, una vez quedó terminada la aplicación. De esta forma se aseguró que toda funcionalidad implementada quedara reflejada en su respectivo diagrama.

No obstante, y para una mayor sencillez en la explicación, se muestran las clases y métodos clave del proyecto, es decir, aquellos elementos que mejor reflejan el código implementado. Los diagramas de clase pueden encontrarse en el Anexo B.

Por su parte, todo el código desarrollado puede encontrarse en *AndroidCAS* [23], el repositorio de *GitHub* creado para albergar todo el proyecto.

6.1 Estructura del proyecto

Android Studio permite una separación clara entre el código java y los ficheros XML de definición de recursos variables según el dispositivo, de modo que en una carpeta *java* se encuentra el contenido del primero y en otra *res* el segundo. Una carpeta *assets* contiene recursos que no varían con el dispositivo (en este proyecto únicamente contiene la fuente usada para dibujar las expresiones). Aparte queda el fichero *AndroidManifest.xml*, que contiene información esencial sobre la aplicación.

6.1.1 Fichero *AndroidManifest.xml*

Este fichero indica algunos datos esenciales para la definición de la aplicación Android. Contiene el nombre de la aplicación, el icono de lanzamiento, el estilo de los temas visuales a usar en cada una de las actividades, la orientación del dispositivo para cada actividad, y el tipo de actividades que son.

En este caso, el proyecto consta de una única actividad, con orientación forzada a ser en modo paisaje (horizontal), y que sirve como lanzador y punto de entrada a la aplicación (*android.intent.category.LAUNCHER* y *android.intent.action.MAIN*, respectivamente).

La orientación se fuerza a que sea siempre horizontal para permitir un mayor espacio disponible a la hora de dibujar las expresiones.

6.1.2 Código Java

Dentro de la carpeta *java* (y dentro del paquete por defecto) encontramos los paquetes de código que se han implementado a lo largo del desarrollo. Para una mejor vista del mismo, se agruparon los paquetes siguiendo la arquitectura MVC, de manera que los componentes con misma funcionalidad siguen una estructura jerárquica. Así pues, se encuentran tres paquetes principales: *model*, *view*, *controller*, cada uno con subpaquetes que diferencian los componentes internos. Además existe un cuarto paquete, *util*, que contiene las clases de utilidades creadas como apoyo al resto de elementos.

- *model*: se definen dos subpaquetes dentro del mismo, *history* y *cas*. El primero hace referencia a las clases encargadas de gestionar el historial de expresiones y el segundo a aquellas que implementan la interfaz con el motor algebraico.
- *controller*: contiene los subpaquetes *activity*, *fragment* y *listener*. Los dos primeros hacen referencia a las Actividades y Fragmentos de Android, respectivamente, mientras que el tercero define unas interfaces de comunicación entre la capa Vista y Controlador (consultar sección 6.3 para más información).
- *view*: en este paquete se encuentra la clase *ExpressionView*, encargada de dibujar las expresiones matemáticas. Contiene además el subpaquete *drawable*, en cuyo interior se definen las clases que añaden los elementos necesarios a una expresión para poder dibujarse en la pantalla de manera recursiva.
- *util*: las utilidades usadas en el resto de paquetes incluyen utilidades del CAS, de preferencias y generales.

6.1.3 Recursos

La carpeta *res* contiene la definición de los recursos utilizados en la aplicación, desde las imágenes usadas hasta las interfaces gráficas, pasando por las preferencias de configuración. También contiene los valores usados como dimensión y los elementos de los menús de la aplicación, así como las cadenas de caracteres para distintos idiomas y los colores disponibles.

- *drawable*: imágenes e iconos de menús.
- *layout*: distribución de elementos para las pantallas. Forman parte de la capa Vista.
- *menu*: menús desplegables de la aplicación (*drawer*, menús de la barra de acción o *toolbar*).
- *mipmap*: iconos invariables de la aplicación. En este caso únicamente el icono del lanzador de la aplicación.
- *values*: valores de preferencias, colores, dimensiones, cadenas de texto, estilos de temas.
- *xml*: definición de las preferencias de la aplicación que se mostraran en la pantalla *Settings*.

6.2 Modelo/Model

Contenida en el paquete *model*, la capa Modelo hace referencia a todas aquellas clases que contengan la definición de los datos a usar en la aplicación. En el caso de este proyecto se incluyen dentro de ella:

- La librería externa del CAS.
 - Las clases básicas de elemento de expresión matemática y motor de manipulaciones, así como su fichero de configuración.
 - La interfaz de comunicación del CAS con el controlador.
- Las clases que implementan el historial de expresiones.
 - Contenedor de los datos de una expresión.
 - Interfaz de conexión del historial con el controlador.

6.2.1 Motor Algebraico Externo

El motor algebraico con el que se comunica la aplicación está conformado por dos clases agrupadas en un archivo *.jar* [24]: *Operation* y *AlgebraicEngine*. Los detalles de su

implementación pueden encontrarse en *G-CAS* [2], aquí se hace referencia a la funcionalidad usada en la aplicación, de manera simplificada.

6.2.1.1 Operation

La unidad de datos básica viene dada por la clase externa *Operation*. Ésta es la encargada de modelar una expresión matemática como una lista de listas. Se puede considerar como una lista padre que tiene asociado un operador y una lista de argumentos. Todo dentro del CAS es una lista, por lo que los números son otro tipo de listas con un único argumento.

Así pues, teniendo acceso al operador de una lista y a sus argumentos, se puede manipular cualquier subexpresión y actuar en consecuencia cuando se detecten acciones sobre ellas.

6.2.1.2 AlgebraicEngine y fichero de configuración usado

La clase *AlgebraicEngine* tiene como tarea manipular los elementos *Operation*, transformando unos en otros mediante unas manipulaciones básicas que ofrece el CAS por defecto y otras definidas en un fichero de configuración a partir del cual se generan el resto de reglas de equivalencia. En efecto, la clase *AlgebraicEngine* es el resultado de la implementación de las manipulaciones encontradas en dicho fichero.

La clase toma una expresión como actual, sobre la cual se aplicarán las manipulaciones deseadas, pudiendo cambiar la expresión mediante una función de inicialización.

Las manipulaciones pueden ser llamadas sobre un objeto que instancie la clase del motor. De esta forma, por ejemplo, puede llamarse a la propiedad conmutativa mediante la función *commutativeProperty*.

Dado que estas reducciones no tienen un nombre asociado ya que se generan automáticamente a partir de un fichero de configuración, es necesaria la creación de una interfaz de comunicación que permita agrupar las reducciones en funciones que implementen las acciones matemáticas deseadas.

El fichero de configuración contiene la definición de las manipulaciones que se considerarán válidas y que se podrán aplicar en el CAS. Por lo tanto es el elemento que permite elegir qué acciones pueden realizarse sobre una expresión (recuérdese que una expresión viene modelizada por la clase *Operation*).

En esta sección se muestra un ejemplo que muestra las reducciones para realizar un factor común. El fichero de configuración completo usado para esta aplicación puede encontrarse en el Anexo C.

Ejemplo de uso de reducciones

Suponiendo que se desee realizar la extracción de un factor común en una expresión dada

$$3 + (4*3*x) + (8*3)$$

las reducciones necesarias para realizarlo son las mostradas en la tabla 6.1:

Propiedad/Reducción	Reducción
Conmutativa (primitiva)	<i>commutativeProperty</i>
Asociativa (primitiva)	<i>associativeProperty</i>
Convertir en producto por unidad (red8)	<i>_a >>> * [&ONE[, _a]</i>
Factor común (red19)	<i>+[*[_b,_a],*[_c,_a]] >>> * [+[_b,_c],_a]</i>

Tabla 6.1: Propiedades y reducciones para el factor común

Las reducciones primitivas están contenidas en el propio CAS, mientras que el resto se toman del fichero de configuración para ser autogeneradas por *G-CAS* y ser incluidas en el motor al final.

Así pues, para realizar la manipulación buscada primero se conmutan los términos para que queden en última posición y acto seguido se usa la reducción 8 para convertir los elementos simples en producto por la unidad:

$$(1*3) + (4*x*3) + (8*3)$$

Sobre esta expresión se utiliza la propiedad asociativa para crear un pivote y operar de acuerdo con la reducción 19:

$$((1*3) + (4*x*3)) + (8*3) \quad ((1 + (4*x)) * 3) + (8*3)$$

Una vez creado el pivote se vuelve a aplicar la reducción 19 sobre él y el siguiente elemento, procediendo de esta manera hasta resultar en la extracción del factor común en todos los términos:

$$((1 + (4*x)) + 8) * 3$$

Para terminar, se deshacen las asociaciones creadas durante la creación del pivote:

$$(1 + (4*x) + 8) * 3$$

6.2.1.3 Interfaz CASAdapter

La interfaz *CASAdapter* y la clase *CASImplementation* son las encargadas de proporcionar a la capa Controlador una batería de funciones que realicen las acciones deseadas sobre las expresiones seleccionadas en un determinado instante. La segunda no es más que la implementación de la primera. Esto permite poder cambiar la implementación siempre que se desee sin alterar las llamadas externas en los componentes que usen la interfaz.

Para lograr tal propósito, se ha implementado un patrón *Singleton* [24], de modo que la referencia al CAS sea única en toda la aplicación, consiguiendo que sólo pueda manipularse una expresión global al mismo tiempo.

La clase que realiza la implementación contiene un objeto *AlgebraicEngine*, de manera que puede acceder a todas las manipulaciones o *reducciones* del mismo, y aplicarlas en el orden deseado para lograr una acción concreta sobre una expresión dada (la actual del CAS). Así pues, cada función de la interfaz tiene como implementación una sucesión de

reducciones que se aplican a la expresión actual del CAS, obteniendo otra expresión como resultado. Con el fin de centrarse en la funcionalidad matemática, la tabla 6.2 recoge los métodos que implementan las acciones permitidas en la aplicación. Una explicación detallada del funcionamiento interno de las mismas puede encontrarse en el Anexo D:

Función	Propiedad que implementa
<i>commute</i>	Propiedad conmutativa (una posición)
<i>associate</i>	Propiedad asociativa
<i>dissociate</i>	Desasociar términos
<i>commonFactor</i>	Extraer factor común
<i>operate</i>	Operar términos de una expresión
<i>changeSide</i>	Cambio de lado en ecuación
<i>distribute</i>	Propiedad distributiva

Tabla 6.2: Funciones de *CASAdapter* y propiedad que implementan

6.2.2 Historial de expresiones

Para tener acceso en todo momento a las acciones que se han ido realizando sobre la expresión actual es necesario crear un contenedor de las mismas, o dicho de otra manera, una “base de datos” que permita gestionarlas.

Para ello se han implementado dos clases y una interfaz: *ExpressionRecord*, *ExpressionHistoryDB* e *ExpressionHistory*. La primera representa un dato del historial, mientras que la segunda es una implementación de la interfaz de comunicación (la tercera) para que el controlador pueda llamar a las funciones de guardado y acceso a los datos.

De esta forma, la interfaz permite listar los cambios realizados, añadir un nuevo registro y volver hacia atrás a alguno de ellos.

Un dato del historial se compone de cuatro campos:

- mGlobalExpression*: expresión principal sobre la que se realizó la acción en formato texto.
- mAction*: acción ejecutada sobre la expresión principal en formato texto.
- mSelectedExpression*: subexpresiones seleccionadas al realizar la acción en formato texto.
- mCASExpression*: representación en texto de la expresión global en el formato propio del CAS. Se utiliza para tener una referencia en caso de querer volver a ella.

Si se desea volver a una expresión anterior, el historial devolverá el cuarto campo para que el CAS sea inicializado de nuevo con dicha expresión, sustituyendo la expresión actual por la que se quiera.

6.3 Controlador/Controller

El controlador, como se ha mencionado anteriormente y a lo largo del capítulo 5, se compone de las Actividades y Fragmentos que Android proporciona, así como las clases auxiliares creadas para apoyar las tareas de control que se deben realizar en este componente. Está dentro del paquete *controller*.

El funcionamiento del controlador es muy simple: por un lado gestiona la navegación entre pantallas de la aplicación, gracias a la actividad *MainActivity* (y su clase predecesora, *NavigationDrawerFragmentActivity*). Por otro lado, cada Fragmento de Android se ocupa de controlar su respectiva pantalla, es decir, que cada pantalla se autogestiona, pasando los datos necesarios a la actividad principal.

6.3.1 Navegación

La clase *MainActivity* dispone de un *widget*, el *NavigationDrawer*, que junto con su contenedor *DrawerLayout* permite elegir una pantalla y navegar a ella mediante la pulsación en un menú emergente lateral, donde se listan todas las pantallas disponibles. Cada pantalla posee un identificador que permite seleccionarla de manera única. De esta forma, una única actividad es capaz de cambiar un elemento concreto por el contenido de diversos fragmentos, utilizando siempre el mismo espacio visual. Esto indica que lo que se visualiza siempre es la Actividad *MainActivity*, con el menú de navegación y el contenido que se indique mediante un Fragmento.

Al mismo tiempo, y gracias a la interfaz *Callbacks*, las pantallas (que vienen representadas por Fragmentos de Android), son capaces de utilizar los métodos de dicha interfaz para delegar funcionalidad superior en la actividad. La actividad implementa la interfaz, mientras que los fragmentos guardan una referencia a la primera almacenándola en un objeto de clase *Callbacks*. De esta manera cada fragmento tiene una conexión directa con la actividad principal, siendo ésta la encargada de gestionar el cambio entre ellos.

Por tanto, existen dos modos de navegación entre los fragmentos o pantallas de la aplicación:

1. **Navegación por menú:** Es la navegación estándar (figura 6.1). Se aplica cuando se desea ir de una pantalla a otra sin ser resultado de una acción. En la figura sólo aparecen dos, pero es extensible a todos los fragmentos que forman la aplicación:
 - a. Se selecciona la pantalla a la que se desea ir, mediante el identificador del fragmento.
 - b. La actividad invoca el método con el identificador adecuado y se navega a la nueva pantalla.

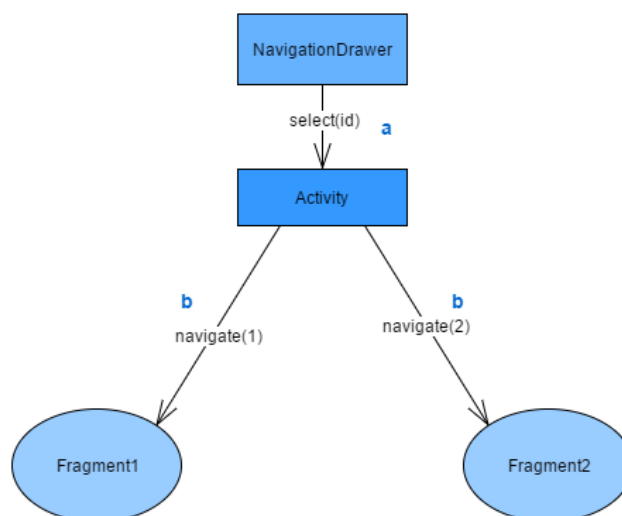


Figura 6.1: Navegación por menú

2. **Navegación por acción:** Esta navegación se origina cuando en un fragmento se desea o bien volver al anterior o cuando una acción determinada necesita navegar a un cierto fragmento (por ejemplo, al seleccionar una expresión de ejemplo en la pantalla *Expresiones de ejemplo*, se vuelve a la pantalla *Tablero*). La figura 6.2 refleja el procedimiento.
 - a. Cada fragmento, al crearse desde la Actividad *MainActivity*, guarda una referencia a ésta como un objeto de tipo *Callbacks*, interfaz que la actividad implementa.
 - b. Cuando se origina la acción desencadenante, el fragmento invoca el método de la interfaz con el identificador del fragmento objetivo. De esta forma, la llamada pasa a la actividad.
 - c. La actividad invoca su método interno de navegación, permitiendo ir al fragmento deseado.

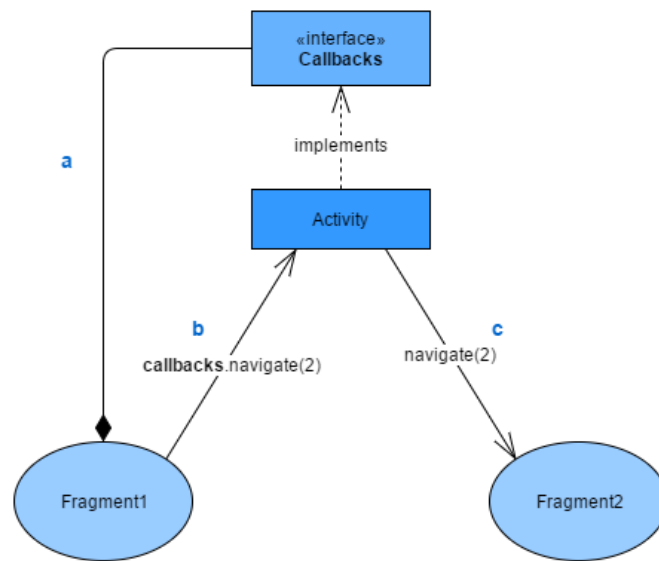


Figura 6.2: Navegación por acción

6.3.2 Autogestión de pantallas

Dado que cada Fragmento de Android representa el contenido de una pantalla, se explica su funcionalidad por separado. Cada fragmento se gestiona a sí mismo, controlando sus objetos internos y utilizando la interfaz *Callbacks* cuando es necesario invocar a la actividad principal, *MainActivity*. En esta sección a veces se usa el nombre del fragmento como referencia a la pantalla que implementa, de manera que quedan del siguiente modo (tabla 6.3):

Pantalla	Fragmento
<i>Tablero/Board</i>	<i>ExpressionFragment</i>
<i>Ayuda/Help</i>	<i>HelpFragment</i>
<i>Historial/History</i>	<i>HistoryFragment</i>
<i>Ajustes/Settings</i>	<i>SettingsFragment</i>
<i>Expresiones de Ejemplo/Showcase</i>	<i>ShowcaseFragment</i>

Tabla 6.3: Pantallas y Fragmentos asociados

6.3.2.1 ExpressionFragment

Este Fragmento de Android gestiona la visualización gráfica de una expresión matemática y las acciones aplicables a ella. Para ver la distribución visual del fragmento, véase la figura 5.3 en la sección 5.2.

