Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Diagramas de factorización interactivos

Interactive factorization diagrams

Eduardo Estévez Rodríguez

La Laguna, 13 de septiembre de 2022

D. **José Gil Marichal Hernández**, con N.I.F. 78.677.406-H, profesor contratado doctor adscrito al área Teoría de la Señal y Comunicaciones, del Departamento de Ingeniería Industrial de la Universidad de La Laguna, como tutor

D. **Óscar Gómez Cárdenes**, con N.I.F. 78.859.209-Y, doctor en Informática por la Universidad de La Laguna, como cotutor

C E R T I F I C A N

Que la presente memoria titulada:

"*Diagramas de factorización interactivos*"

ha sido realizada bajo nuestra dirección por D. **Eduardo Estévez Rodríguez**, con N.I.F. 51.168.987-J.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 13 de septiembre de 2022.

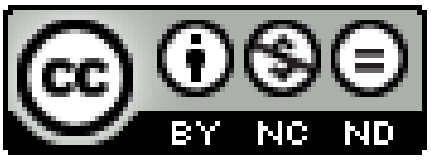
Agradecimientos

En primer lugar, a mi tutor José Gil Marichal Hernández, y cotutor, Óscar Gómez Cárdenes, por su confianza en mi persona para llevar a cabo este proyecto desde el primer momento, por brindarme su tiempo, su conocimiento y su ayuda durante el transcurso de este período.

Por otro lado, quisiera agradecer a la Universidad de La Laguna por haberme formado durante estos años y que me ha permitido disfrutar de una enseñanza de calidad.

Por último, a mi familia y a mis amigos, por aportarme el apoyo y la comprensión que necesitaba en todo momento durante el transcurso de esta etapa universitaria.

Licencia



Resumen

Este proyecto tiene como objetivo elaborar una aplicación Android para la visualización interactiva de la descomposición factorial de números enteros. De esta manera, la aplicación permite navegar dentro de un rango de números enteros y a su vez mostrar todas las descomposiciones posibles de dichos números, construyendo un sistema recursivo de engranajes.

El resultado de este proyecto es una herramienta didáctica para facilitar el aprendizaje de la descomposición factorial de un número entero. Para su implementación se ha utilizado el entorno de desarrollo integrado Android Studio y el lenguaje de programación Kotlin.

Las principales tecnologías Android utilizadas en el desarrollo de la aplicación: *Gesture Detector* y *Canvas*.

Palabras clave: diagramas de factorización, recursividad, didáctica, descomposición factorial, kotlin, android.

Abstract

This project aims to develop an Android application for visualizing interactive factorial decomposition of integers. In this way, the application allows you to navigate within a range of integers and display all possible decompositions of said numbers, building a recursive system of gears. The result of this project is a didactic tool which eases the learning of factorial decomposition of an integer. For its implementation, it has been used the Android Studio Integrated Development Environment and Kotlin programming language. The main Android technologies used during the development of the application were: Gesture Detector and Canvas.

Keywords: factorization diagrams, recursion, didactics, factorial decomposition, Kotlin, Android.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Descripción general del proyecto

El objetivo principal de este proyecto es la creación de una aplicación capaz de ser ejecutada en dispositivos móviles Android, que muestre todas las factorizaciones posibles de un número en forma de diagrama animado que es construido mediante engranajes anidados.

Como se trata de una herramienta didáctica debe mostrar la información de manera concisa. Por ello, no solo se debe plasmar el diagrama en funcionamiento, sino que también se debe indicar mediante texto en qué número se encuentra y la factorización actual del mismo. De esta manera, se busca que el usuario pase el mayor tiempo posible interactuando y así retenga la mayor cantidad de información posible. Por último, se debe indicar cuál es la factorización canónica del número, puesto que es el conjunto de factores que normalmente más se utiliza en el ámbito matemático.

1.2. Antecedentes

La asignatura de matemáticas es considerada de las más difíciles con las que una persona se puede tropezar a lo largo de su etapa estudiantil. Es por ello, que se han desarrollado múltiples maneras de impartir el temario para conseguir el mejor método de enseñanza. Sin embargo, con la evolución de la tecnología se han abierto infinidad de posibilidades para la instrucción de éste área de conocimiento. Tanto es así, que se han realizado diversos estudios donde comprueban el rendimiento académico del estudiante después de implantar el uso de las TIC.

Según un estudio realizado por la Universidad de Córdoba sobre el uso de las TIC en la enseñanza de la matemática básica(1), concluyeron que la tecnología permite construir una visión más amplia y profunda sobre este ámbito. Además, que de la muestra que utilizó este método, el 91 % consiguió superar la asignatura y el 46 % de ese porcentaje consiguió altas calificaciones.

Por otro lado, la Universidad Técnica de Cotopaxi (Latacunga, Ecuador), expone que *”La utilización de recursos didácticos interactivos atrae la atención del estudiante y es de*

gran importancia para el desarrollo del proceso de enseñanza aprendizaje.” tras realizar un estudio sobre la utilización de recursos didácticos interactivos a través de las TIC en el proceso de enseñanza aprendizaje en el área de matemática.(2)

En este sentido, el uso de herramientas tecnológicas para la enseñanza matemática en los centros de educación cada vez es más común. A continuación se muestran algunos ejemplos de herramientas(3) utilizadas en la enseñanza a nivel nacional:

- KBruch
- CaRMetal
- wxMaxima
- GeoGebra

1.3. Factorización de números enteros

La factorización numérica de números enteros es el término matemático que se aborda en este proyecto. Dicho término consiste en descomponer un número compuesto (no primo) en divisores no triviales, que cuando se multiplican dan el número original.(4)

Este se enseña en los centros educativos a una temprana edad como herramienta para llevar a cabo otras operaciones. El método convencional de enseñanza consiste en plasmar una línea, donde en su parte izquierda se van apuntando los dividendos del número entero y en su parte derecha se apuntan los divisores del mismo. A continuación se puede observar un ejemplo visual:

120	2
60	2
30	2
15	3
5	5
1	

Figura 1.1: Método de descomposición en factores convencional del número 120.

Sin embargo, existe una manera alternativa, más visual, de representar esta descomposición: mediante diagramas. Consiste en representar el número mediante agrupaciones de elementos, formando así una figura armónica donde se aprecian claramente los factores es lo que está descompuesto el número. A continuación se puede observar un ejemplo:

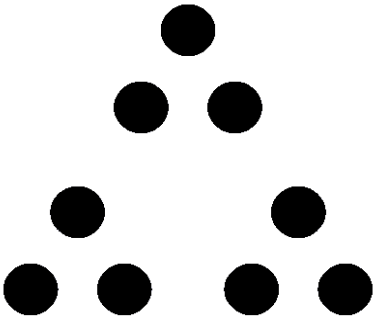


Figura 1.2: Diagrama de factorización del número 9.

Las operaciones en donde son más comunes las primeras apariciones de este término matemático son el *mínimo común múltiplo (m.c.m)* y el *máximo común divisor (m.c.d)*. Se trata de una operación básica para el posterior aprendizaje de la factorización algebraica.

1.4. Estado del arte

Como se ha visto con anterioridad, existen métodos para enseñar a factorizar números enteros. No obstante, como ya se ha comentado, el objetivo del proyecto es desarrollar una herramienta didáctica visualmente atractiva para despertar el interés en el usuario.

El primer paso ha sido realizar una búsqueda a través de las aplicaciones gratuitas ya disponibles en la tienda de Android. Tras ello, se han analizado diversas opciones para que se tengan como punto de referencia. Sin embargo, ninguna aplicación representaba la descomposición mediante diagramas animados interactivos. A continuación se puede observar la aplicación(5) más relevante en cuanto a la factorización de números enteros bajo la búsqueda de *diagramas de factorización*:

Figura 1.3: Captura de la aplicación Prime Numbers and Factors

Partiendo desde este punto, se ha de realizar un diseño sencillo para facilitar la interacción del usuario de cualquier rango de edad. Aunque, a su vez deberá contar con un diseño atractivo visualmente para despertar el interés y facilitar el aprendizaje de este término matemático.

Capítulo 2

Tecnologías

En este capítulo se presentarán las tecnologías utilizadas para la realización del proyecto, tanto para el desarrollo de la aplicación, como para la organización interna del equipo. Además, se expondrán las tecnologías que permiten la detección de gestos a la hora de interactuar con la aplicación, y que plasman la animación del diagrama en pantalla.

2.1. Android Studio

Android Studio(6) es el Entorno de Desarrollo Integrado (IDE) oficial para el desarrollo de aplicaciones en Android. Este IDE brinda infinidad de posibilidades, ya que permite realizar en el mismo entorno tanto la parte lógica de la aplicación como su diseño. Esta flexibilidad nos brinda la posibilidad de desarrollar desde proyectos muy sencillos hasta aplicaciones mucho más complejas.

Al comenzar el proyecto se barajaron distintas opciones de entornos de desarrollo, tales como *React Native* o también *Xamarin*. Aunque éstas permiten el desarrollo de aplicaciones no solo para Android, sino también para dispositivos iOS, se ha elegido Android Studio debido a la previa experiencia del estudiante en este entorno y con el lenguaje de programación *Kotlin*. Además, la curva de aprendizaje de estas otras tecnologías habría sido mayor, y el tiempo que se dispone para llevar a cabo el proyecto es reducido.

Una de las características a destacar de Android Studio es la capacidad de realizar el diseño de las vistas en diversos dispositivos, consiguiendo así un diseño responsivo y adaptable a cualquier tamaño de pantalla. Además, se puede ejecutar cualquier cambio prácticamente en tiempo real, tanto en un dispositivo Android físico, conectado al equipo en el que se esté desarrollando, como también haciendo uso de dispositivos virtuales mediante el emulador del entorno. Esto permite que el diseño de la aplicación sea compatible con cualquier dispositivo en la mayor medida de lo posible. En este sentido, se puede comprobar el rendimiento de la aplicación en ejecución con diferentes recursos; es por ello que durante el desarrollo del proyecto se utilizaron ambas formas de ejecución.

Por otro lado, el desarrollo de una aplicación interactiva encaja con el patrón de arquitectura de software **Modelo-vista-controlador**. Gracias a todo lo comentado anteriormente, este IDE es ideal para llevar a cabo este patrón de desarrollo.

Patrón Modelo-Vista-Controlador (MVC)

Modelo-vista-controlador(7) es un patrón de arquitectura de software cuya característica principal es que permite separar la parte lógica de la aplicación de la representación gráfica, que es con la que va a interactuar el usuario directamente. De esta manera, se facilitan en gran medida el desarrollo y el mantenimiento del proyecto. Principalmente se pueden distinguir tres capas diferentes:

- *Modelo*: es la parte lógica del proyecto, es decir, las clases que se utilizarán para representar los datos que componen la aplicación. En este caso, se podría referir a los elementos que conforman el engranaje que representa el diagrama.
- *Vista*: es la parte gráfica, es decir, el diseño que estará de cara directamente con el usuario para que este pueda interactuar con la aplicación. Todos los ficheros que representan las diferentes vistas se encuentran dentro del directorio */layout*, separándolas así de la parte lógica.
- *Controlador*: es la parte intermediaria entre la vista y el modelo. Esto es, las clases que definen el comportamiento de las diferentes vistas y que además procesan toda la información que introduce el usuario mediante los diferentes eventos. La particularidad de estas es que todos los ficheros que son controladores, contienen por convenio el término *Activity* en su nombre.

A continuación, se hará hincapié en las **clases** de la tecnología Android **más relevantes** a la hora de desarrollar las funcionalidades del proyecto.

2.1.1. `View`

La clase `View`(8) es el nivel más alto del que heredan todos los elementos que conforman la GUI de una aplicación Android, ya sea un botón, un campo de texto, una lista de elementos, etc. Nos permite incorporar a nuestra parte gráfica cualquier elemento indispensable para la misma. Tanto es así que, para crear la animación del diagrama, se ha tenido que generar una nueva clase `DrawingView` que heredase de esta para poder plasmar el diagrama en movimiento. Para llevar a cabo esta tarea se ha hecho uso de la funcionalidad comentada a continuación.

`onDraw`

Este método de la clase `View` ha sido el más relevante de todo el proyecto, ya que ha permitido plasmar visualmente el diagrama en la pantalla del dispositivo. Este método

recibe como parámetro un `Canvas`, elemento que permite dibujar figuras de cualquier forma y color de manera relativamente fácil y eficiente; contiene un `Bitmap` en el que plasma las rutas que va trazando, siguiendo una técnica idéntica a la de una memoria RAM. Es decir, para dibujar una figura, el `Canvas` solamente activa en el `Bitmap` los píxeles exactos del color especificado. Sin embargo, la magia de este método reside en que se ejecuta constantemente en bucle, por lo que se puede generar una animación simplemente variando los frames.

2.1.2. GestureDetector

La clase `GestureDetector`(9) es la que ha permitido detectar los gestos más comunes que se realizan con un dispositivo táctil Android. Es una clase proporcionada por el entorno para poder diferenciar los distintos gestos que realiza el usuario y actuar en consecuencia. Así, el usuario puede utilizar gestos para navegar entre los distintos números y factorizaciones que ofrece la aplicación.

Por otro lado, como esta clase solo detecta los gestos, se necesitaba una clase que heredase de ella para poder disparar un evento cuando una interacción era detectada. La clase que se ha desarrollado se llama `GestureListener`, la cual hereda de `GestureDetector.OnGestureListener` y tiene sobrecargados dos métodos principales, cuya funcionalidad básica se explica a continuación.

onFling

Este método detecta el gesto que se conoce comúnmente como “deslizar”, la manera en la que lo hace es detectando cuando el usuario pulsa la pantalla (constante `ACTION_DOWN`), y acto seguido detecta cuándo el usuario deja de estar en contacto con el dispositivo (método `ACTION_UP`). Ambas constantes pertenecen a las clase `MotionEvent` que, a su vez, es un elemento de la clase `GestureDetector.OnGestureListener`.

Para poder identificar qué movimiento ha realizado el usuario, partiremos de los dos puntos proporcionados por la clase `MotionEvent`, aunque esto lo explicaremos más adelante.

Gracias a esta funcionalidad es posible realizar un “swipe” tanto vertical como horizontalmente para navegar a través de la aplicación.

onLongPress

Este método detecta el gesto que es conocido como una pulsación larga, es decir, cuando el usuario mantiene pulsado por un tiempo prolongado algún elemento de la interfaz gráfica de la aplicación. Este evento es mucho más fácil de disparar puesto que no requiere de cálculos previos para identificar la dirección en la que se realiza. Gracias a este gesto se ha permitido implementar la aparición de una segunda pantalla que muestra la factorización canónica del número actual que se muestra en la clase `DrawingView`.

2.2. Kotlin

Kotlin es un lenguaje de programación estático que admite tanto la programación funcional como la orientada a objetos. Aunque Android Studio permite la utilización del lenguaje Java, se ha escogido la primera opción debido a su interoperabilidad con Java. Esto es, si bien algunas API de Android son exclusivas de Kotlin, la mayoría están escritas en Java y pueden ser llamadas por el lenguaje Kotlin. Por tanto, este lenguaje permite beneficiarse tanto de las características más novedosas de la programación de alto nivel, como de toda la documentación y código Java que se ha desarrollado durante todos los años previos a su lanzamiento.