El fragmento se compone de dos botones superiores colocados en lo que en Android se llama *ToolBar* o *ActionBar*, una referencia a la clase *ExpressionView* (ver sección 6.4.2), y una botonera que permita manejar las acciones aplicables. Para esto último se ha implementado una clase adicional, *ActionButtons*, de manera que permita separar las acciones disponibles en un componente externo a la clase, mientras que sigue siendo ésta quien realiza la ejecución de las acciones.

Con el fin de ilustrar mejor la funcionalidad de la clase, la figura 6.3 representa los posibles flujos que pueden darse (sin contar la navegación por menú):

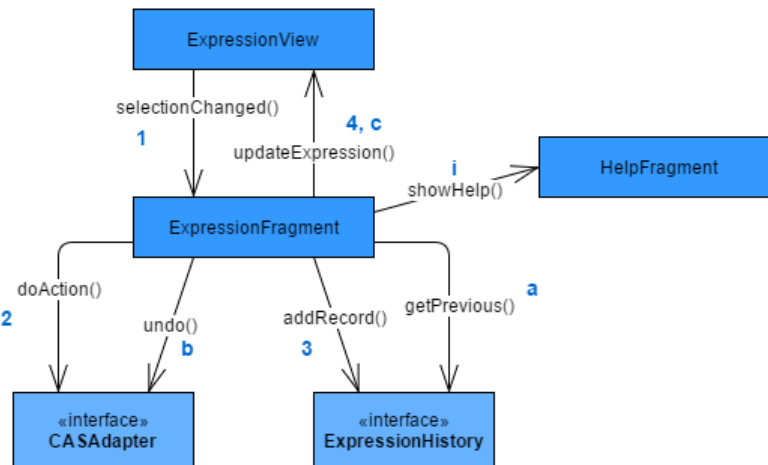


Figura 6.3: Funcionamiento de *ExpressionFragment*

1. Para conocer en todo momento la selección realizada sobre la vista, se ha forzado al fragmento a implementar una interfaz que permite avisar cuándo el usuario ha realizado una selección, ya sea simple o múltiple. Esta interfaz se conoce como *listener* y viene dada por la interfaz de java *OnExpressionActionListener*. La clase *ExpressionView* guarda una referencia a este *listener*, invocando sus métodos cuando el usuario realiza una selección o la cancela. De esta forma, en cuanto se ha seleccionado una subexpresión, el fragmento lo detecta gracias a la interfaz y guarda la selección actual en un campo interno, listo para actuar en cuanto el usuario presione un botón de acción.
2. Dada una expresión matemática, una selección del usuario y una acción a realizar, debe transformarse la primera en otra expresión que se obtenga como resultado de aplicar la acción sobre la selección del usuario. Esto se consigue mediante la inclusión de la interfaz *CASAdapter* (descrita en la sección 6.2.1.3). *ExpressionFragment* llama a estos métodos en función del botón pulsado, conociendo cuál ha sido gracias a la clase *ActionButtons*.
3. Así mismo, cada vez que se realiza una acción debe poder guardarse en un historial. Esto se consigue gracias a la interfaz *ExpressionHistory* usada de manera interna.

4. Una vez realizada la acción, se debe avisar a *ExpressionView* ya que la expresión interna del CAS ha cambiado. La clase de la capa Vista implementa una interfaz *listener* con una función que permite regenerar la vista de la expresión actual una vez se ha realizado una acción sobre ella. La interfaz mencionada es *OnExpressionUpdatedListener*.

También es tarea de esta clase mostrar una pantalla de ayuda, mostrando al usuario cómo interactuar con el tablero:

- i. Una vez pulsado el botón designado para tal efecto, se le indica a la clase *MainActivity* que debe navegar al Fragmento *HelpFragment* mediante la interfaz *Callbacks* (ya explicada en la sección 6.3.1).

La clase además debe permitir deshacer una acción inmediata, volviendo a la expresión anterior:

- a. Una vez pulsado el botón de deshacer, se llama al método adecuado de la interfaz *ExpressionHistory* para obtener la expresión anterior.
- b. Se le indica al CAS mediante la interfaz *CASAdapter* que debe volver a una expresión anterior, inicializando de nuevo la expresión actual.
- c. Se notifica a *ExpressionView* el cambio de la expresión, permitiendo el refresco de la vista.

6.3.2.2 *HelpFragment*

Este Fragmento de Android se ocupa de mostrar al usuario la información necesaria para interactuar con la pantalla *Tablero*. Utiliza campos de texto cuya información contiene cómo usar la selección de expresiones y los botones que reflejan las acciones disponibles.

6.3.2.3 *HistoryFragment*

La funcionalidad que implementa esta clase se reduce a gestionar una estructura que sirve como base de datos en cuyo interior guarda los registros de acciones sobre las expresiones actuales mostradas en la pantalla *Tablero*.

El funcionamiento de este Fragmento de Android es muy simple: provee una lista de expresiones que hacen referencia a cada uno de los cambios realizados. Al pulsar sobre una de ellas, se reinicia el CAS con dicha expresión, y se vuelve a *ExpressionFragment*, el cual cargará la expresión para trabajar con ella.

La lista de expresiones se obtiene llamando al método *getHistory* de la interfaz *ExpressionHistory*. Éste devuelve una lista de registros con la información guardada hasta el momento. Para mostrarla en pantalla, se utiliza un *widget* de Android llamado *RecyclerView*, el cual, junto con el apoyo de dos componentes, *Holder* y *Adapter*, es capaz de mapear cada registro en un *layout* que contiene los campos a mostrar. De modo que el *RecyclerView* se carga con los datos que le indica el *Adapter*, y éste a su vez se llena con los registros que devuelve el historial desde la capa Modelo.

El componente *Holder* no es más que un contenedor para un registro *ExpressionRecord* (ver sección 6.2.2), es decir, cada uno de los elementos que *RecyclerView* muestra en la pantalla. En el código viene implementado por la clase interna *ExpressionHolder*. Su funcionamiento consiste en dado un registro, mapearlo a los campos internos para

mostrarlo, y asignar un *listener* que permita cargar la expresión asociada al registro cuando se seleccione ese registro.

Por su parte, *Adapter* es la clase que permite gestionar internamente una lista de elementos en un *RecyclerView*. En este caso es la clase interna *ExpressionAdapter* quien lo implementa. Este componente carga el *layout* asociado al *Holder*, guarda una lista para usarla internamente y asocia un *Holder* a un registro. Así se consigue que cuando se seleccione un *Holder*, se esté seleccionando al mismo tiempo el registro asociado.

6.3.2.4 ShowcaseFragment

El funcionamiento de esta clase es idéntico al de *HistoryFragment*, con el único cambio de que en vez de utilizar *ExpressionRecord* como datos, utiliza cadenas de caracteres. De esta forma los datos se obtienen de la clase *CASUtils* (explicada en la sección 6.5.1), invocando al método que devuelve una lista con las expresiones de prueba.

Al realizar una selección sobre una de ellas, se reinicia la expresión actual del CAS y se vuelve a *ExpressionFragment* para que cargue dicha expresión.

6.3.2.5 SettingsFragment

La clase *SettingsFragment* no extiende de la clase *Fragment* de Android, sino de *PreferenceFragmentCompat*, una versión de *Fragment* que modeliza una pantalla de preferencias de usuario (los ajustes de la aplicación).

Su funcionalidad se reduce a cargar un fichero de preferencias en XML desde la carpeta *xml* en el directorio *res*, el cual contiene la definición de las preferencias, e inicializar lo que en Android se conoce como *SharedPreferences*, preferencias compartidas. Estas preferencias son aquellas accesibles desde cualquier punto de la aplicación.

La gestión de preferencias se realiza de forma automática: si el usuario cambia alguna, se quedará registrada en la clase *SharedPreferences*. Para usarlas, la clase de apoyo *PreferenceUtils* da una serie de funciones que acceden a esta clase, devolviendo el valor de la preferencia seleccionada. La explicación de la funcionalidad de esta última clase se encuentra en la sección 6.5.2.

6.3.3 Acciones y selección del usuario

Para hacer efectivas las interacciones del usuario con la aplicación, es necesario disponer de una referencia a las selecciones que se hagan en la interfaz (capa Vista), asociando más tarde ciertas acciones que invoquen a las funciones que provee la interfaz del CAS.

Esto se consigue mediante una referencia a la clase *ExpressionView* (sección 6.4.2), permitiendo la comunicación entre la interfaz y el controlador. Gracias a ello se puede conocer la selección que el usuario está realizando en cada momento, y una vez actualizada la expresión mediante manipulaciones del CAS, actualizar la interfaz pasando la nueva expresión con el *listener OnExpressionUpdated*.

Por otro lado, y una vez conocida la selección que realiza el usuario, es necesario poder mapear los botones de acción que contiene la clase *ActionButtons* a las funciones de

CASAdapter. La clase *ActionButtons* contiene los botones de la interfaz que permiten realizar las operaciones matemáticas permitidas por la aplicación.

De esta forma, para cada botón se usa una función del CAS (explicadas en la sección 6.2.1.3).

6.4 Vista/View

La capa Vista está compuesta por los *widgets* que Android provee como parte de su código. Éstos no son más que cada uno de los elementos que componen la pantalla (desde un campo de texto hasta los botones). En la aplicación se ha creado un elemento propio, *ExpressionView*, que extiende de la clase básica *View*, de modo que represente visualmente una expresión matemática. La clase *View* necesita un lienzo sobre el que dibujar sus elementos, un *canvas*, además de un método *onDraw* que permita a los objetos que la componen autodibujarse sobre el mismo *canvas*.

Los puntos a tener en cuenta para entender la decisión tomada son:

1. Se debe pintar texto, pero de manera “especial” (una expresión matemática es una lista de listas según la implementación del CAS usado); es necesaria la creación de clases de apoyo para la representación unitaria de los términos.
2. Debe controlarse la selección del usuario, modificando el contenido de la vista y resaltando los elementos que se encuentran en dicha selección: debe utilizarse un detector de gestos que maneje las pulsaciones del usuario y actualice los datos de selección.

Debido a los puntos anteriores, se separó la capa en dos componentes: el detector de gestos junto con el lienzo (*SelectionGestureListener* y *ExpressionView* respectivamente) y las clases de apoyo (*DrawableExpression* y las que heredan de ella).

6.4.1 *DrawableExpression*, clases de apoyo

Estas clases permiten la modelización de un elemento *Operation* como una vista en Android. Existen cuatro tipos a parte de la clase básica *DrawableExpression*: *DrawableExpressionList*, *DrawableSingleExpression* y *DrawableParenthesis*, y *DrawableOperator*.

A modo de resumen, la figura 6.4 muestra un ejemplo de cómo se considera una *DrawableExpression* cuando se dibuja en el *canvas*. Una descripción más detallada de cómo son internamente las clases puede encontrarse en el Anexo E.

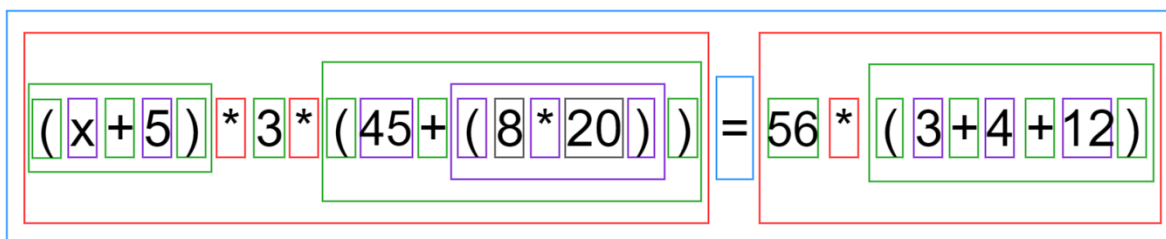


Figura 6.4: Representación visual de *DrawableExpressionList*

Se considera un enfoque modular interno: cada *DrawableExpressionList* se compone de objetos *DrawableExpression*, permitiendo utilizar todas las clases que heredan de ella

como elementos en una lista. De esta forma, la unidad mínima (los rectángulos más pequeños en la figura) responden a *DrawableSingleExpression* si son números o variables, *DrawableParenthesis* si son paréntesis y *DrawableOperator* si corresponden a un operador matemático.

En la figura, cada *DrawableExpressionList* tiene el borde de un color diferente para diferenciarlo. Dentro de ellas se encuentran las clases que modelizan expresiones únicas. Los elementos que se consideran al mismo nivel también tienen el mismo color de borde, de manera que seleccionar un operador o un paréntesis implica seleccionar la subexpresión entera.

En la aplicación, los bordes no se muestran, pero son tomados en cuenta a la hora de seleccionar una expresión. Además se considera que todos los elementos tienen la misma altura de borde.

6.4.2 ExpressionView, SelectionGestureListener: Selección

La funcionalidad clave de *ExpressionView* reside en la selección de un objeto de tipo *Operation* a través de unas coordenadas en la pantalla.

Para ello extiende de *View*, la clase básica de Android para crear *widgets* personalizados. Su funcionalidad permite mostrar en la pantalla *Tablero* una expresión matemática, seleccionar sus términos y resaltarlos con un determinado color para destacar la selección actual frente al usuario.

Además, hay que controlar dos tipos pulsación, por lo que se hace necesaria la creación de una clase que extienda de la clase Android *GestureDetector*. Ésta no es más que un detector de gestos. Para la aplicación se ha utilizado la implementación de la clase *SimpleOnGestureListener*, ya que sólo se desea reconocer dos gestos: pulsación simple y larga (para activar selección múltiple).

De esta forma, y usando las clases de apoyo, se puede devolver el objeto *Operation* asociado a una pulsación en pantalla (en el *canvas*) en función de la selección:

6.4.2.1 Selección Simple

Se encuentra implementada dentro del método *onSingleTapUp* del componente *SelectionGestureListener*, llamado cuando se ha realizado una pulsación completa y única.

Permite la selección única de una subexpresión en la global. Para ello basta con detectar la pulsación del usuario, comprobar que ha pulsado dentro de una subexpresión, que no hay selección múltiple (ver sección inmediata posterior) y llamar a las funciones de selección de la clase *DrawableExpression*. Una vez obtenida la selección y cambiada de color en el *canvas*, se notifica al controlador mediante el *listener OnExpressionActionListener* de que la selección actual ha cambiado, permitiendo al componente de control actualizar los datos en función de la siguiente acción del usuario.

6.4.2.2 Selección múltiple

Esta funcionalidad se encuentra dividida entre los métodos *onLongPress* y *onSingleTapUp* del detector de gestos. El primero permite activar la selección múltiple al detectar que el

usuario ha realizado una pulsación larga, seleccionando el primer elemento, mientras que el segundo, una vez comprueba que está activa la selección múltiple, añade las siguientes selecciones a la colección de elementos seleccionados.

La comprobación de activación de selección múltiple se realiza mediante la escritura de una variable de estado (*flag*) de tipo *boolean*, que se cambia a *true* en el método *onLongPress* y se comprueba en *onSingleTapUp*. La variable cambia al estado *false* cuando se cancela la selección o se realiza de nuevo una pulsación larga, reiniciando la selección múltiple.

6.4.2.3 Selección cancelada

Si se detecta que la pulsación ha sido fuera de la expresión (la función de selección ha devuelto un valor nulo), se devuelve la selección a su estado vacío, volviendo al color normal en toda la expresión.

6.4.2.4 Actualización de la expresión

Una vez se han enviado al controlador (representado por *ExpressionFragment*) mediante el *listener* de acción, y el controlador modifica la expresión utilizando el CAS y su interfaz, es el propio controlador quien avisa a la clase *ExpressionView* de que ha habido un cambio en la expresión actual, y debe refrescarse la pantalla.

Para ello, la clase que sirve como lienzo implementa el *listener* *OnExpressionUpdated*, cuyo método *onExpressionUpdated* tiene como primer argumento la nueva expresión. Así pues, *ExpressionView* actualiza su campo interno de tipo *DrawableExpression* con la información contenida en la nueva expresión.