Por último, Por otra parte, cabe destacar la utilización de la herencia en el código desarrollado, por ejemplo, a la hora de desarrollar la clase que permite dibujar la animación del engranaje a través del `Canvas`; esta clase hereda todas las características de la clase interna `View` de Android Studio. Finalmente, se han utilizado diversos tipos de *Listeners* para detectar cualquier evento que dispare el usuario a la hora de interactuar con la interfaz de la aplicación, aunque de estas tecnologías se hablará en detalle más adelante.

2.3. Latex

Latex(10) es el lenguaje principal de la plataforma online **Overleaf**. La plataforma presenta un editor de texto para generar documentos de alta calidad tipográfica y su uso es más común en la elaboración de artículos científicos.

En este caso, ha sido el entorno donde se ha elaborado la memoria de este proyecto, recogiendo así hasta el último detalle de su desarrollo.

2.4. Inkscape

Inkscape(11) es un editor profesionales de vectores gráficos libre y de código abierto. Este programa se ha utilizado para realizar todo el diseño de logos e imágenes que se emplean en la interfaz gráfica.

Las imágenes vectorizadas(12) son un recurso muy importante en las aplicaciones móviles. Esto se debe a que al no saber qué dispositivo es el que va a ejecutar la aplicación, no se puede establecer un tamaño fijo de imagen. Sin embargo, con este tipo de imágenes esta cuestión pasa a ser irrelevante, puesto que los vectores permiten que la imagen varíe su tamaño sin perder resolución.

2.5. Telegram

Telegram(13) es un sistema de mensajería basado en la nube. La comunicación interna del equipo es prácticamente tan importante como el proyecto en sí. Por ello, se ha utilizado en gran medida esta aplicación para comunicar cualquier duda o avance del proyecto.

Por otro lado, se ha utilizado para aportar ideas de mejoras o nuevas funcionalidades, además de aportar recursos audiovisuales que aportaran datos relevantes para el desarrollo de la aplicación.

Capítulo 3

Metodología de trabajo

3.1. Metodologías ágiles

La metodología de trabajo que más se asemeja a la utilizada en el equipo durante el período de desarrollo del proyecto, ha sido una fusión entre las metodologías ágiles(14) **Extreme Programming XP** y **Kanban**.

- *Extreme programming XP*: esta metodología destaca por la división del proyecto en fases: primeramente, se realiza una planificación junto al cliente; luego, se realiza un diseño del proyecto; seguidamente, se codifica en parejas de programadores; por último, se testea lo implementado para corregir errores de código.
- *Kanban*: la característica principal es la elaboración de un diagrama con tres columnas donde se vayan plasmando por separado los objetivos *conseguidos*, os que están *en proceso* y los que permanecen *pendientes*. Este diagrama está al alcance de todos los miembros del equipo, mejorando así la coordinación y comunicación interna, y aumentando la productividad.

La fusión de ambas metodologías ha sido la clave de este proyecto, puesto que se han realizado todos los pasos de *Extreme Programming XP* y, además, se ha utilizado un diagrama para llevar un control de las tareas que se iban completando y las que quedaban pendientes. A continuación, se exponen las tecnologías utilizadas para poner en uso estas metodologías de trabajo.

3.1.1. Bitbucket

Bitbucket(15) es un servicio de alojamiento basado en web, para los proyectos que utilizan el sistema de control de versiones Git(16). Como cualquier proyecto, lo más conveniente es que sea almacenado en la nube, de modo que sea accesible por todos los miembros del equipo y se vayan almacenando las distintas versiones de la aplicación. Este fue el principal motivo de la elección de este servicio para alojar el código de la aplicación.

Una de las funcionalidades más ventajosas de Git es la posibilidad de crear diferentes ramas. De esta manera, el programador puede crear una rama para codificar una nueva funcionalidad, evitando tener que modificar todo el código anterior; en caso contrario, se podrían originar errores e inconsistencias en el código y, por ende, comportamientos inesperados en la aplicación. Una vez se termina de desarrollar la nueva funcionalidad, tras una serie de *commits*, el código es revisado por un supervisor y se da el visto bueno para fusionar esa *feature* con la rama semi-principal *develop* donde se van almacenando las versiones estables del proyecto. Una vez se completa la fusión, la rama *feature* es eliminada.

Finalmente, cuando la aplicación se da por finalizada y está preparada para lanzar su primera versión 1.0, se fusiona la rama *develop* con la rama principal *master*.

A continuación, se puede observar cómo se ha realizado la distribución de ramas:

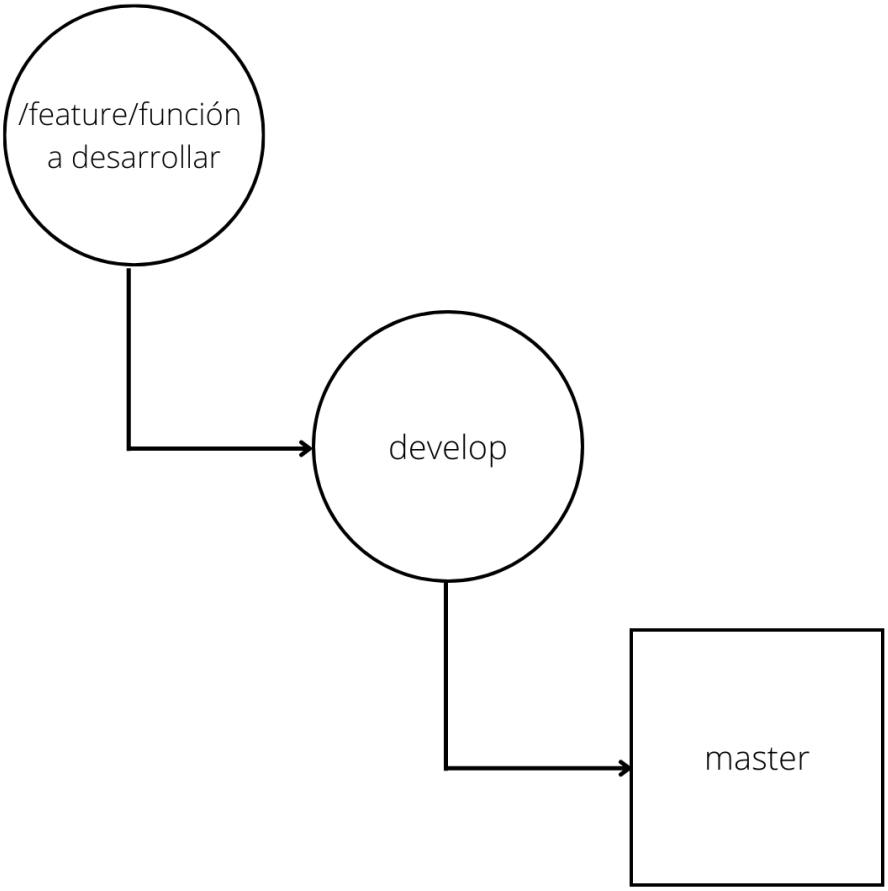


Figura 3.1: Esquema gráfico de las ramas del repositorio Git

3.1.2. Trello

Trello(17) es una herramienta visual que permite a los equipos gestionar cualquier tipo de proyecto y flujo de trabajo, así como supervisar tareas. Este instrumento permite generar un diagrama con diversas columnas, aunque lo más común es que el cuadro esté compuesto por tres *To Do*, *In process* y *Done*. De esta manera, el proyecto ha sido controlado y supervisado de forma organizada, facilitando la división de tareas tanto al

estudiante como al equipo tutor, teniendo claros en todo momento los objetivos a corto plazo que se buscaba alcanzar. A continuación, se muestra un ejemplo del uso de esta herramienta durante el proyecto:

Figura 3.2: Captura realizada del tablero compartido en Trello a mitad de la realización del proyecto.

Capítulo 4

Desarrollo

4.1. Splash screen

Una vez es ejecutada la aplicación, el primer elemento que se encuentra el usuario es una pantalla de bienvenida, donde se muestra tanto el logo de la aplicación como el de la Universidad de La Laguna, puesto que bajo el nombre de esta última se ha desarrollado este proyecto.