Con todo lo visto en esta sección, queda claro cuál es el flujo de datos de esta capa:

1. Se recibe una expresión a dibujar en pantalla. Se crean los campos necesarios mediante las clases de apoyo y se dibuja la expresión actual en función de ellas.
2. El usuario realiza una selección. El detector de gestos la procesa y se actualiza la selección actual, ya sea simple o múltiple, resaltando en pantalla dicha selección.
3. Se notifica al controlador del cambio en la selección, para permitir la manipulación de la expresión actual por parte del mismo.
4. Una vez el controlador da el aviso mediante el *listener* adecuado de que se han finalizado las manipulaciones y que la expresión ha cambiado, se repite el paso 1, pero actualizando la expresión.

6.5 Utilidades

El paquete *util* contiene las clases de apoyo que se usan a lo largo de toda la aplicación y por la mayoría de sus clases, por lo que todos los métodos de una clase de apoyo son estáticos, esto es, puede utilizarse sin instanciarse (crearse) un objeto de dicha clase.

Dado que en la aplicación hay clases que necesitan acceder a funcionalidades del CAS, leer el valor de las preferencias guardadas en *SharedPreferences* y tener acceso a una etiqueta común de Log y al fichero de fuente de texto para el estilo de algunos textos, es necesaria la creación de las clases *CASUtils*, *PreferenceUtils* y *Utils*, respectivamente.

6.5.1 CASUtils

Hay clases en la aplicación que necesitan acceder a algunas de las funcionalidades del CAS. Estas funcionalidades, algunas de ellas existentes en la interfaz, no son por defecto del CAS, si no que se consideran de apoyo al mismo. Entre ellas encontramos funciones que permiten, por ejemplo:

- Obtener la expresión infija de un objeto *Operation*.
- Saber si una expresión es una operación, una variable o una constante.
- Saber el operador de la expresión padre de una dada.
- Conocer la representación de un operador interno del CAS.
- Determinar si una expresión está en el nivel principal de la ecuación (o su padre o su abuelo son la igualdad en sí).
- Crear expresiones de ejemplo para utilizarlas.

6.5.2 PreferenceUtils

Como se mencionó en la sección 6.3.2.5, esta clase provee funciones que permiten la lectura de las preferencias cambiadas en la pantalla *Ajustes*. Las funciones acceden a las preferencias compartidas, *SharedPreferences*, de la aplicación y devuelven los valores a usar para el ajuste solicitado:

- *getBoardColor*: devuelve el color a usar en función de la preferencia que indica si el usuario desea un tablero de color oscuro (verde pizarra) o claro (blanco).
- *getExpressionColor*: consulta la misma preferencia de color de tablero oscuro o no para devolver un color de expresión claro (blanco) u oscuro (negro), respectivamente.
- *getExpressionHighlightColor*: consulta el valor de la preferencia en la que el usuario seleccionó el color de la selección de una expresión en el tablero y devuelve el color elegido (anaranjado, rojo o azul).

6.5.3 Utils

Esta clase de apoyo contiene únicamente la etiqueta del Log de Android común para todas las clases del proyecto, con fines de pruebas y control. Además se incluye en ella la ruta al fichero de fuente para los estilos de los textos de las expresiones.

7 Pruebas y resultados

Con el fin de encontrar posibles fallos en la ejecución de la aplicación, se prepararon y ejecutaron diversos tipos de pruebas durante el desarrollo del proyecto. Además, y dado el modelo de desarrollo aplicado, se consiguió realizarlas en paralelo con la funcionalidad, lo que agilizó la codificación.

7.1 Pruebas internas

Se entienden por pruebas internas aquellas relacionadas directamente con el código desarrollado en la aplicación. Durante el período de implementación del mismo se realizaron las pruebas de la lógica mediante tres aproximaciones.

7.1.1 Inspección, funcionamiento y refactorización

La primera barrera de pruebas consistió en una investigación visual del código implementado, comprobando que estuviera toda la funcionalidad requerida hasta el momento y estuviera implementada de manera correcta visualmente (estilo limpio y modular de programación, nombres representativos de funciones y variables internas, fácil entendimiento del código al realizar una lectura, etc.).

Acto seguido se probaron los aspectos internos de la aplicación mediante pruebas de caja blanca: revisión del funcionamiento de un proyecto a partir de su lógica. De este modo, y dado que Android está muy orientado a la interfaz gráfica, se realizaron pruebas mediante el uso de todas las funciones disponibles, realizando las acciones necesarias para recorrer todos los flujos posibles de la aplicación.

En aquellos apartados donde se consideró no cumplir los requisitos de “código limpio” [26], se procedió (una vez probada funcionalmente), a refactorizar el código. Esto significa que se ordenó, modularizó, y mejoró su forma visual para cumplir el nivel de limpieza buscado.

7.1.2 Caja negra: componentes

Una vez probados los caminos internos, se siguió con la realización de pruebas de caja negra, aquellas que comprueban un resultado sin tener en cuenta el procedimiento por el cual se llevaron a cabo. Esto indica que se provee una entrada, se realizan ciertos procedimientos internos (no relevantes) y se obtiene una salida, siendo ésta esperada, no esperada o incorrecta.

De esta forma las pruebas de caja negra vienen dadas por las funcionalidades disponibles en los botones de la interfaz gráfica:

- Seleccionado un término de una expresión, se permuta una posición a la derecha o izquierda.
- Habiendo seleccionado dos elementos de una expresión, se agrupan los términos comprendidos entre ellos, extremos inclusive, generando paréntesis alrededor. En

caso de no poder realizarse, se informa al usuario de la imposibilidad de realizar la acción.

- Se puede extraer un factor común de expresiones simples (elementos únicos) y complejas (multiplicaciones) contenidas en una suma, resultando en un elemento que multiplica a una suma de los términos no comunes.
- Se realiza correctamente el cambio de signo de una expresión al cambiar de lado de una igualdad.
- Se permite realizar la propiedad distributiva sobre una expresión multiplicación con un término suma.
- Se eliminan los ceros y unos de las sumas y multiplicaciones, respectivamente. También se anulan los términos opuestos en sumas (signo negativo) y en multiplicaciones (inversos).
- Se realizan operaciones matemáticas que incluyen varios operadores, resultando en un número final.
- Se guardan todos los pasos realizados sobre una expresión, permitiendo acceder a ellos desde un historial de expresiones.

7.2 Pruebas de interfaz

Una vez con la aplicación probada, se pasó a realizar una serie de pruebas de manera informal sobre la interfaz por medio de cinco usuarios externos, con el fin de controlar la facilidad de uso del proyecto y comprobar posibles funcionamientos atípicos debido a acciones humanas imprevistas.

El objetivo de estas pruebas es verificar los siguientes puntos:

1. La interfaz es fácil de usar: los usuarios deben poder controlar la aplicación de manera rápida y sencilla.
2. La aplicación no permite acciones incorrectas debido a la aplicación de una acción de manera inadecuada.
3. La aplicación responde bien a cambios de pantalla en dispositivos de distinto tamaño.
4. La aplicación es compatible con versiones anteriores de Android, hasta un mínimo de versión 4.2 (*JellyBean*).

Una vez finalizada la prueba, los usuarios dieron sus opiniones sobre la aplicación, explicando las ventajas y desventajas de la aplicación mostrada. Dichas conclusiones se recogen en la tabla 7.1:

Ventajas	Desventajas
✓ Fácil de usar	✗ Limitación en colores
✓ Manipulación intuitiva	✗ Si una expresión es larga, puede salirse de la pantalla
✓ Apta para varios tamaños de pantalla	✗ Pocas operaciones, sólo expresiones lineales
✓ Detección automática de aplicación de propiedades sobre la selección	✗ Botón operar no deja elegir tipo de operación

Tabla 7.1: Resultados de pruebas de interfaz

Si bien es cierto que las ventajas muestran un éxito de la aplicación, las desventajas dan una idea de hacia dónde podría evolucionar el proyecto en un futuro, siguiendo dichas líneas de pensamiento como base para desarrollar más funcionalidades.

7.3 Resultados

A modo de resultado, se muestran aquí dos capturas de pantalla de la aplicación (figuras 7.1 y 7.2), para servir de primera impresión al lector. Dichas imágenes muestran dos funcionalidades clave del proyecto: manipulación de expresiones en la pantalla *Tablero* y el historial de cambios sobre esa expresión en la pantalla *Historial de Expresiones*.

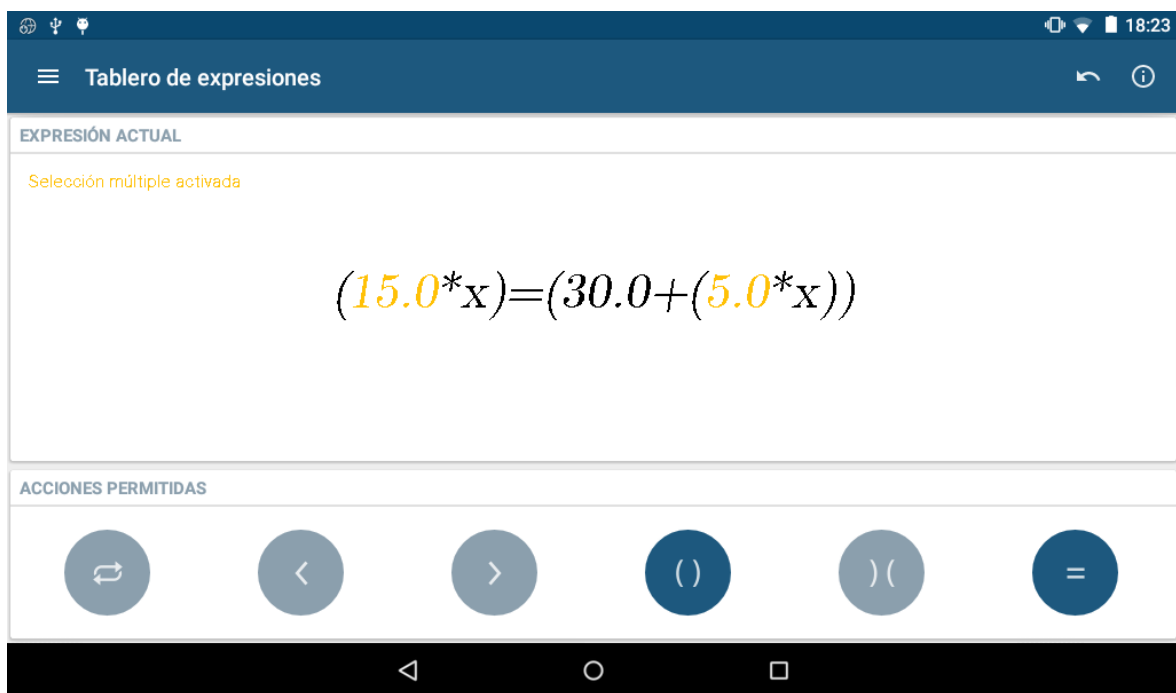


Figura 7.1: Pantalla *Tablero* en la aplicación final

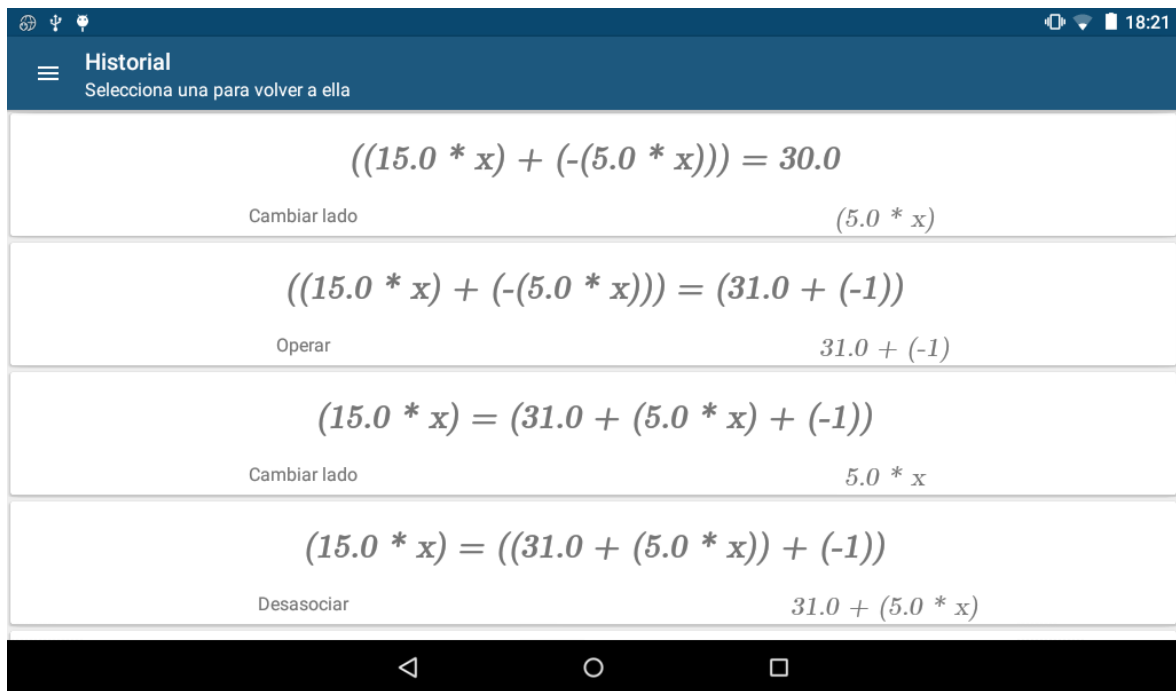


Figura 7.2: Pantalla *Historial de expresiones* en la aplicación final

Un ejemplo más extenso y desarrollado puede encontrarse en el Manual de Usuario, contenido en el Anexo A.

8 Conclusiones y trabajo futuro

8.1 Conclusiones

Tras el desarrollo de la interfaz para *Chalkpy* se ha obtenido una aplicación en Android que permite la manipulación de expresiones matemáticas de primer orden. Se realizan las acciones conmutar, asociar, desasociar, propiedad distributiva, sacar factor común, cambiar de lado de la ecuación y operar, cumpliendo los objetivos que se intentaban alcanzar.

Permitiendo controlar una expresión algebraica de manera intuitiva y dinámica, se puede utilizar como herramienta de apoyo al docente en explicaciones que incluyan contenido matemático. Al mismo tiempo, al estar integrada en un dispositivo móvil Android permite su uso tanto en entornos académicos como informales, sin necesidad de herramientas externas.

Está construida en Android bajo un patrón Modelo-Vista-Controlador, utilizando como datos las expresiones que provee un CAS externo generado por la herramienta *G-CAS*. Mediante las Actividades y Fragmentos de Android se controla el flujo de datos, permitiendo una comunicación entre la capa de datos y la interfaz gráfica. Ésta última viene representada por el tablero de expresiones, que permite la selección de términos de una expresión y la manipulación por medio de acciones que varían en función de la selección actual.

Se puede considerar un éxito dado que se han alcanzado los objetivos propuestos, permitiendo una manipulación rápida, sencilla y dinámica de las expresiones matemáticas en la aplicación.

8.2 Trabajo futuro y posibles mejoras

Los objetivos de la aplicación pueden considerarse cumplidos con éxito, sin embargo, existen líneas de trabajo que permitirían la extensión del proyecto en un futuro. Se tiene pensado continuar con el desarrollo de esta aplicación, incluso se valora la opción de ponerla de manera gratuita en la plataforma de aplicaciones de Google, *Play Store* [27].

Aunque la realización de manipulaciones mediante botones permite una rápida adaptación del usuario a la interfaz, en un futuro podría mapearse dichas acciones a gestos del usuario mediante la tecnología *Drag and Drop*, que permite el arrastre de elementos por la pantalla y, tras su colocación en otro lugar, realizar acciones determinadas. Esto permitiría una mejor experiencia de usuario, ya que se controla la ecuación mediante los dedos y de manera más natural.

En lo relativo a la integración con el CAS externo usado, podría incluirse la herramienta *G-CAS* en la aplicación, permitiendo definir en la propia aplicación las reducciones y equivalencias que el usuario quisiera añadir.

Así mismo, y haciendo referencia a las desventajas apuntadas por los usuarios que realizaron las pruebas, sería una mejora significativa la extensión de las operaciones a problemas no lineales, incluyendo, por ejemplo, potencias o derivadas.

Referencias

- [1] G. S. Pantoja, Herramienta para mejorar la presentación de contenidos matemáticos en el aula (TFG_UAM), Madrid, 2016.
- [2] L. S. Valderrama, «Desarrollo de un Motor de Matemática Simbólica para la Herramienta Chalkpy (TFG_UAM),» Madrid, 2016.
- [3] R. d. Blas, «Álgebra Computacional (trabajo para la asignatura Historia de las Matemáticas),» 2015. [En línea]. Available: <https://dl.dropboxusercontent.com/u/5522764/THM.RdeBlas.CompAlg.pdf>. [Último acceso: 7 Junio 2016].
- [4] A. Artacho, «Matemáticas Cercanas,» [En línea]. Available: <http://matematicascercanas.com/aplicaciones-matematicas-para-android/>. [Último acceso: 7 Junio 2016].
- [5] Wolfram, «Wolfram Alpha,» [En línea]. Available: <https://www.wolframalpha.com/examples/Math.html>.
- [6] Photomath, Inc., «Photomath - Cámara calculadora,» [En línea]. Available: <https://play.google.com/store/apps/details?id=com.microblink.photomath&hl=es>. [Último acceso: 7 Junio 2016].
- [7] Wolfram Group, «Wolfram Alpha - Google Play,» [En línea]. Available: <https://play.google.com/store/apps/details?id=com.wolfram.android.alpha&hl=es>. [Último acceso: 7 Junio 2016].
- [8] Mathination, «Mathination,» [En línea]. Available: <http://www.mathination.com/>. [Último acceso: 7 Junio 2016].
- [9] Regular Berry, «Algebra Touch,» [En línea]. Available: <http://www.regularberry.com/>. [Último acceso: 7 Junio 2016].
- [10] I. Sommerille, Ingeniería del Software, Pearson Educación, 2005.
- [11] Android, «Android,» [En línea]. Available: https://www.android.com/intl/es_es/. [Último acceso: 7 Junio 2016].
- [12] Oracle, «Java,» [En línea]. Available: <https://www.java.com/es/>. [Último acceso: 7 Junio 2016].
- [13] W3C, «XML,» [En línea]. Available: <http://www.w3schools.com/xml/default.asp>.
- [14] LaTeX, «LaTeX,» [En línea]. Available: <https://www.latex-project.org/>.
- [15] Android, «Android Studio,» [En línea]. Available: <https://developer.android.com/studio/index.html?hl=es>. [Último acceso: 7 Junio 2016].
- [16] GitHub, «GitHub,» [En línea]. Available: <https://github.com/>. [Último acceso: 7 Junio 2016].
- [17] Atlassian, «SourceTree,» [En línea]. Available: <https://www.sourcetreeapp.com/>. [Último acceso: 7 Junio 2016].
- [18] syntevo, «SmartGit,» [En línea]. Available: <http://www.syntevo.com/smartgit/>. [Último acceso: 7 Junio 2016].
- [19] JGraph, «draw.io,» [En línea]. Available: <https://www.draw.io/>. [Último acceso: 7 Junio 2016].
- [20] balsamiq, «Balsamiq Mockups,» [En línea]. Available:

- <https://balsamiq.com/products/mockups/>. [Último acceso: 7 Junio 2016].
- [21] Gimp, «Gimp,» [En línea]. Available: <http://www.gimp.org.es/>. [Último acceso: 7 Junio 2016].
- [22] Wikipedia, «Model-View-Controller,» [En línea]. Available: <https://en.wikipedia.org/wiki/Model-view-controller>. [Último acceso: 7 Julio 2016].
- [23] R. d. Blas, «AndroidCAS - Github,» [En línea]. Available: <https://github.com/rodrixan/AndroidCAS>. [Último acceso: 17 Junio 2016].
- [24] Oracle, «JAR (file format),» [En línea]. Available: <https://docs.oracle.com/javase/tutorial/deployment/jar/index.html>. [Último acceso: 7 Junio 2016].
- [25] Wikipedia, «Singleton Pattern,» [En línea]. Available: https://en.wikipedia.org/wiki/Singleton_pattern. [Último acceso: 17 Junio 2016].
- [26] R. C. Martin, Clean code: A Handbook of Agile Software Craftsmanship, Prentice Hall, 2008.
- [27] Google, «Play Store Apps,» [En línea]. Available: <https://play.google.com/store/apps>. [Último acceso: 17 Junio 2016].

Glosario

CAS	<i>Computer Algebra System</i> , Sistema Algebraico Computacional. Herramienta que permite realizar manipulaciones sobre expresiones matemáticas mediante el uso de reducciones o reglas de equivalencia.
Smartphone	Teléfono inteligente, móvil.
Tablet	Tableta, dispositivo móvil similar a un Smartphone, con un mayor tamaño.
Término	Cualquier expresión matemática que se encuentre en otra expresión contenedora.
API	<i>Application Programming Interface</i> , Interfaz de Programación de Aplicaciones. Conjunto de funcionalidades que ofrece una cierta biblioteca o librería de software para ser usada como una capa de abstracción.
Framework	Conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar (Wikipedia).
MVC	<i>Model-View-Controller</i> . Patrón de arquitectura Modelo-Vista-Controlador.
Paquete (java)	Unidad contenedora de clases java que cumple una misma funcionalidad, permitiendo una estructura jerárquica.
Widget (Android)	Elemento visual de Android, normalmente incluido dentro de un contenedor de los mismos (<i>layout</i>).
Listener (Android)	Función llamada para gestionar un evento de entrada.

Anexos

Anexo A. Manual de uso

Este anexo contiene los pasos necesarios para utilizar la aplicación una vez se encuentra en el dispositivo móvil. Todas las capturas de pantalla del mismo se han realizado sobre una Tablet *Acer Iconia One 7*, con el fin de aprovechar su mayor tamaño de pantalla.

A modo de ejemplo se resuelve una ecuación, ilustrando las diversas opciones que la aplicación ofrece.

Inicio de la aplicación y pantalla de ayuda

Una vez se dispone de la aplicación en el dispositivo, para lanzarla basta con ir al menú principal y pulsar el icono de la aplicación (figura A.1)

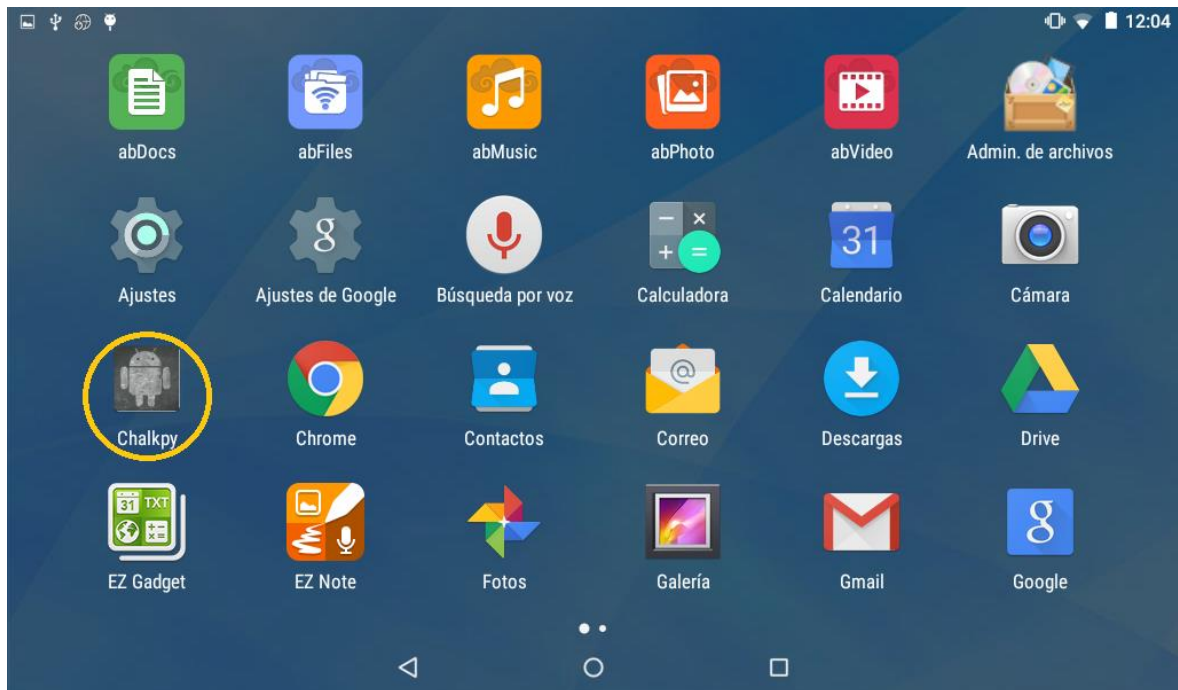


Figura A.1: Lanzamiento de la aplicación

Una vez lanzada, la aplicación cargará una expresión de ejemplo de las que se encuentran ya cargadas en su interior (figura A.2):

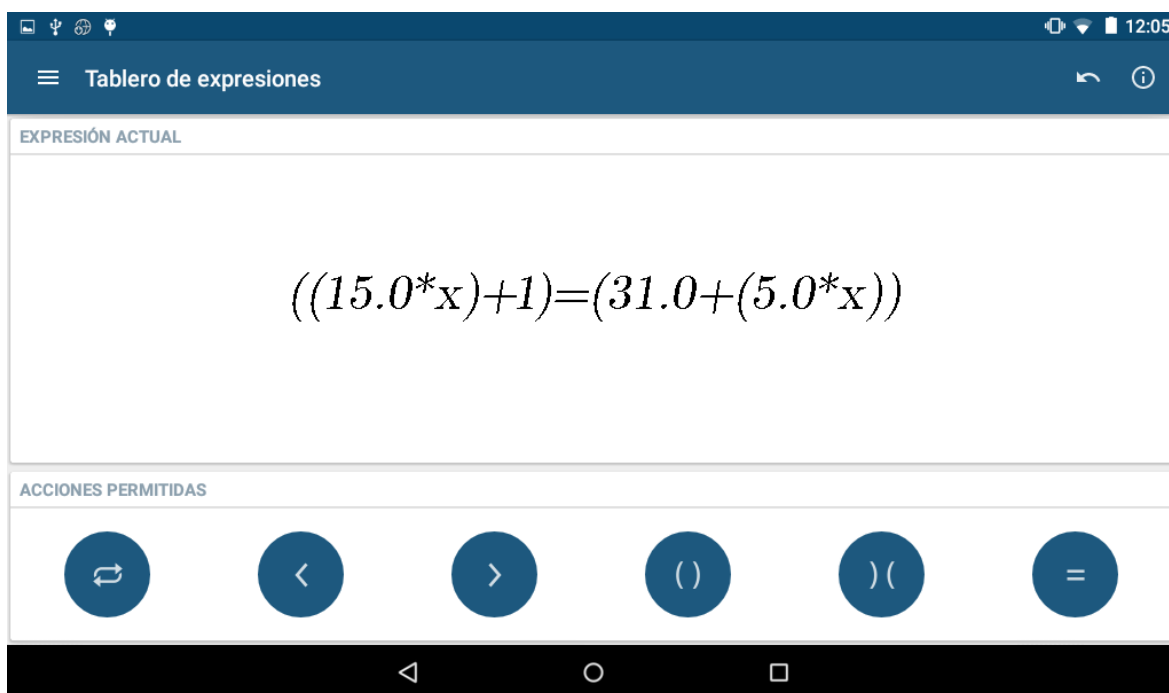


Figura A.2: Pantalla principal

Pulsando en el icono de información (esquina superior derecha, con forma de globo de información), se muestra la pantalla de ayuda, que contiene información relativa al uso de la pantalla *Tablero* (figura A.3):

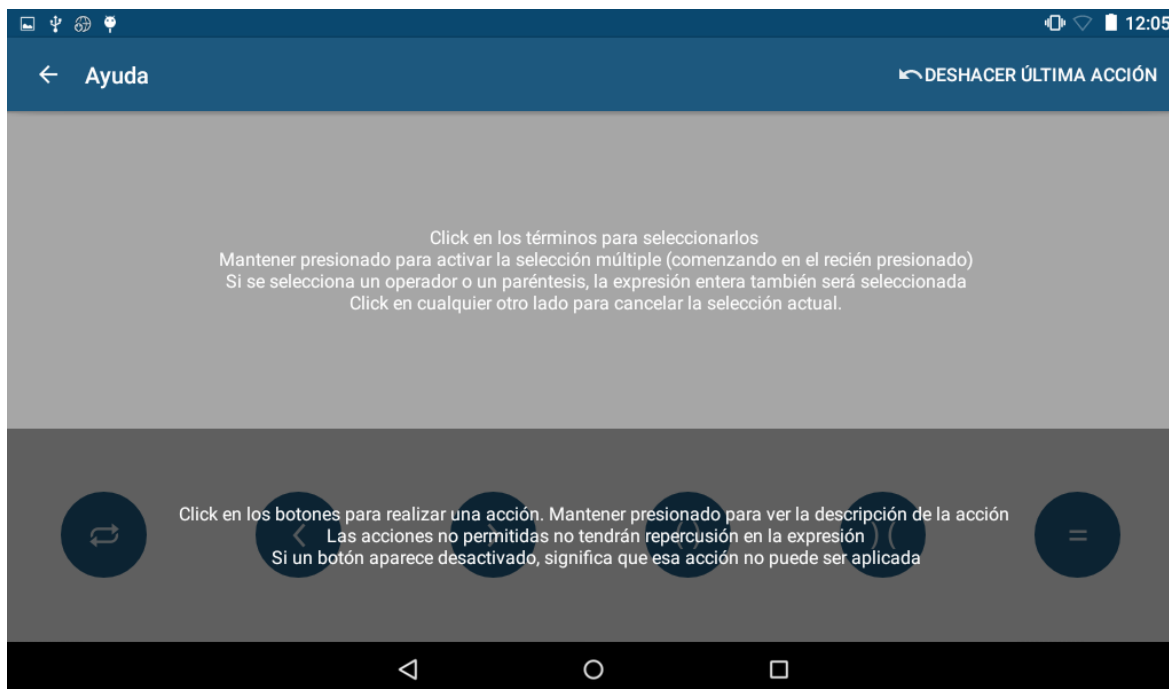


Figura A.3: Pantalla de ayuda

Para volver a la pantalla anterior puede pulsarse la flecha situada en la esquina superior izquierda o el botón de atrás del dispositivo. En ambos casos se volverá a la pantalla principal (*Tablero*).

Navegación y ajustes

Para navegar por la aplicación puede pulsarse el icono con tres líneas horizontales de la esquina superior izquierda, o deslizar el dedo desde el borde lateral izquierdo de la pantalla. Esto desplegará un menú con las pantallas de la aplicación (figura A.4):

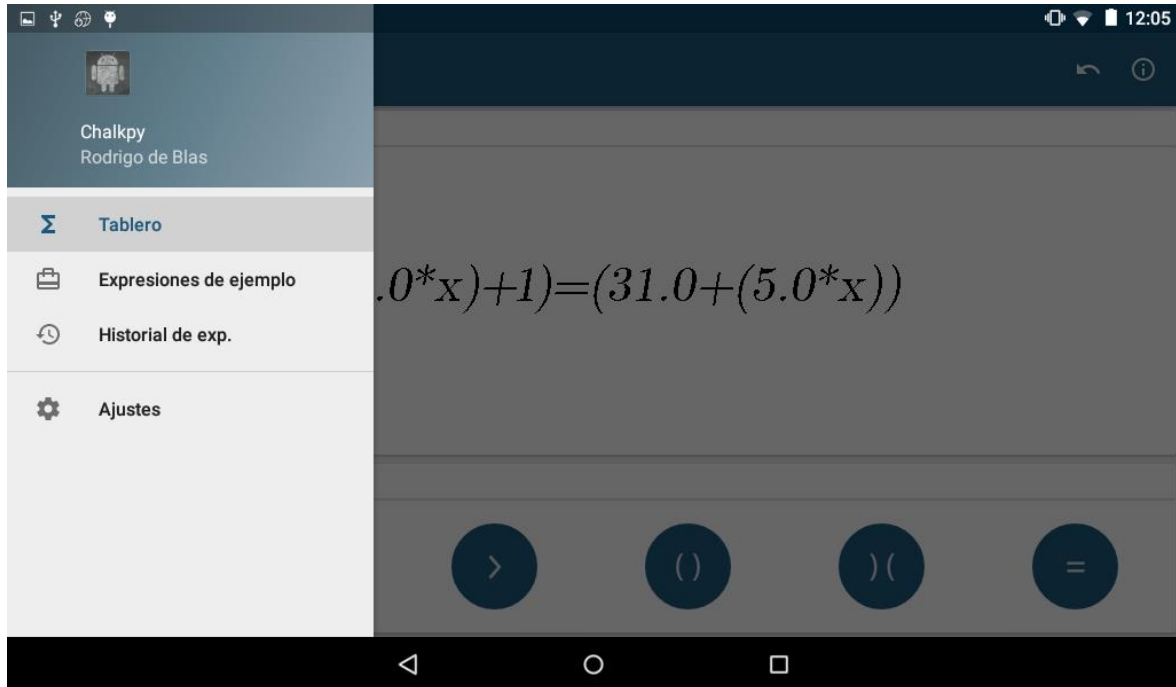


Figura A.4: Menú desplegable de navegación

Para cambiar las preferencias disponibles de la aplicación, se debe pulsar en la entrada “Ajustes”. Esta acción cambiará la pantalla a la selección de ajustes (figura A.5):