La única **funcionalidad** de esta pantalla reside en ocultar el tiempo de carga de la aplicación cuando se inicie por primera vez, por lo que el usuario no podrá interactuar con ella. Cuando la aplicación está lista para ser utilizada, esta pantalla desaparece, dejando paso a la pantalla principal.

El método que se ha llevado a cabo para implementar esta pantalla es muy sencillo. Tras generar ambos logos como **imágenes vectoriales**(12) y añadirlos al entorno de desarrollo, se crea un nuevo *drawable*, que es un recurso que se puede dibujar en pantalla, mediante una *layer-list*; así, ya estaría diseñada esta pantalla.

Por otro lado, para poder utilizar este diseño se necesita crear un nuevo estilo en el archivo de temas de la aplicación, *themes*. Esto se realiza para poder establecer este tema como predeterminado en la `MainActivity`, modificando el fichero `AndroidManifest.xml`.

Por último, para que la pantalla principal recobre su diseño original, se debe especificar programáticamente en el método `onCreate` del `MainActivity`. De esta manera, se genera ese efecto de transición entre ambas pantallas para dar una sensación de fluidez.

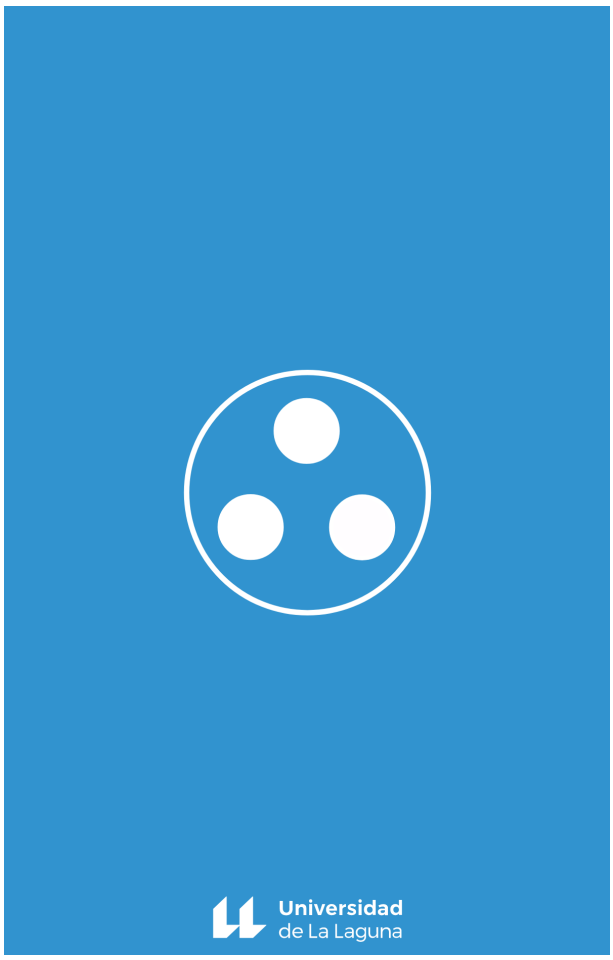


Figura 4.1: Captura realizada de la splash screen.

4.2. Pantalla de instrucciones

Esta segunda pantalla aparece únicamente la primera vez que la aplicación es iniciada, después de ser instalada, motivo por el que se ha escogido este orden de enumeración. No obstante, la interfaz gráfica de la pantalla principal cuenta con un botón que permite al usuario acceder en cualquier momento a esta funcionalidad.

La **funcionalidad** de esta nueva pantalla reside en esclarecer las acciones que puede realizar el usuario para interactuar con la aplicación mediante gestos. De esta manera, el usuario podrá entender perfectamente cómo funciona la interactividad con la interfaz gráfica de la pantalla principal.

El desarrollo de esta sección no tiene un alto grado de complicación, ya que aunque se haya generado una nueva actividad (`infoActivity`), el único propósito de la misma es mostrar una imagen vectorizada que ocupe toda la pantalla del dispositivo en que se ejecute la aplicación. Esta imagen se trata de un *overlay* donde se muestra una guía de gestos y sus funcionalidades.

Se ha añadido un *listener* al elemento `ImageView` para el evento `onClick`, siendo este el único evento programado en esta sección. Su propósito principal es que, cuando el usuario pulse sobre el elemento, que ocupa toda la pantalla del dispositivo, la actividad finalice, dando paso de nuevo a la pantalla principal.

Para que la aplicación obtenga el comportamiento de abrir automáticamente esta actividad, pero solo la **primera vez** que se inicia después de ser instalada, se ha utilizado el elemento *Preferences*. Así, la primera vez que se inicia, abre un fichero interno de claves e incluye un valor especificando que la aplicación ya ha sido iniciada. Por lo tanto, el resto de ocasiones en las que se inicie la app no se mostrará esta pantalla de información.

Figura 4.2: Captura realizada de la pantalla de instrucciones.

4.3. Pantalla principal

En este apartado se recoge la mayor parte del trabajo que se ha realizado durante todo el desarrollo del proyecto. Se desglosarán por apartados todas las funcionalidades que se han implementado en la pantalla principal.

La interfaz gráfica está dividida en **tres secciones**: una barra en la parte superior, un lienzo donde se plasma la animación del diagrama y, por último, una serie de campos de texto y un botón que muestran información sobre la factorización que se está representando en ese momento en el lienzo.

En primer lugar, la barra de acciones presenta en su lado izquierdo el logo de la aplicación junto con el nombre de la misma, y en su lado derecho se puede encontrar el botón de ayuda que da acceso a la pantalla descrita en el *punto 4.3*.

A continuación, se presenta el lienzo donde se muestra la animación del conjunto de engranajes que representan la factorización del número actual. Esta animación está en constante movimiento; más adelante se explicará cómo se han desarrollado cada uno de sus elementos, ya que es el corazón de la aplicación.

Por último, se pueden observar dos campos de texto, en uno se refleja el número actual y en el otro se muestra su factorización actual. Además, justo en la parte inferior de ambos se presenta un botón, aunque su funcionalidad se explicará en los siguientes apartados. Para poder visualizar la información anterior, se procede a mostrar una imagen con la pantalla principal de la aplicación:

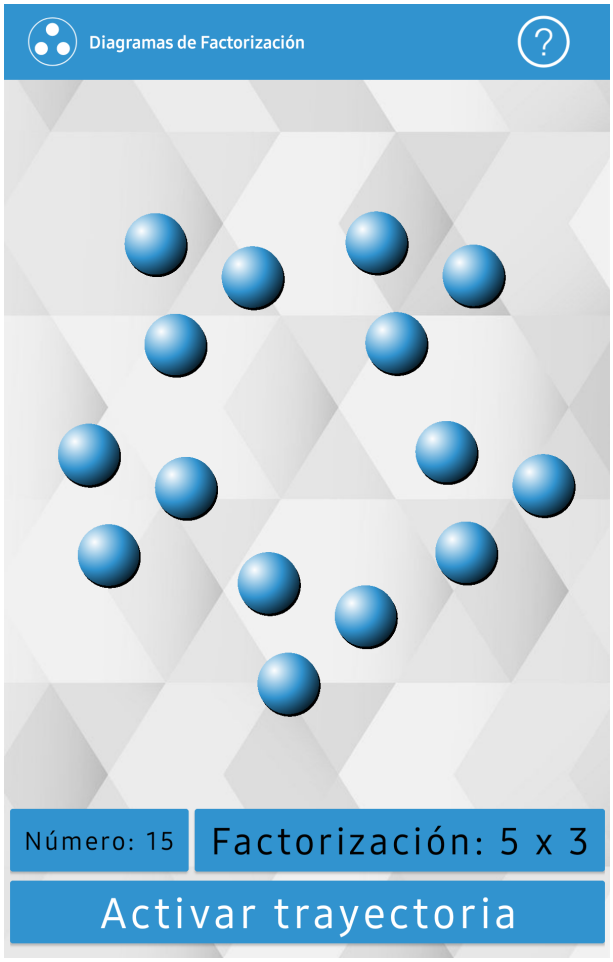


Figura 4.3: Captura realizada de la pantalla principal.

Una vez se ha llegado a este punto, se procede a desglosar cada funcionalidad desarrollada para dar vida a esta aplicación.

4.3.1. Procesamiento de la factorización

En esta sección se planteó en un momento el desarrollo de un algoritmo para generar todas las composiciones posibles del mismo. Sin embargo, tras ver que en números grandes la visualización del diagrama perdía funcionalidad y la oportunidad de ahorrar recursos computacionales, se ha decidido generar las descomposiciones en un rango

numérico desde el 1 al 120. Estas descomposiciones se han generado con un código externo a la aplicación, el cual genera un fichero con el resultado del algoritmo.

Partiendo desde este punto, el fichero con las factorizaciones ha sido añadido al entorno para poder acceder a él en modo de lectura. Así, se ahorra una gran cantidad de tiempo a la hora de construir el sistema de engranajes.

Una vez se ha llegado a este punto, la codificación se divide en varios procesos:

- **Atributos**
  En la pantalla principal se han generado tres atributos fundamentales. El primero, `currentNumber` que hace referencia al número que está representado actualmente. En segundo lugar, `numberFactorizationList` que como su propio nombre indica, almacena todas las factorizaciones del número actual. Por último, `indexNumberFactorization`, que indica qué factorización de la lista se debe representar en el diagrama.

- **Lectura del fichero**
  En este apartado se accede al fichero de texto que contiene las factorizaciones. Esta funcionalidad reside en el método `selectNumberFactorization` en el `MainActivity` y es posible llevarla a cabo gracias dos elementos. El primero es el método `open` de la clase `AssetsManager`(18) que permite acceder un fichero desde el directorio `assets`. El otro elemento se trata de una clase, `BufferedReader`(19), que permite extraer la información de un fichero en concreto.

  De este modo, se actualizaría la lista de factorizaciones del número que se va a representar mediante el sistema de engranajes.

- **Procesamiento de la factorización**
  Debido a que nuestro engranaje debe recibir una lista de números enteros y el fichero devuelve una lista de cadenas de caracteres, se debe realizar un pequeño procesamiento previo.

  De esta manera, mediante diversos métodos, primero se limpia la factorización de caracteres que sean irrelevantes, dejando solamente los números. Tras esto, se realiza una conversión de estos números en formato de cadena de caracteres para transformarlos en enteros.

Finalmente, el resultado de esta serie de procesos es una **lista numérica** con la factorización que se debe plasmar en el diagrama.

4.3.2. Sistema de engranajes

Tras realizar el paso anterior, la estructura de engranajes recibe la factorización que debe representar. Este sistema es el corazón de la aplicación, por lo que a continuación se detalla cómo ha sido el razonamiento que se ha llevado a cabo.

Creación del sistema de engranajes estático

En primer lugar, se ha tenido en cuenta que el engranaje no puede sobrepasar los **límites de la pantalla** del dispositivo en el que se vaya a generar. Por ello, se ha necesitado utilizar un método específico, `onSizeChanged`, de la clase `DrawingView` para obtener las medidas del `Canvas` dinámicamente y así escoger la medida más pequeña para establecerla como radio del engranaje, además de obtener el centro exacto del mismo. De esta manera, se generaría un primer engranaje "padre" centrado en la pantalla y con el radio adecuado para no sobrepasar los límites del dispositivo.

Figura 4.4: Captura realizada del sistema de engranajes centrado en la pantalla.

Una vez establecida esta medida, se continua con la **adición de los engranajes "hijos"** de manera recursiva, generando así el sistema completo del diagrama.

Para llevar a cabo este paso, se necesita partir de la factorización que se ha extraído del punto anterior. De esta manera, el engranaje recorre recursivamente cada uno de sus hijos, añadiendo tantos engranajes nuevos como se especifique en la factorización. La realización de este proceso ha resultado un tanto compleja, ya que para poder añadir un elemento al sistema se deben realizar los cálculos de su posición exacta mediante **semejanza de triángulos** tal y como se expone en la siguiente imagen:

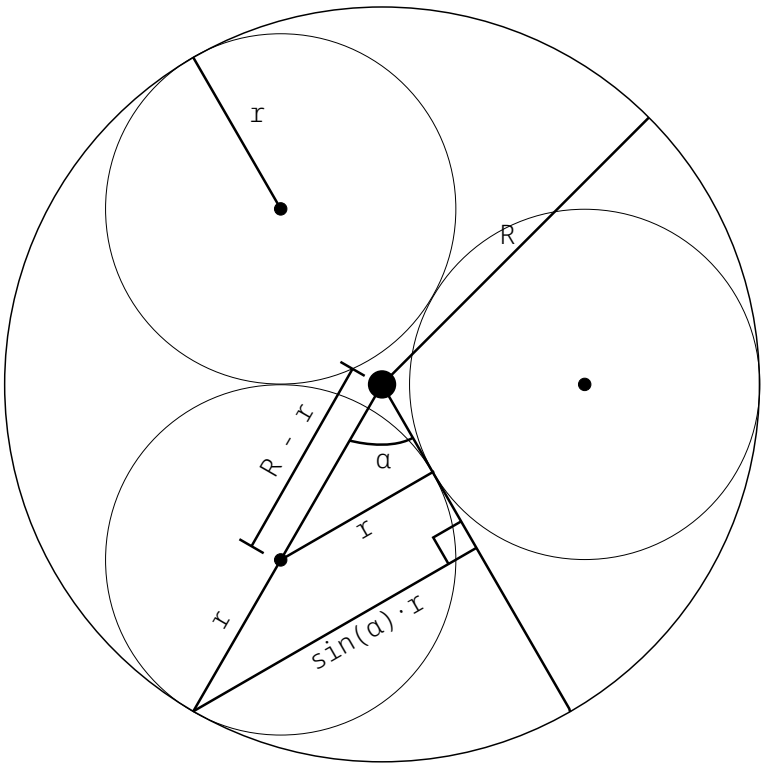


Figura 4.5: Esquema para el cálculo del radio y centro del engranaje hijo.

Fórmulas extraídas del esquema de la *figura 4.5.*

$$\begin{aligned} \alpha &= \frac{\pi}{\texttt{nChild}} \\ r &= \sin(\alpha) \times \frac{R}{1+\sin(\alpha)} \\ C_{i,x} &= (R-r) \times \sin(i \times 2 \times \frac{\pi}{\texttt{nChild}}) \\ C_{i,y} &= (R-r) \times \cos(i \times 2 \times \frac{\pi}{\texttt{nChild}}) \end{aligned} \tag{4.1}$$

De esta manera, se consigue generar un sistema como el que se puede observar en la *figura 4.4*. Cada hijo i estará caracterizado mediante un centro $(C_{i,x}, C_{i,y})$ y un radio r. Con lo expuesto hasta ahora simplemente se ha conseguido generar un **sistema estático**. El siguiente paso que se va a realizar es diseñar un sistema para la **animación**.

Sistema de engranajes animado

Para desarrollar este efecto de animación se debe partir de la clase que controla lo que se va a plasmar en la pantalla: `DrawingView`.