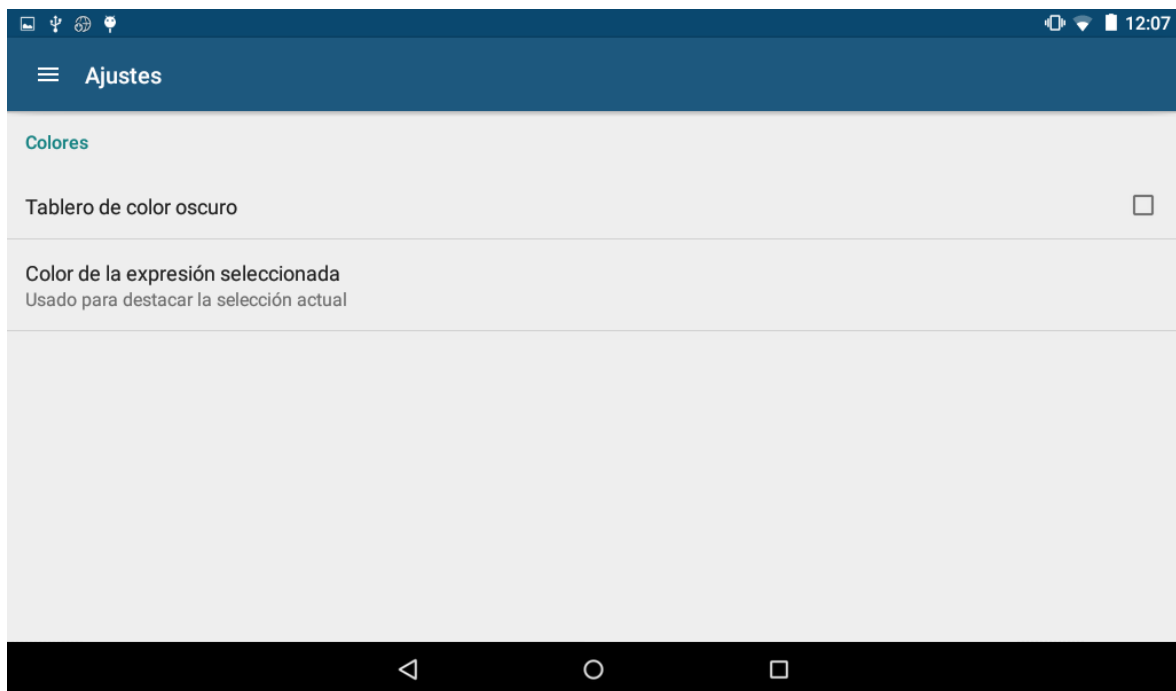


Figura A.5: Ajustes de la aplicación

La primera preferencia permite cambiar el color de fondo del tablero de expresiones, pudiendo elegir entre claro (blanco) y oscuro (verde pizarra). La segunda selecciona el color de la expresión que el usuario ha seleccionado. Por ejemplo, para este manual se deja el fondo claro y se marca “Azul” como color de la selección (figura A.6):

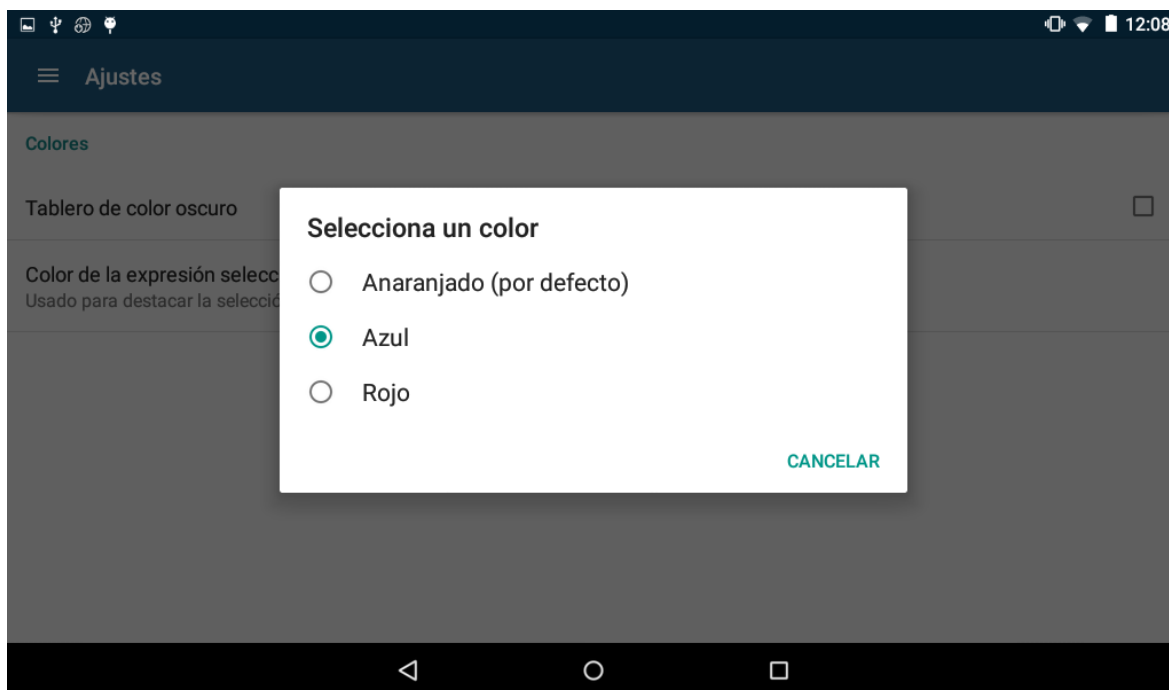


Figura A.6: Ajustes de color de la selección de usuario

Selección de una expresión global

Para cargar una expresión predefinida, debe desplegarse el menú de navegación y seleccionarse la pantalla “*Expresiones de Ejemplo*”. Ésta contiene una lista de expresiones seleccionables como globales para trabajar con ellas (figura A.7):

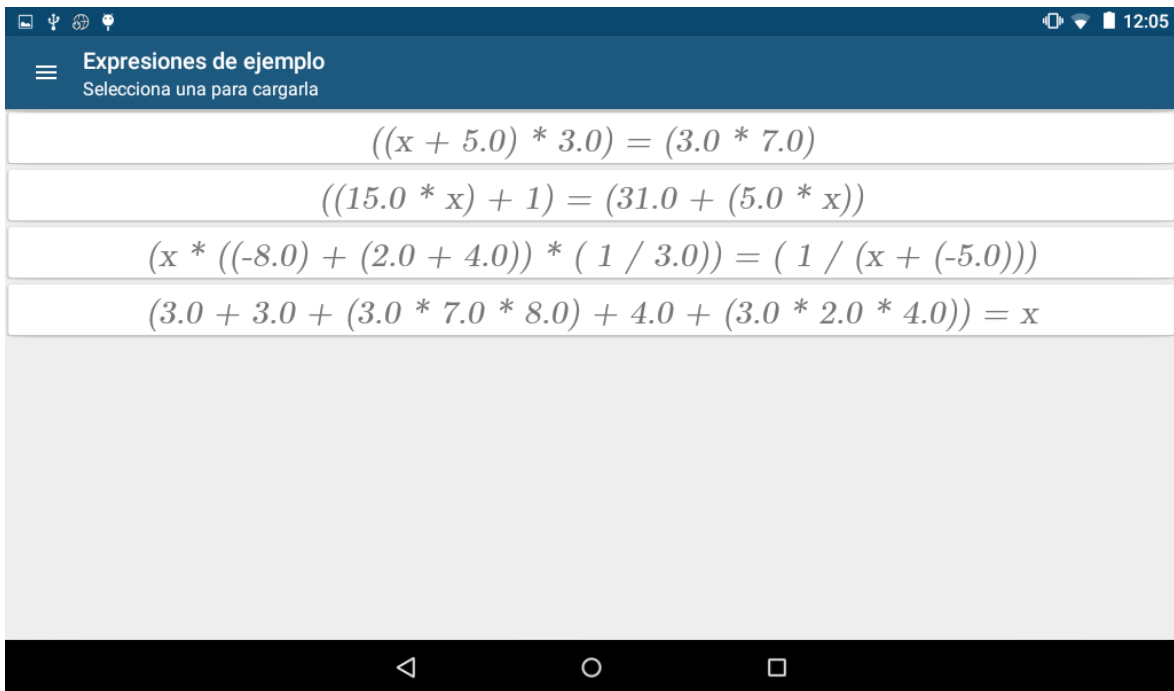


Figura A.7: Expresiones de ejemplo predefinidas

Pulsando cualquiera de ellas se cargará en el motor algebraico interno y se irá a la pantalla *Tablero*, donde el usuario puede realizar las manipulaciones que desee sobre ella (figura A.8):

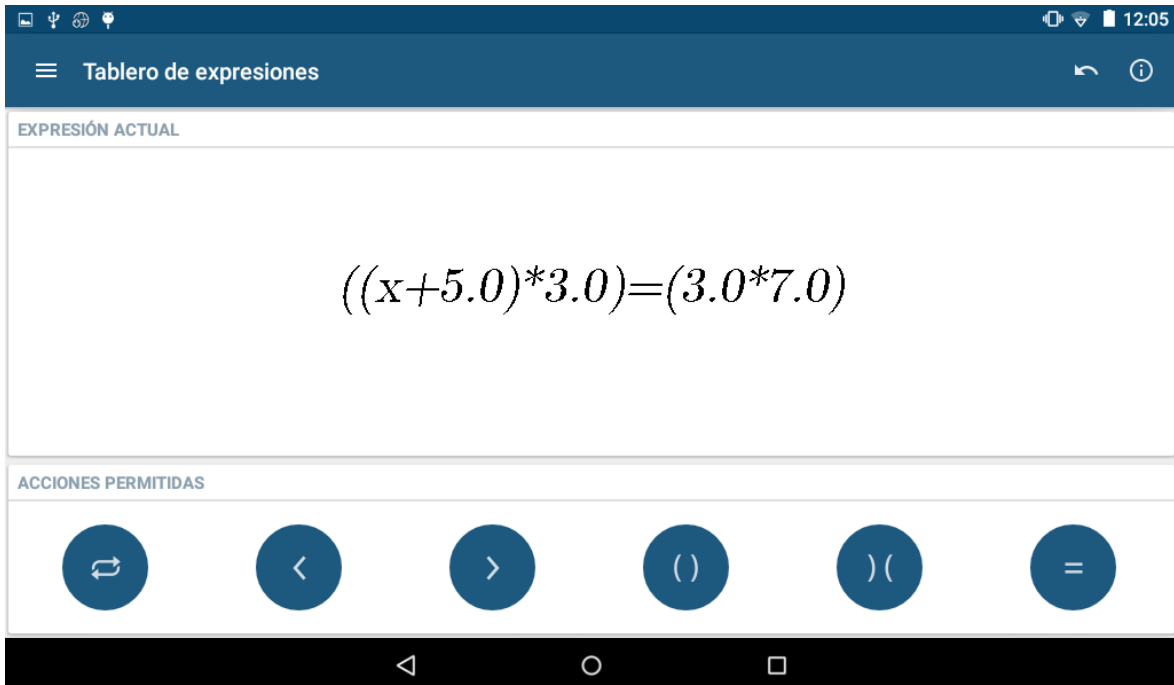


Figura A.8: Expresión seleccionada para su manipulación

Manipulación de expresiones: ejemplo de uso

Una vez cargada una expresión, el usuario puede realizar selecciones de sus términos de manera única o múltiple y ejecutar acciones sobre ellas mediante la pulsación de los botones en la parte inferior. Para ello debe haber realizado al menos una selección.

Como ejemplo ilustrativo, se detallan los pasos seguidos para resolver la ecuación cargada en el apartado anterior. Nótese que durante el desarrollo, las acciones disponibles van variando, indicando cuáles pueden realizarse mediante el cambio de color de los botones asociados. Si en cualquier momento se desea deshacer la última transformación, basta pulsar el botón con una flecha circular en la esquina superior izquierda, a la derecha del icono de ayuda.

Primero se aplicará la propiedad distributiva al término izquierdo de la ecuación. Para ello se seleccionan los términos involucrados mediante selección múltiple, manteniendo pulsado el primero para activarla y luego seleccionando con una pulsación el otro (figura A.9).

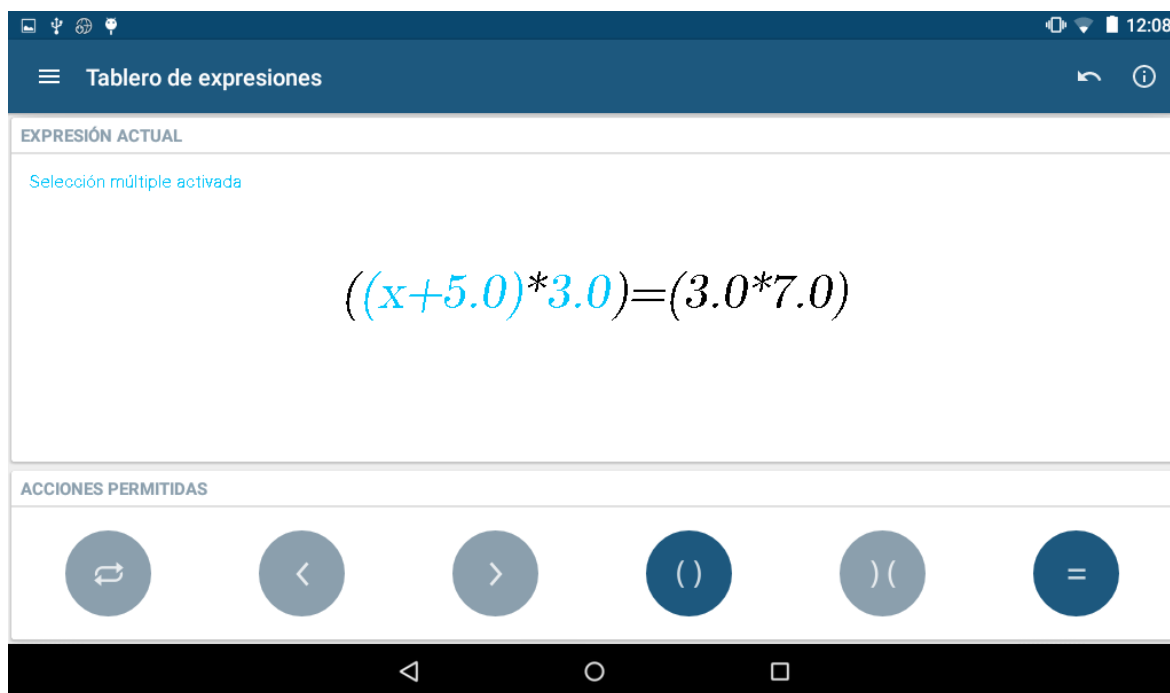


Figura A.9: Selección múltiple de términos para propiedad distributiva

Una vez realizada la selección, pulsamos el botón “Operar”, localizado en última posición (su icono es un símbolo ‘=’). La aplicación detectará que debe aplicarse la propiedad distributiva y operará con la expresión, resultando en una nueva expresión (figura A.10):

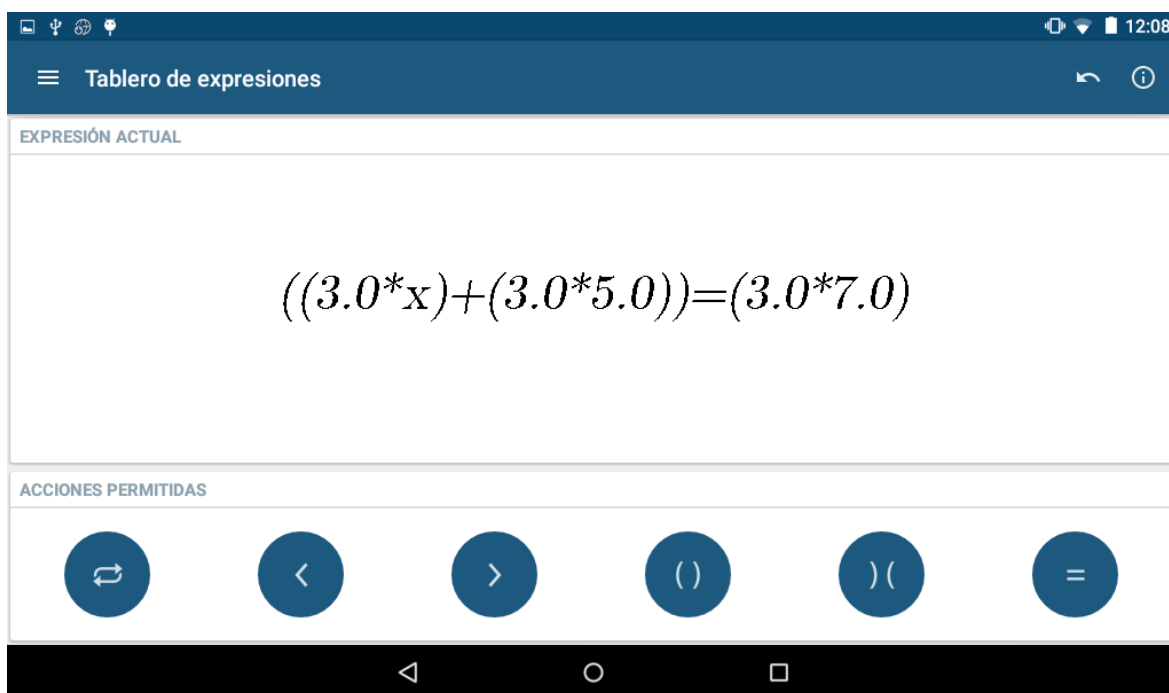


Figura A.10: Expresión resultante de aplicar la prop. distributiva

Un detalle menor: la selección múltiple queda activa tras la realización de la acción, con el fin de poder seguir trabajando con múltiples términos. Es necesario pulsar fuera de la expresión para desactivarla.