En primer lugar, se ha establecido el número de *frames*(20) que se van a dibujar por segundo, en este caso se ha fijado en 60fps. Esto se puede llevar a cabo gracias al método `invalidate` que contiene esta clase. Este resetea el *Canvas* cada cierto período de tiempo gracias a un `Timer`. Para calcular el período exacto para limitar la animación a

60fps se ha utilizado la siguiente expresión:

$$\texttt{period} = \frac{1}{\texttt{fps}} \times 1000 \text{ ms} \tag{4.2}$$

Así, se entiende que la **fase** de cada engranaje tendrá 60 posiciones diferentes por segundo durante su movimiento circular(21). Dicha fase dependerá directamente del número del frame que se esté dibujando en ese momento y de la velocidad del engranaje, por lo que esta es la manera de obtener la fase (ϕ):

$$\phi = \texttt{frame} \times \texttt{speed} \tag{4.3}$$

Aquí entra en escena la **velocidad** del engranaje, puesto que dependiendo de ella, la fase se desplazará más o menos en la circunferencia. Sin embargo, aunque la velocidad del engranaje padre será de 1 / fps, ya que así coincidirá con los fps a los que se ha fijado el movimiento, no ocurrirá lo mismo con la de los hijos.

La velocidad de los hijos se irá incrementando e invirtiendo en cuanto se vaya avanzando por los niveles de profundidad hasta llegar al nivel número dos. A partir de ahí, solo se invertirá el signo de la velocidad para que no se dispare el movimiento de rotación de los engranajes más pequeños. De esta manera, al invertir el signo de este valor, se consigue que la fase de ese engranaje gire en sentido contrario para generar un efecto visual atractivo en el movimiento del sistema.

Una vez se ha señalado la importancia de estos valores en la animación del sistema, se puede dar paso a recalcular la nueva posición de cada engranaje dependiendo de la fase, el frame y la velocidad que tengan en ese momento exacto. Así, se genera el movimiento, actualizando constantemente la posición de los engranajes “hijos”.

Como lo único que se modifica durante el movimiento es el centro del engranaje, simplemente debemos aplicar las fórmulas presentadas en la *figura 4.5* para calcular el centro de cada hijo $(C_{i,x}, C_{i,y})$ dentro de la animación. Sin embargo, se debe mover dependiendo de la fase actual de la circunferencia, por lo que la calcularemos el nuevo centro animado de cada hijo $(P_{i,x}, P_{i,y})$ según la fórmula:

$$P_{i,x} = (C_{i,x} + (R - r) \times \cos(\phi + i \times 2 \times \frac{\pi}{\texttt{nChild}}) \tag{4.4}$$

$$P_{i,y} = (C_{i,y} + (R - r) \times \sin(\phi + i \times 2 \times \frac{\pi}{\texttt{nChild}}) \tag{4.5}$$

Finalmente, se ha completado la implementación del sistema de engranajes animado. Así, se mantiene separada la construcción y actualización del sistema de la realización del dibujo en el *Canvas*, aportándole calidad al código y facilidad para su mantenimiento.

Dibujar sistema de engranajes

El sistema diseñado para dibujar el sistema es casi tan importante como el sistema en sí, ya que es lo que el usuario verá como resultado final. Su implementación no ha

conllevado grandes problemas, puesto que con una simple función recursiva se pueden recorrer todos los engranajes "hijos" que conforman el diagrama hasta llegar al *nodo hoja*, el cual será el único que plasme su centro en la pantalla; el centro de este engranaje se trata de un objeto de la clase `CircumferenceElement`.

La dificultad reside en darle una **textura de pelota 3D** sin el uso de imágenes, puesto que esta práctica ralentizaría en gran medida el rendimiento de la aplicación. Para ello se ha ingeniado un sistema donde se pintan dos puntos en pantalla.

El primer punto es completamente de color negro y efectúa el efecto de sombra. En cambio, al segundo punto se le ha asignado un **gradiente** de colores y se ha desplazado su centro en un pequeño porcentaje para obtener un efecto de profundidad. De esta manera, se ha conseguido obtener un efecto de 2.5D como se observa en la *figura 4.3*.

Para cerrar este apartado, se ha de especificar que tanto la actualización de la posición de los engranajes como su posterior dibujado, se efectúa periódicamente en el método `onDraw` expuesto con anterioridad.

4.3.3. Detección de gestos

La detección de gestos conforma una gran parte de la interactividad de la aplicación, pues de esta manera el usuario puede tanto cambiar de número como navegar dentro de las distintas factorizaciones que posee.

Aunque este apartado ya ha sido previamente introducido en el *apartado 2.4.*, se ha de explicar con mayor detalle cómo se ha implementado cada uno de los métodos que permiten disparar distintos eventos dependiendo de la acción que realice el usuario.

Deslizar

La acción `deslizar` permite al usuario cambiar de número si desliza horizontalmente; en cambio, si desliza verticalmente puede navegar a través de las distintas factorizaciones de dicho número.

Como se expone en el *apartado 2.4.1*, el método `onFling` de la clase `GestureListener` es el encargado de detectar cuándo el usuario realiza la acción de deslizar su dedo por la pantalla. Este método nos proporciona dos coordenadas: el punto origen y el punto destino del movimiento.

Gracias a estos dos puntos ha sido posible codificar el comportamiento deseado, por lo que, el proceso de implementación se puede dividir en dos partes:

- **Horizontal o vertical**: para diferenciar ente ambos movimientos se ha implementado un pequeño cálculo, que identifica qué movimiento se ha realizado mediante el cálculo de la diferencia de coordenadas entre el punto origen y final. Esto significa que, si la diferencia en valor absoluto de las coordenadas en X es mayor que la dife-

rencia en valor absoluto de las coordenadas en Y, se ha deslizado horizontalmente. En caso, contrario, se ha deslizado verticalmente.

- **Direccionado del movimiento**: para saber si dentro del movimiento horizontal se ha deslizado hacia la derecha o a la izquierda, o por el contrario, para diferenciar entre un movimiento hacia arriba o hacia abajo en el movimiento vertical, se ha de tener en cuenta el siguiente dato. Partiendo de la diferencia de coordenadas en X o en Y, dependiendo del caso, se distingue la dirección del movimiento gracias al signo de este valor.

Mantener presionado

En este caso, el gesto que se ha programado no requiere de ningún cálculo previo, puesto que la clase de la que hereda ya realiza todo el trabajo automáticamente. Esto quiere decir que simplemente se ha implementado el evento que dispara esta acción, el cual se expondrá más adelante, puesto que abre una nueva pantalla.

4.3.4. Efecto borde

Una de las funcionalidades que se ha implementado para incrementar la calidad de la interacción con el usuario, ha sido **delimitar bordes**. Es decir, cuando el usuario comience a deslizar e intente salirse del rango de números y factorizaciones establecidos, se mostrará un elemento visual en la pantalla en forma de borde que irá desapareciendo poco a poco. De esta manera, el usuario entenderá que ha llegado al límite y no podrá seguir interactuando en esa dirección.

La manera en la que se ha implementado esta funcionalidad parte de la detección de gestos, ya que esta, antes de ejecutar el evento, comprueba que el usuario está intentando sobrepasar el rango de factorizaciones establecidas y lanza una señal del borde de la pantalla donde debe mostrar la indicación de que ha llegado al final.