El siguiente paso para resolver la ecuación consiste en operar y cambiar de lado de la ecuación el producto $3*5$. Para ello, primero se pulsa sobre el operador $*$ que los une o uno de los paréntesis que envuelve el producto y se presiona el botón “Operar” (figura A.11):

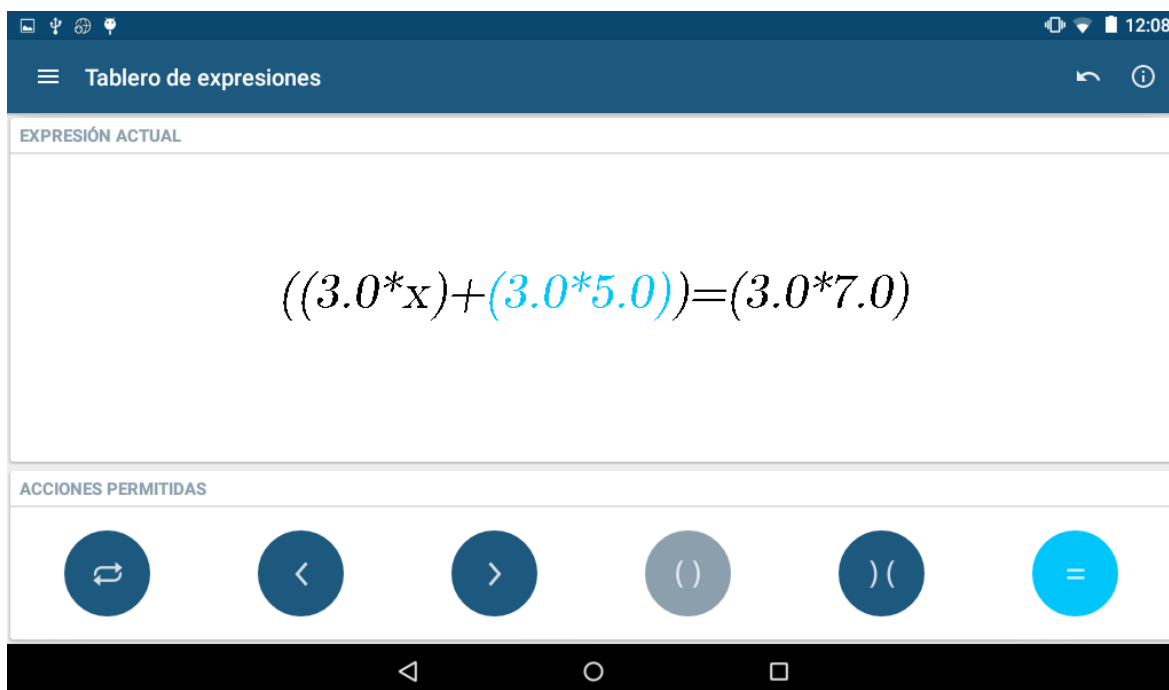


Figura A.11: Selección simple de una expresión para operar

El resultado sustituirá el producto en la expresión. Seleccionando el nuevo término, se pulsa sobre el botón “*Cambiar lado*”, en primera posición (figura A.12):

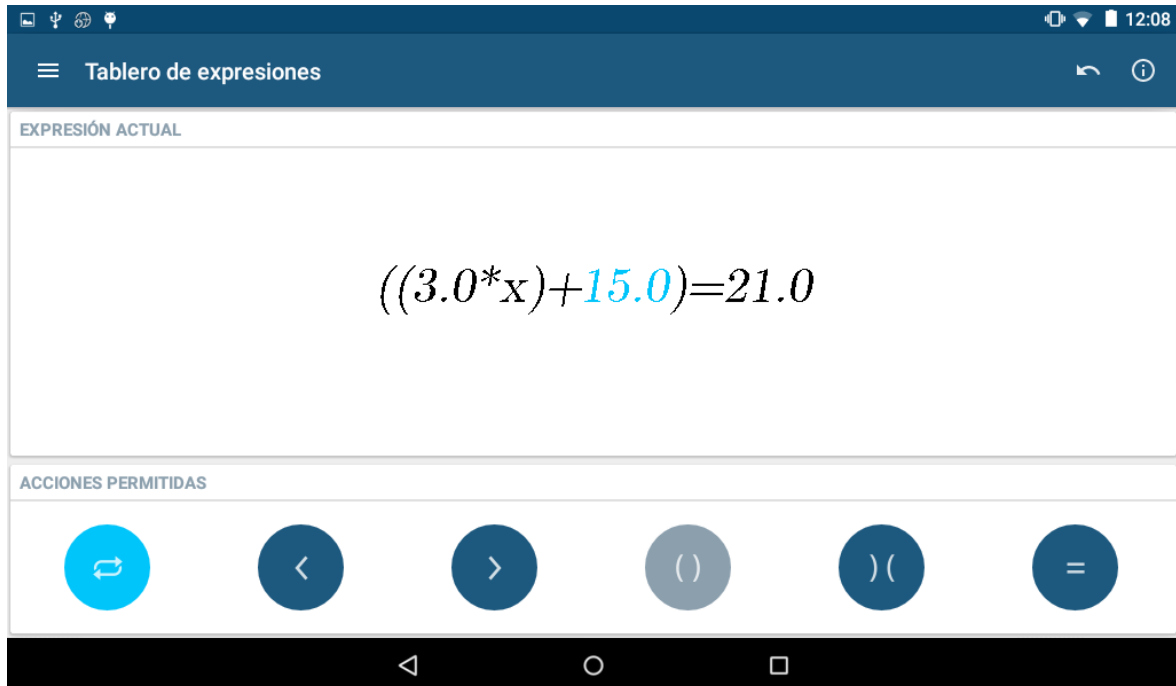


Figura A.12: Selección para cambio de lado en una ecuación

Esta acción desemboca en un cambio de signo para el término, puesto que es un elemento de una suma, apareciendo en el lado contrario de la igualdad (figura A.13):

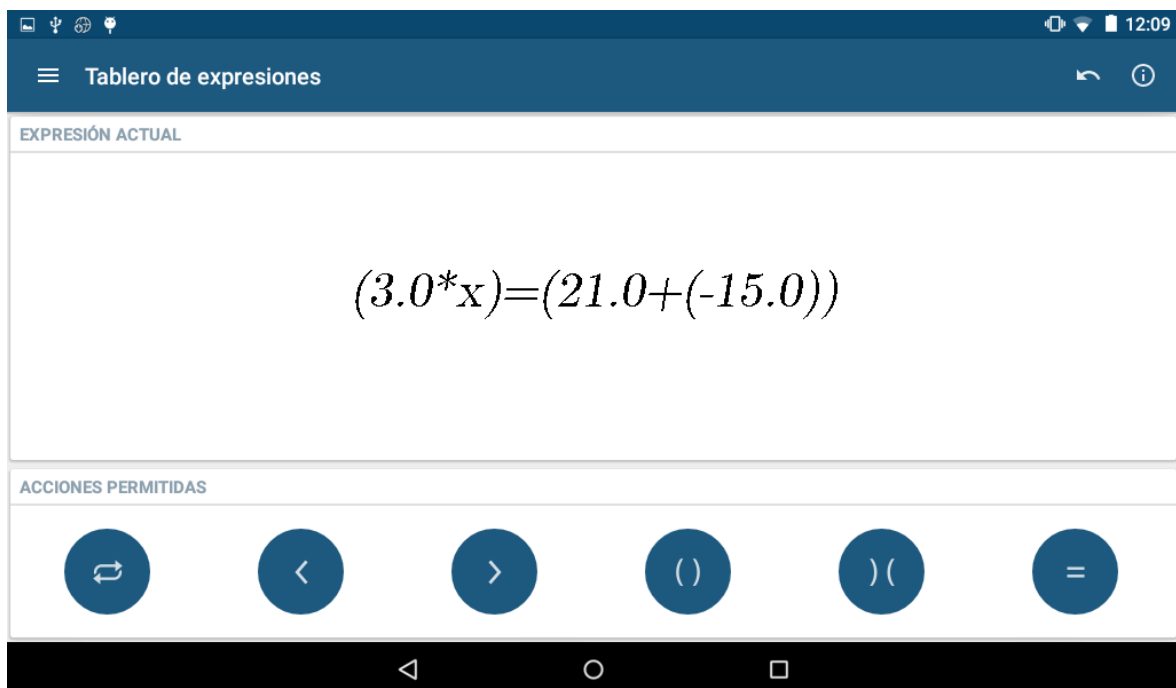


Figura A.13: Resultado del cambio de lado en una expresión

A continuación, se opera la suma del lado derecho de la ecuación, repitiendo el proceso visto anteriormente. El resultado sustituirá la suma en la igualdad. Una vez realizada esa operación, se debe cambiar el 3 que multiplica a la x de lado, por lo que se vuelve a realizar el procedimiento para el cambio de lado. En este caso, al ser un producto, el término cambia por su inverso (1/3).

Sobre la nueva ecuación, y para colocar el número fraccionario en primera posición, se selecciona éste y se pulsa el botón de “Conmutar a la derecha” (figura A.14):

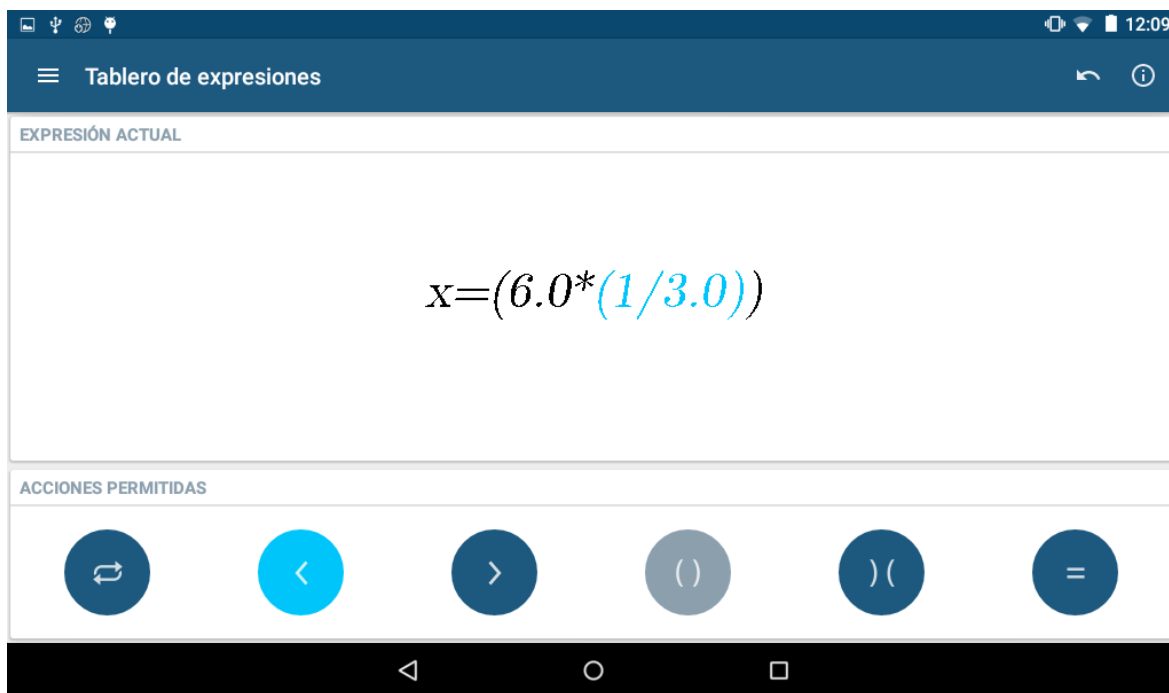


Figura A.14: Selección para aplicar la prop. conmutativa

El resultado de conmutar tiene a 1/3 como primer término (figura A.15). La propiedad conmutativa en la aplicación permite permutar un término de una suma o una multiplicación una posición a la derecha/izquierda. Si no fuera posible permutar una posición más en una dirección, la aplicación avisará de esta situación, sin modificar la expresión actual.

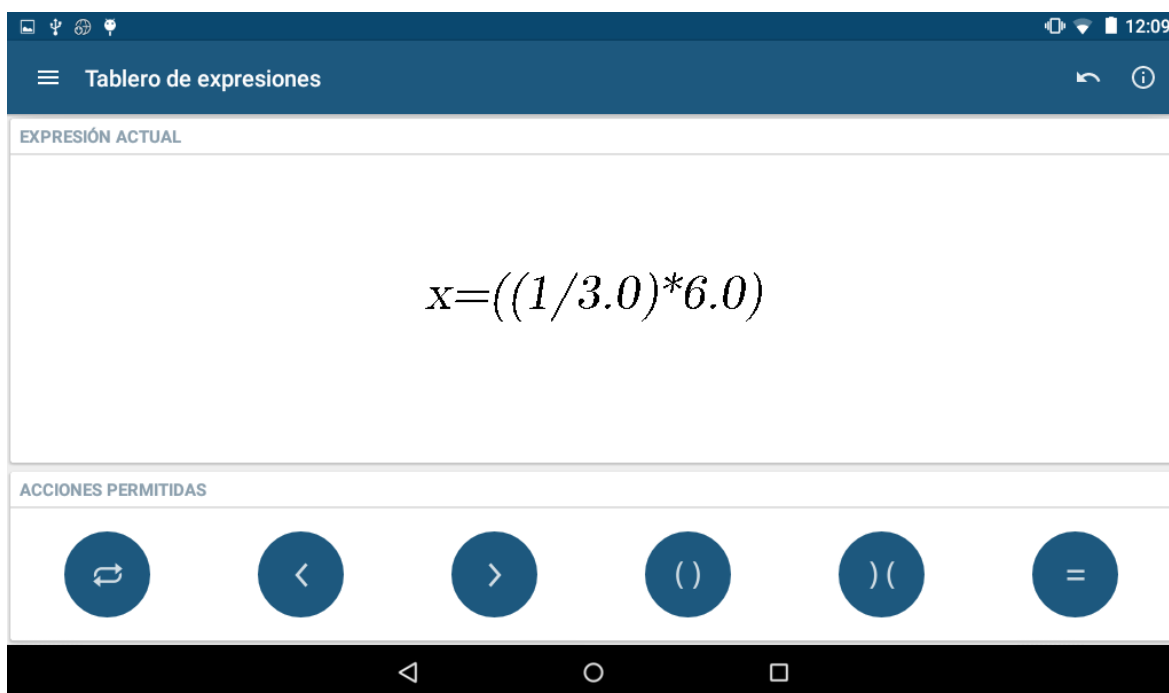


Figura A.15: Resultado de aplicar la prop. conmutativa a una expresión

El último paso se reduce a operar el lado derecho de la igualdad, despejando el valor de la incógnita x y resolviendo así la ecuación (figura A.16):

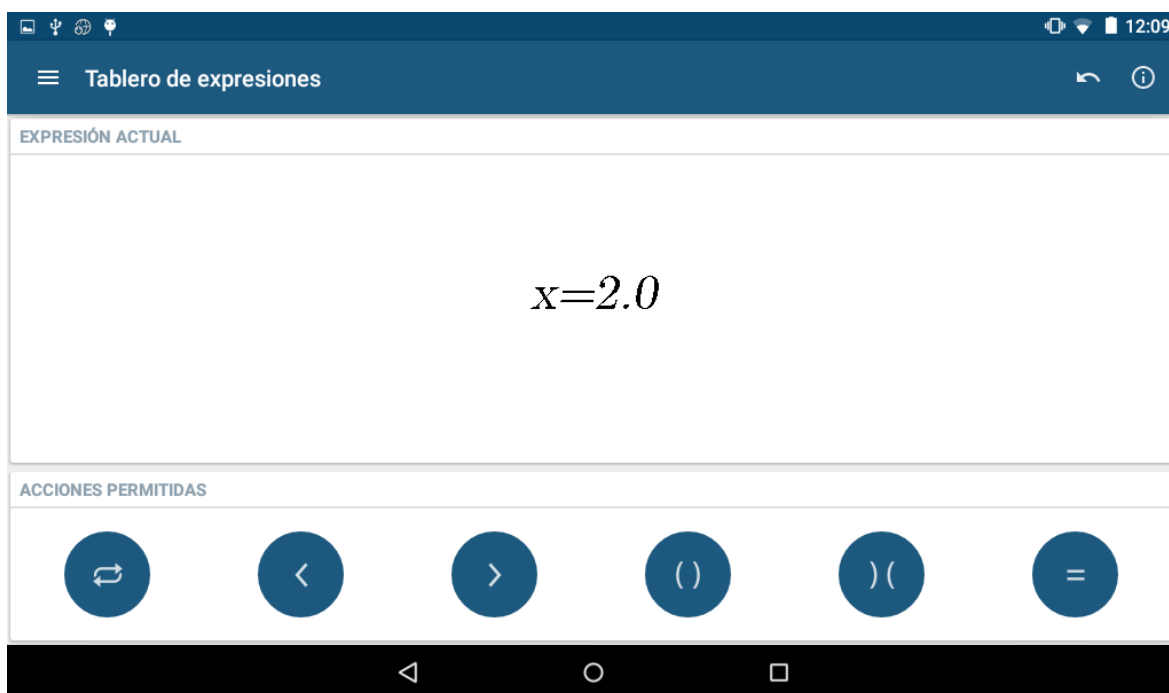


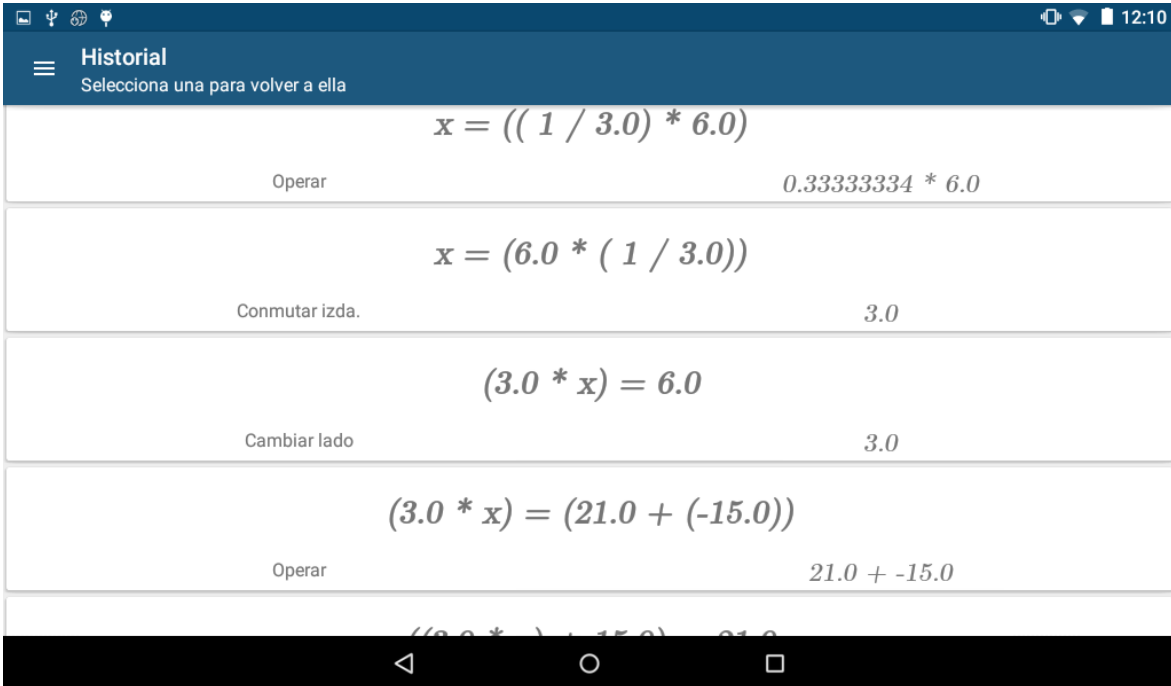
Figura A.16: Solución de la ecuación

Historial de expresiones

Es posible consultar todas las manipulaciones realizadas hasta el momento sobre las expresiones utilizadas. Para ello, se selecciona la opción “*Historial de expresiones*” del menú desplegable.