Tras obtener esta información, la aplicación dibujará una línea mediante el *Canvas* de la clase *DrawingView* en las coordenadas exactas. Sin embargo, se desea conseguir un efecto de fluidez para que ese borde vaya despareciendo poco a poco. Por lo tanto, se debe establecer una duración de esta **animación** y un contador de frames para que vaya desapareciendo el color en cuanto el contador vaya aumentando. Este efecto se consigue con el componente *alfa* del RGBA(22):

$$\texttt{color} = \textbf{argb}(\frac{\texttt{frameCounter}}{\texttt{animationFrames}}, \texttt{redValue}, \texttt{greenValue}, \texttt{blueValue}) \quad (4.6)$$

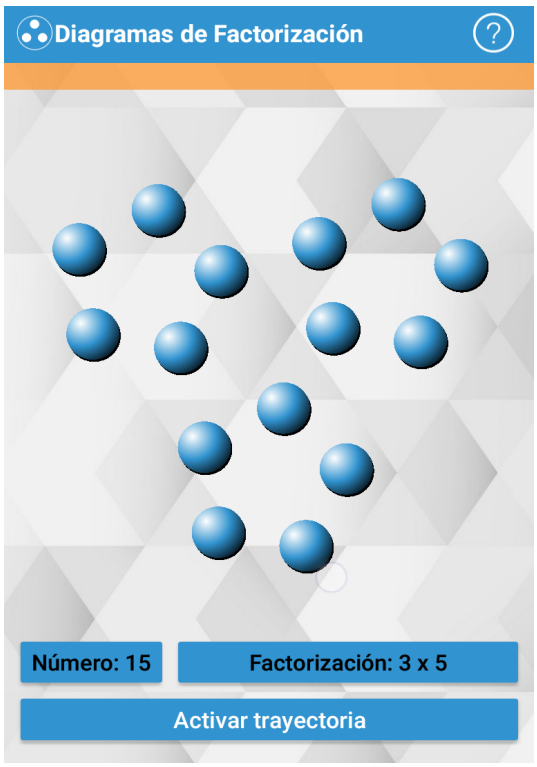


Figura 4.6: Captura realizada del efecto de borde delimitando la interactividad.

4.3.5. Transición de trayectorias suaves

Hasta el momento, el engranaje que estaba representado en la interfaz gráfica estaba en constante movimiento; sin embargo, si el usuario realizaba la acción de deslizar, el engranaje era actualizado inmediatamente. Esto generaba un corte en la animación, destruyendo por completo la sensación de fluidez en el movimiento.

Para poder evitar este corte, se ha diseñado un proceso que genera un efecto visual para que el usuario no se percate de lo que realmente está ocurriendo. En primer lugar, cuando el evento interactivo de deslizar es lanzado, se realiza el mismo proceso que se ha ejecutado para crear el sistema de engranajes con la nueva factorización, pero esta vez se creará un sistema de engranajes auxiliar.

La técnica que se ha utilizado en este apartado consiste en la consecución de una **serie de pasos** que se ejecutan en el siguiente orden:

- **Posición actual del sistema de engranajes**
  Para poder realizar un movimiento realista, justo en el momento en el que se dispara el evento de deslizar, se calculan las posiciones exactas de los engranajes finales en el *frame* en el que se ha ejecutado el evento, mediante el método `getCurrentCoordinates` de la clase `Gear`. De esta manera, ya se obtiene una **lista de posiciones** para el engranaje actual y se repite este proceso para obtener otra lista para el sistema de engranajes nuevo.

- **Algoritmo de emparejamiento**
  Partiendo de las dos listas generadas en el apartado anterior, se ha desarrollado un algoritmo que empareja los puntos del sistema nuevo con la opción más cercana del

sistema actual. Esto se ha realizado calculando para cada punto todas las distancias a todos los puntos del sistema actual, y se ha ordenado la lista resultante por el campo de *distancias*.

A continuación, se ha ido recorriendo dicha lista y marcando ambos puntos como visitados cada vez que se introducía la pareja en la lista final. De esta manera, cada punto estaría emparejado con su **mejor opción** siempre que esta no haya sido visitada con anterioridad por otro de los puntos.

La única complicación que existe aparece cuando ambas listas son de distinto tamaño, es decir, cuando el usuario ha cambiado de número y no solo de factorización. No obstante, esto se ha solucionado añadiendo puntos "vacíos" a la lista de menor tamaño, señalando así cuándo se trata de un punto **alfa**.

- **Pintar transición**
  Para finalizar este efecto, simplemente hay que plasmar la animación en el `Canvas` donde se ha realizado todo hasta el momento. Partiendo del *punto 4.3.4.* se ha explicado cómo realizar una animación con un número específico de *frames* y este no es un caso diferente. Para comenzar la animación se debe pausar la constante actualización del sistema de engranajes actual, para poder pintar perfectamente la transición.

  Tras definir un tiempo de ejecución de la animación en *frames*, se da comienzo a la misma. Esta no utiliza engranajes en ningún momento, simplemente pintará puntos que se trasladarán de una posición a otra por la pantalla. El proceso comienza con los puntos en la posición donde se encontraban en el engranaje actual, y a través de una técnica conocida como **lerp**(23) se van trasladando los puntos desde su posición inicial hasta la posición destino, que se corresponden con las posiciones del sistema nuevo.

$$\begin{aligned} p &= \frac{\texttt{lerpCount}}{\texttt{frames}} \\ q &= 1 - p \\ \texttt{current}_x &= q \times \texttt{origin}_x + p * \texttt{final}_x \\ \texttt{current}_y &= q \times \texttt{origin}_y + p * \texttt{final}_y \end{aligned} \tag{4.7}$$

  Esta técnica es tan poderosa que se ha utilizado para realizar el *lerp* con el **radio del punto**, para que así el radio aumente o disminuya en función del radio de los puntos del nuevo sistema de engranajes. Además, se ha utilizado la misma técnica que en el *apartado 4.3.4.* para los puntos *alfa*, ya que estos nodos no tienen correspondencia en la lista de emparejamiento; por lo tanto, deberán aparecer o desaparecer progresivamente. Por ello, se ha vuelto a utilizar el parámetro **alfa** del sistema *RGBA*.

  Una vez termine la transición, simplemente se sobreescribe el sistema antiguo con el nuevo y se reinicia la constante animación del sistema de engranajes. De esta forma, se genera una ilusión de fluidez en el cambio de un sistema a otro.

Es importante saber que todos los métodos que se utilizan tanto para hacer los emparejamientos como para pintar la transición, se encuentran en la clase **`SmoothTransition`**.

4.3.6. Efecto trayectoria

Como uno de los objetivos de este proyecto es mantener al usuario el mayor tiempo posible interactuando con la aplicación para que así retenga la mayor cantidad de información posible, se ha añadido un efecto visual bastante llamativo. Aprovechando que el sistema de engranajes va trazando una trayectoria anidada, se ha decidido ir **dibujando esa trayectoria** para que se puedan observar las diferentes trazas de las factorizaciones. Este efecto recibe el nombre de ***espirógrafo***(24).

La forma de implementar esta nueva funcionalidad comienza por la adición de un botón que active y desactive este efecto en la pantalla principal. Acto seguido, se han declarado dos elementos nuevos en la clase `DrawingView`, un `Bitmap` y un `Canvas`.

La razón por la que aparecen estos elementos reside en la optimización del rendimiento de la aplicación. Para poder dibujar la trayectoria evitando almacenar el rastro de coordenadas por las que pasa cada engranaje final del sistema, se ha utilizado un `Bitmap`. De esta manera, la trayectoria va activando la coordenada exacta en este elemento del color especificado, sin necesidad de ocupar memoria. Sin embargo, esto generaba un dibujo bastante pixelado debido a la densidad de la malla; para solucionar esto, se ha utilizado el segundo `Canvas` para utilizar sus métodos e ir dibujando círculos, generando una traza uniforme y suavizada.

Por último, este `Bitmap` se dibuja dentro del `Canvas` principal mostrando la traza al usuario mediante la interfaz gráfica. Además, cada vez que desactive esta opción o cambie de factorización, se limpiará el trazado anterior.

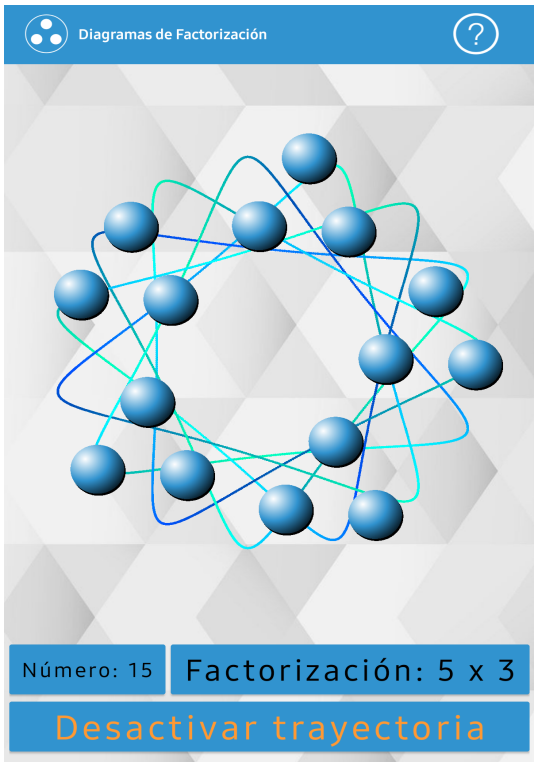


Figura 4.7: Captura realizada del efecto trayectoria.