La pantalla mostrará una lista con las expresiones sobre las que se realizaron las acciones, las acciones ejecutadas y la selección que se usó para el desarrollo de la manipulación.

En el caso del ejemplo realizado, se siguen perfectamente los pasos realizados en orden inverso, de más reciente a más antiguo (figuras A.17 y A.18):



Historial	
Selecciona una para volver a ella	
$x = ((1 / 3.0) * 6.0)$	
Operar	$0.33333334 * 6.0$
$x = (6.0 * (1 / 3.0))$	
Conmutar izda.	3.0
$(3.0 * x) = 6.0$	
Cambiar lado	3.0
$(3.0 * x) = (21.0 + (-15.0))$	
Operar	$21.0 + -15.0$
$(3.0 * x) = (21.0 + (-15.0))$	

Figura A.17: Historial de expresiones (I)

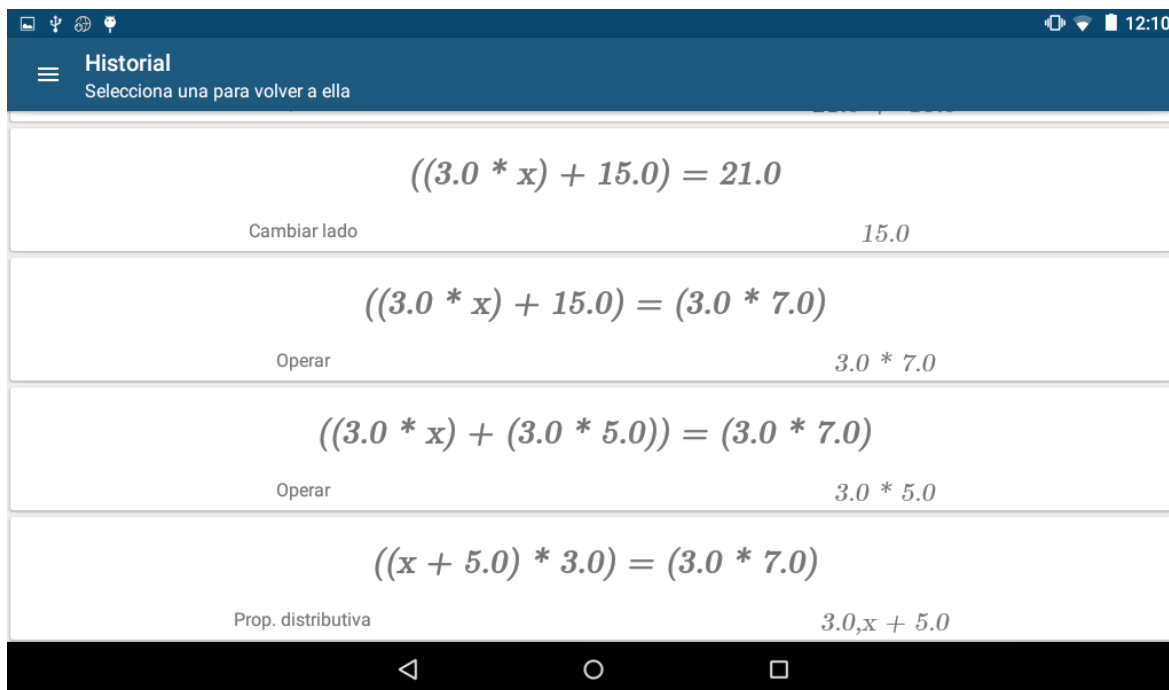


Figura A.18: Historial de expresiones (II)

Tablero de color oscuro

A modo de final, y para ilustrar el caso específico, se muestra la pantalla *Tablero* con la opción “*Tablero de color oscuro*” activada (figura A.19):

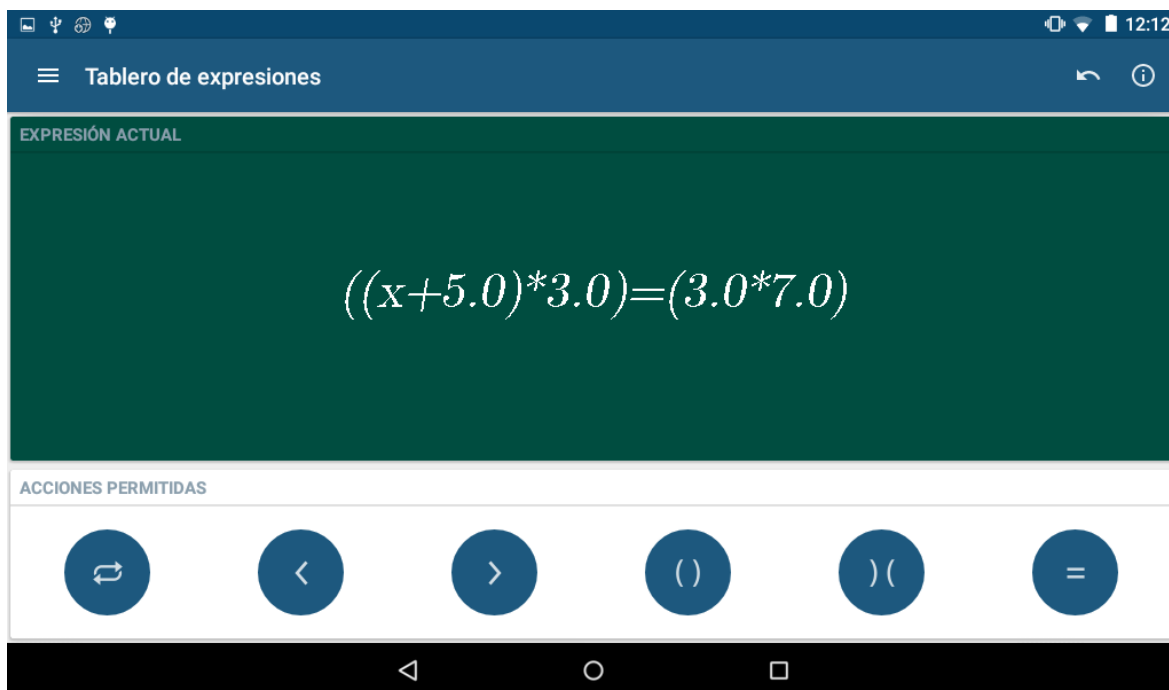


Figura A.19: Pantalla *Tablero* con color de fondo oscuro

Anexo B. Diagramas de clases

Aquí se muestran los diagramas de clases completos de la aplicación, agrupados según la arquitectura MVC. En las clases se muestran los métodos relevantes para la implementación con el fin de condensar la información útil.

A fin de ilustrar la conexión entre los elementos del proyecto, en cada diagrama de capa se muestran las clases de las demás capas que son usadas en la primera.

B.1 Modelo

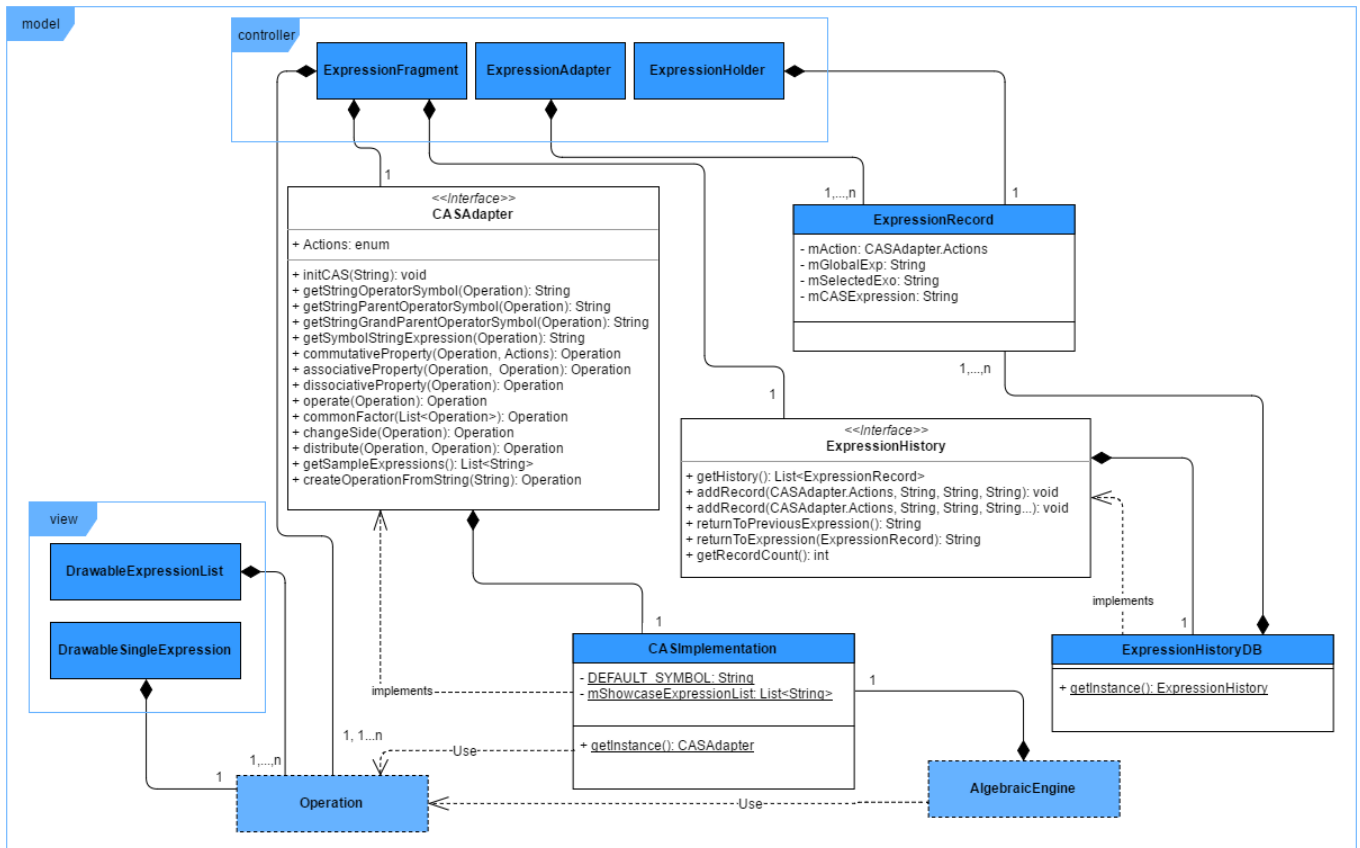


Figura B.1: Diagrama de clases de la capa Modelo

B.2 Vista

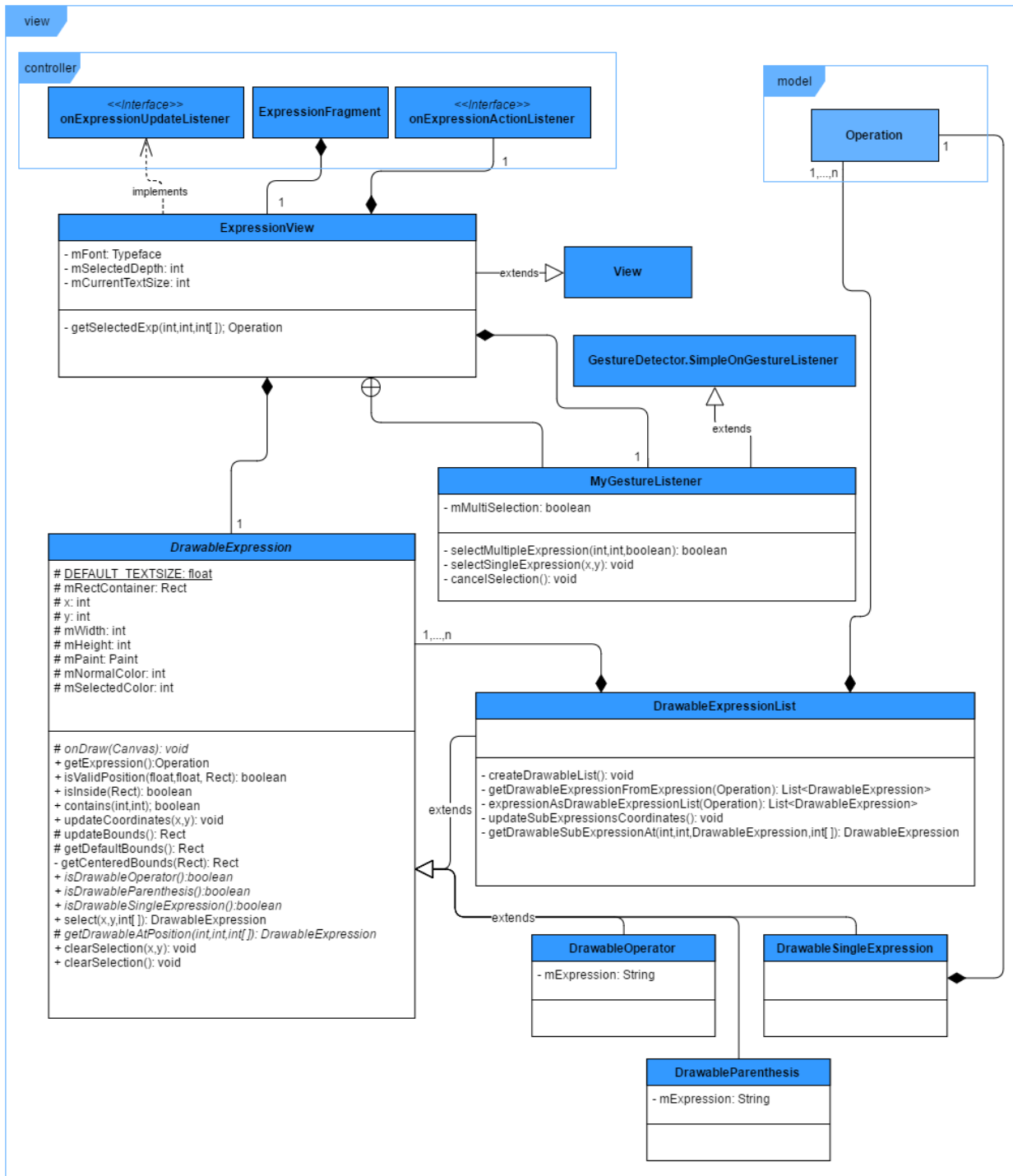


Figura B.2: Diagrama de clases de la capa Vista

B.3 Controlador

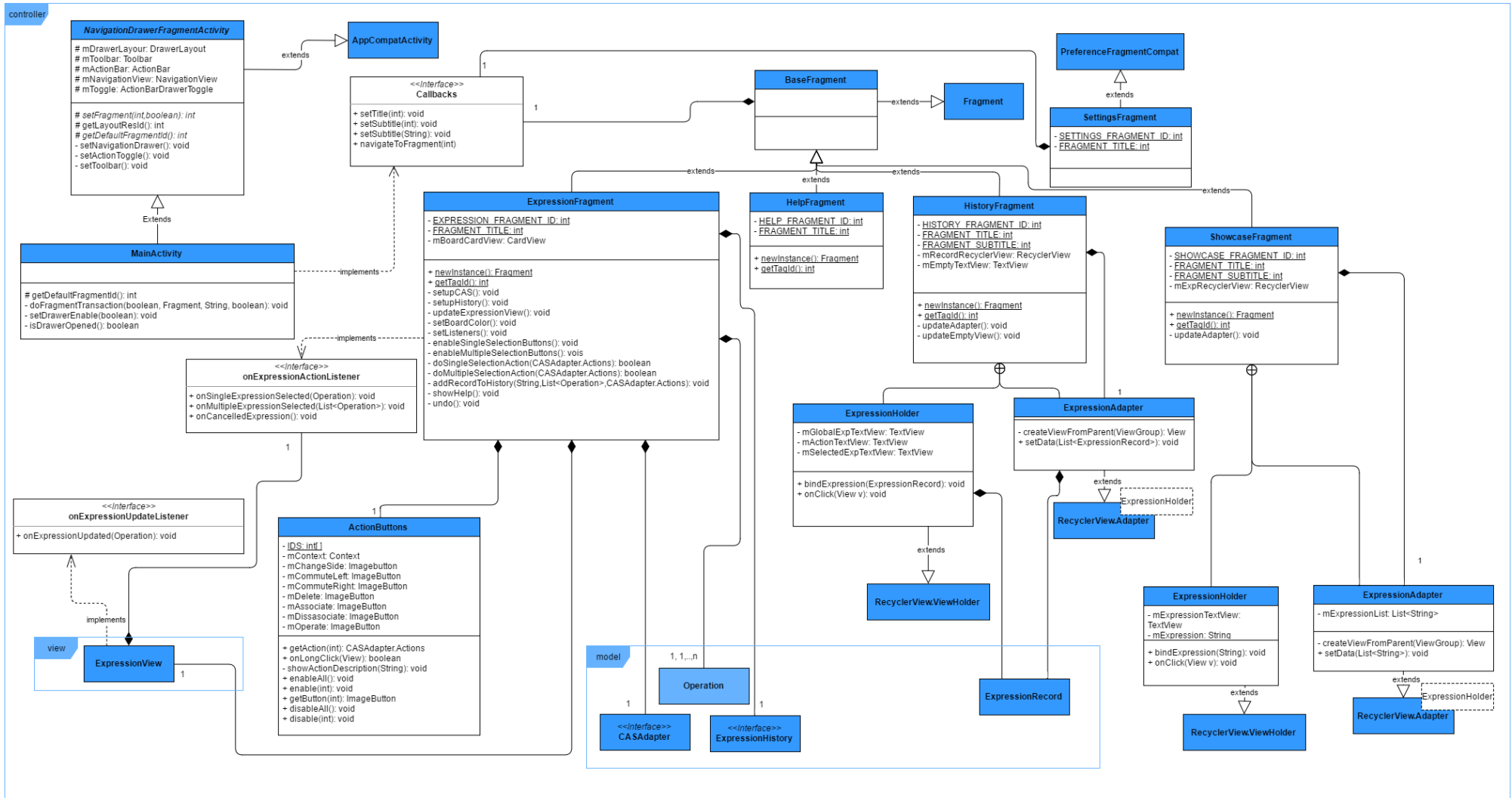


Figura B.3: Diagrama de clases de la capa Controlador

Anexo C. Fichero de configuración

Este anexo contiene el fichero de definición usado para la generación del CAS que utiliza la aplicación. La definición de las reglas de equivalencia, así como su estructura y sus características se encuentran en el capítulo 7 del Trabajo Fin de Grado *Desarrollo de un motor de matemática simbólica para la herramienta Chalkpy* [2].

```
//DEFINICIONES DE ELEMENTOS
SUM(flag commutative:true, tag type args:(oper...), flag associative:true, tag desc:"Es
una suma", flag_in_cf:true, flag_in_args:true, tag_symbol:"+")
MINUS(tag type args:(oper), tag symbol:"-", tag desc:"Representa el menos unario")
PROD(tag_symbol:"*", flag commutative:true, flag_associative:true,
tag_type_args:(oper...))
EQU(tag_symbol:"=", tag_type:oper, tag_type_args:(oper,oper), flag_in_args:false)
NUMBER(tag symbol:"#", flag in cf:true, flag in args:true, tag_type_args:(number))
VAR(tag type:oper, tag symbol:"$", tag_type_args:(string))
ZERO(tag_symbol:"&ZERO", tag_type:oper)
ONE(tag_symbol:"&ONE", tag_type:oper)
MONE(tag_symbol:"&MONE", tag_type:oper)
INV(tag symbol:"@INV", tag_type_args:(oper))

//PROPIEDADES
//-----
//-----SUMA-----
//-----
+#[_a],[_b]] >>> #[operate("_a+_b")] //red0 +#[3],[5]
+#[_a],[&ONE[]] >>> #[operate("_a+1")] //red1 +#[100],[&ONE[]]
+#[&ONE[],&ONE[]] >>> #[operate("(float) 1+1")] //red2 +#[&ONE[],&ONE[]]
+[_a,&ZERO[]] >>> _a //red3 +#[4],[5],[&ZERO[]]
_a >>> +[_a,&ZERO[]] //red4 +#[4],[5]

//-----
//-----PRODUCTO-----
//-----
*#[_a],[_b]] >>> #[operate("_a*_b")] //red5 *#[3],[5]
*[_a,&ZERO[]] >>> &ZERO[] //red6 *#[100],[&ZERO[]]
*[_a,&ONE[]] >>> _a //red7 *#[53],[&ONE[]] *#[&ONE[],&ONE[]]
_a >>> *#[&ONE[],_a] //red8 +#[4],[5]

//-----
//-----MENOS-----
//-----
-[-[_a]] >>> _a //red9 -[-#[4],[5]]
-#[_a] >>> #[operate("-1*_a")] //red10 -#[5]
-#[&ZERO[]] >>> &ZERO[] //red11 -#[&ZERO[]]
-#[&ONE[]] >>> &MONE[] //red12 -#[&ONE[]]
-#[&MONE[]] >>> &ONE[] //red13 -#[&MONE[]]

//-----
//-----SUMA Y RESTA-----
//-----
+[_a,-[_a]] >>> &ZERO[] //red14 +#[x],[-#[x]]
+#[_a,-#[_b]] >>> #[operate("_a-_b")] //red15 +#[4],[-#[5]]
+[_a,-#[&ZERO[]]] >>> _a //red16 -#[5],[&ZERO[]]
+#[&ZERO[],-_a] >>> -[_a] //red17 -#[&ZERO[]],[5]

//-----
//-----SUMA Y PRODUCTO-----
//-----
+[_a,*[_b,_a]] >>> *#[&ONE[],_b],_a //red18 +#[4],[5],[4]]
+[*[_b,_a],*[_c,_a]] >>> *#[_b,_c],_a //red19 +[*#[4],[3]],[5],[3]]
*[_a,+[_b,_c]] >>> +*[_a,_b],*[_a,_c] //red20 *#[4],[3],[5]]

//-----
//-----INVERSO-----
//-----
@INV[@INV[_a]] >>> _a //red21 @INV[@INV#[6]]
@INV[-[_a]] >>> -[@INV[_a]] //red22
@INV#[_a] >>> #[operate("(float) 1.0/_a")] //red23 @INV#[10]]
```

```

@INV[&ONE[]] >>> &ONE[] //red24 @INV[&ONE[]]
@INV[&MONE[]] >>> &MONE[] //red25

//-----INVERSO Y MULT-----
//-----INVERSO Y MULT-----
//-----INVERSO Y MULT-----
*[_a,@INV[_a]] >>> &ONE[] //red26 *[#[4],@INV[#[4]]]
*[-[_a],@INV[_a]] >>> &MONE[] //red27
*[_a,@INV[-[_a]]] >>> &MONE[] //red28
*[#[_a],@INV[#[_b]]] >>> #[operate("(float) _a/_b")] //red29

//-----IGUALDAD-----
//-----IGUALDAD-----
//-----IGUALDAD-----
=[+[_a,_b],_c] >>> =[_a,+[_c,-[_b]]] //red30 =[+[#[4],#[3]],#[5]]
=[_a,_a] >>> =[+[_a,-[_a]],&ZERO[]] //red31 =[#[10],#[10]]
=[*[_a,_b],_c] >>> =[_a,*[_c,@INV[_b]]] //red32 =[*#[4],#[10],#[3]]

+[#[_a],&MONE[]] >>> #[operate("_a-1")] //red33
+[&MONE[],&MONE[]] >>> -[#[operate("(float) 1+1")]] //red34
*[_a,&MONE[]] >>> -[_a] //red35
_a >>> *[#&MONE[], -[_a]] //red36
*[#[_a], -[#[_b]]] >>> -[#[operate("_a*_b")]] //red37
=[+[_a,-[_b]],_c] >>> =[_a,+[_c,_b]] //red38
=[_a,_b] >>> =[&ZERO[], +[_b,-[_a]]] //red39
=[_a,_b] >>> =[+[_a,-[_b]], &ZERO[]] //red40
=[_c,+[_a,_b]] >>> =[+[_c,-[_b]],_a] //red41
=[_c,*[_a,_b]] >>> =[*[_c,@INV[_b]],_a] //red42
=[_a,+[_c,-[_b]]] >>> =[+[_a,_b],_c] //red43
=[*[_a,@INV[_b]],_c] >>> =[_a,*[_c,_b]] //red44
=[_a,*[_c,@INV[_b]]] >>> =[*[_a,_b],_c] //red45
-[_a] >>> *[#&MONE[],_a] //red46

```

Anexo D. Funciones de CASAdapter

Aquí se incluyen las funciones descritas en la sección 6.2.1.3 con un poco más de detalle, con el fin de ilustrar al lector sobre su funcionamiento interno.

commute

Esta función realiza una permutación a izquierda o derecha de un término en la expresión general. Por ello, los argumentos recibidos son el término a conmutar y una acción que permita decidir si se conmuta a izquierda o derecha una única posición. Para conseguir dicha transformación se llama a la función *commutativeProperty* de la clase *AlgebraicEngine*.

associate

El método *associate* implementa la propiedad asociativa de las operaciones matemáticas, creando una agrupación de términos entre paréntesis. Como argumento recibe dos objetos de clase *Operation* que referencian los términos de inicio y final de la asociación. Ambos términos deben estar al mismo nivel (tener el mismo padre) y no contener la subexpresión entera. Una vez realizadas las comprobaciones pertinentes se llama al método *associativeProperty* del motor algebraico externo.

dissociate

La propiedad disociativa trata de eliminar los paréntesis alrededor de una expresión. Por lo tanto, el método que la implementa recibe como argumento una expresión de tipo *Operation*. Se comprueba que sea una operación matemática y que esté contenida en otra expresión inmediata (su padre) que sea del mismo tipo, es decir, que tenga el mismo operador (por ejemplo, no se podría desasociar una multiplicación dentro de una suma y viceversa).

commonFactor

El sacar factor común consiste en, dadas unas expresiones unidas por una suma con un término común en cada sumando, obtener una nueva expresión en forma de multiplicación en la que haya dos multiplicandos: el término común y la suma de los términos no comunes. El método que realiza esta operación recibe una lista de objetos *Operation* y comprueba que se pueda aplicar la equivalencia. Para esto es necesario que se cumpla alguno de los siguientes casos:

- a. Todos los términos de la suma son multiplicaciones con el término común como uno de sus factores. Recuérdese que en este caso el padre de los mismos debe ser idéntico.
- b. Misma situación que el caso a), pero algún término no es una multiplicación. En ese caso, el padre de dicho término debe ser el abuelo de los que sí son una multiplicación. De esta forma, el término que no cumple ser una puede ser reescrito como una multiplicación de él mismo multiplicado por un uno. Por ejemplo:

$$a + (a*b) + (c*a)$$

puede ser considerado como:

$$(1*a) + (a*b) + (c*a)$$

de modo que al sacar factor común se obtiene:

$$a*(1 + b + c)$$

Una vez transformada la expresión inicial, se aplica la reducción adecuada de las que el CAS provee para realizar la extracción del factor común de manera recursiva.

operate

La función que permite operar los términos dentro de una expresión se llama *operate*. Para poder operar es necesario, en primer lugar, un único objeto *Operation* que sea una operación matemática (esto es, que disponga de operador). Una vez sabido si se puede proceder, de manera recursiva se van agrupando elementos de dos en dos y aplicando las reglas de operación para cada tipo de operación. Esto permite seleccionar operaciones que contengan a su vez otro tipo de operadores. Al mismo tiempo permite realizar las transformaciones de cambio de signo en los elementos con doble signo menos.

changeSide

Cambiar de lado una expresión en una igualdad implica conocer el operador que la contiene y si se encuentra en el nivel principal de la ecuación, es decir, si su padre o abuelo es el objeto *Operation* asociado a la igualdad. Si resulta que ésta se encuentra en el nivel inmediato superior, la expresión seleccionada es un término entero de la ecuación, y se pasa al otro lado con signo más o menos, dependiendo del signo anterior. En caso de que el término a cambiar sea un nieto de la igualdad, pasa al otro lado con el signo opuesto al del operador de su padre (por ejemplo, en una suma pasaría con signo menos; en una multiplicación con signo de inverso, esto es $1/\text{expresión}$). Por tanto recibe como argumento un único objeto de tipo *Operation*.

distribute

Permite aplicar la propiedad distributiva sobre una selección doble: un elemento único (que no sea una operación matemática) que actúe como pivote y una expresión que sea una suma de términos. El padre de ambas debe ser una multiplicación. De manera similar al factor común, utiliza el pivote para ir aplicando la propiedad de dos en dos.

Anexo E. *DrawableExpression* y clases de apoyo

En este anexo se incluye una explicación más extensa sobre las clases que permiten dibujar una expresión en la pantalla *Tablero*.

Utilizando la estrategia “*divide y vencerás*”, que permite atacar un problema dividiéndolo en otros más simples y, resolviéndolos, llegar a la solución del principal, se crearon clases de apoyo a *ExpressionView*, de manera que al llamar al método *onDraw*, lo único que hubiera que hacer fuera llamar al método homónimo de los componentes de apoyo.

Dichos componentes son los que heredan de la clase llamada *DrawableExpression*. Modelizan los elementos de una expresión matemática de manera que puedan autodibujarse en un *canvas*.

E.1 *DrawableExpression*

Es la clase abstracta básica. En ella se encuentra la funcionalidad básica de estos componentes: coordenadas, cálculo de los límites del contenedor, colores, tamaño de texto, etc. Además provee ciertos métodos para indicar cuál es el tipo de expresión que es: paréntesis, operador, número/variable o una operación.

Por otro lado, da el esqueleto básico para las funciones *select*, que devuelve el elemento que se encuentra en una posición dada y lo selecciona (esto es, cambia el color con el que se pinta), y *getDrawableAtPosition*, que dadas unas coordenadas devolverá la expresión que se encuentre en ellas o *null* en caso de no encontrar nada.

Ambas funciones devuelven además la profundidad a la que se ha encontrado la expresión, permitiendo saber si selecciones sucesivas pertenecen a la misma subexpresión o no.

E.2 *DrawableSingleExpression*

Modeliza una expresión matemática de un único término, siendo éste un número o una variable. Hereda de *DrawableExpression*, por lo que obtiene de ella su funcionalidad básica. Las diferencias radican en la posesión de un campo interno de tipo *Operation*, que permite saber que expresión es la que modeliza, y la modificación de la función *getDrawableAtPosition*, que lo único que hace es comprobar si las coordenadas están dentro de los límites del contenedor de la clase.

E.3 *DrawableExpressionList*

Esta clase modeliza una expresión matemática entera, es decir, una lista de listas cuyos elementos son subexpresiones matemáticas. Sus principales diferencias con la clase de la que hereda son las siguientes:

- a. Tiene dos campos internos:
 - i. *mExpression*: un objeto de tipo *Operation* que sirve como referencia de la expresión que modeliza la clase.
 - ii. *mDrawableExpList*: una lista cuyos elementos son de tipo *DrawableExpression*, con lo que se consigue que sea una lista de listas. Es el elemento que se dibujará en el *canvas*. La clase posee un método que permite, dado un elemento *Operation*, generar la lista de

DrawableExpression de manera recursiva y que estén en orden infijo¹ para ser dibujados.

- b. La función *getDrawableAtPosition* cambia para ir buscando la expresión que contenga las coordenadas de manera recursiva, ya que una expresión mayor puede contener las coordenadas, pero también puede ser que una de sus subexpresiones las contenga.

E.4 *DrawableOperator, DrawableParenthesis*

Al igual que su hermana *DrawableSingleExpression*, estas dos clases difieren de la básica en que en vez de contener un elemento *Operation*, el campo interno es una cadena de caracteres que representa la expresión simbólica del operador o del paréntesis, de manera que la clase *DrawableExpressionList* sea capaz de crear estas clases a voluntad.

¹ El orden infijo es el estándar en matemáticas: *exp <operador> exp* (p.ej. $a+b+(c*d)$)