4.4. Pantalla de información

Esta pantalla está destinada a dos funcionalidades principales: buscador de números y factorización canónica. Para poder acceder a ella, el usuario deberá lanzar el evento que se ha programado en el `GestureListener` nombrado antes como `onLongPress`, el cual dará inicio a una nueva actividad, `infoActivity`.

4.4.1. Campo de búsqueda

En la parte superior de la pantalla se encuentra un campo de introducción de texto para que el usuario inserte el número que desea mostrar en la pantalla principal. De esta manera, se evita que el usuario tenga que deslizar un cierto número de veces hasta llegar al número que realmente está buscando.

Esta funcionalidad tiene como predeterminado el **teclado numérico**, obligando al usuario a meter un caracter de este tipo. Esto se ha implementado desde el propio *layout* de la vista.

Además, contiene un método, `invalidInput` que bloquea el retorno a la pantalla principal si se introduce un número fuera del rango establecido por la aplicación, mostrando un *Toast* en la parte inferior de la pantalla.

Por último, para simplificar la interacción del usuario, se ha mapeado la tecla `OK` para que al pulsarla se ejecute directamente el evento de búsqueda volviendo a la pantalla anterior, si es posible en ese caso. Esto ha sido posible gracias a la constante `TIME_ACTION_DONE` que identifica esta tecla anteriormente nombrada.

4.4.2. Factorización canónica

La segunda parte de esta pantalla está dirigida a la didáctica, ya que en ella se muestra la factorización canónica(4), que es la más utilizada en los diferentes ejercicios matemáticos.

Esto ha sido posible desarrollando una nueva clase llamada `CanonicalFactorization` donde, una vez se recibe el número que se está mostrando en el sistema de engranajes, este es descompuesto mediante un simple algoritmo. El resultado de este algoritmo se almacena en dos listas: **dividendos y divisores**. Ambas listas son plasmadas en dos campos de texto separados por una línea vertical, para así representar esta descomposición como aparece en los libros de matemática básica.

Por último, para aclarar un poco la información, al final de esta pantalla se ha añadido un campo de texto donde se muestran solo los factores del número que están representando.

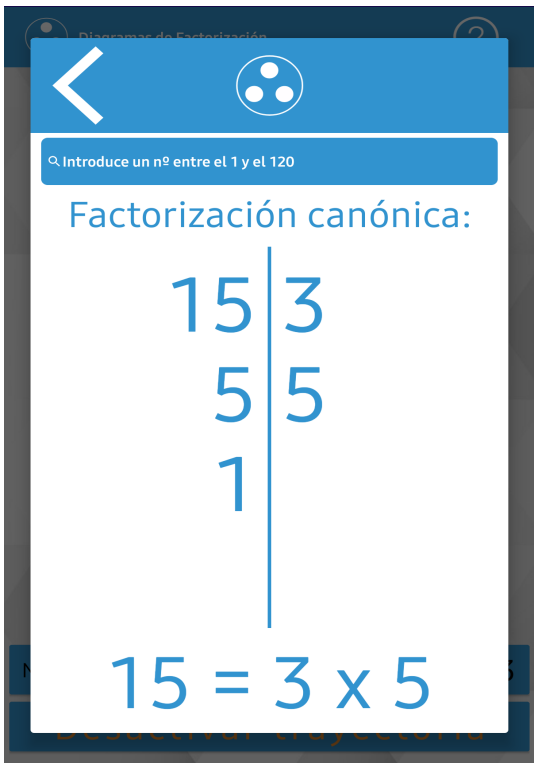


Figura 4.8: Captura realizada del efecto trayectoria.

Capítulo 5

Accesibilidad y guía de usuario

Este apartado es el encargado de tratar todos los aspectos relacionados con el uso de la aplicación. Aunque el proyecto está diseñado para ser bastante sencillo de utilizar, se expondrán todas las funcionalidades de cada pantalla para permitir al usuario aprovechar cada una de ellas.

5.1. Pantalla principal

La pantalla principal cuenta con numerosas funcionalidades, ya que es aquí donde se desarrolla prácticamente toda la actividad de la aplicación. Por ello, permite interactuar con numerosos elementos de la interfaz gráfica. A continuación una lista de las acciones que puede realizar el usuario:

- **Pulsar botón ?**: este botón permite acceder a la pantalla de ayuda.
- **Deslizar de izquierda a derecha**: este gesto permite disminuir en una unidad el número que se está representando en el diagrama. El número más pequeño es el 1.
- **Deslizar de derecha a izquierda**: este gesto permite aumentar en una unidad el número que se está representando en el diagrama. El número más grande es el 120.
- **Deslizar de arriba hacia abajo**: este gesto permite volver avanzar a la factorización siguiente del número actual, siempre que el número contenga más de una descomposición diferente.
- **Deslizar de abajo hacia arriba**: este gesto permite volver a la factorización anterior del número actual. Solo se podrá utilizar si el gesto anterior se ha realizado primero.
- **Mantener pulsado**: este gesto permite acceder a la pantalla de información.
- **Botón trayectoria**: este botón permite activar la funcionalidad que dibuja la trayectoria que va trazando el sistema de engranajes (*apartado 4.3.6.*).

Como se puede observar, la interacción con la interfaz gráfica es bastante simple puesto que no es conveniente que una herramienta didáctica esté sobrecargada de elementos. De esta manera, el usuario se puede centrar en lo realmente importante que es el diagrama de factorización.

5.2. Pantalla de instrucciones

La pantalla de instrucciones cuenta con un pequeño resumen de este mismo apartado, para que el usuario pueda acceder a esa información en todo momento. La única interacción que posee es **pulsar la pantalla**, de esta manera, se cerrará y se dejará paso a la pantalla principal.

5.3. Pantalla de información

La pantalla de información tiene una función complementaria para facilitar el uso de la aplicación y aportando información relevante, como lo es la factorización canónica de la factorización actual.

La lista de acciones que se pueden realizar es la siguiente:

- **Pulsar flecha superior**: este botón permite regresar a la pantalla principal.
- **Pulsar barra de búsqueda**: esta barra de búsqueda despliega el teclado numérico que permite introducir una cifra para su posterior búsqueda.
- **Tecla OK del teclado predeterminado**: esta tecla permite lanzar la búsqueda del número introducido en el elemento comentado en el apartado anterior.

La interacción en esta pantalla es bastante más reducida debido a su carácter meramente informativo. Sin embargo, realiza una función bastante útil a ojos del usuario, ya que evita tener que estar deslizando reiteradamente para acceder a un número relativamente lejano al actual. Por lo que se ahorra en tiempo y aumenta la comodidad de uso.

Capítulo 6

Conclusiones y líneas futuras

6.1. Conclusiones

La aplicación, en su última versión(25), cumple con todos los objetivos propuesto al comienzo del proyecto. Además, se han mejorado las expectativas iniciales tanto a nivel de diseño como puede ser con la implementación de la *splash screen* o la pantalla de instrucciones, como a nivel funcional, ya que la pantalla de información no estaba contemplada desde un principio.

Por otro lado, aunque se trate de un proyecto a corto plazo, el acabado de la aplicación en cuestiones de diseño es bastante armónico. Además, pese al la cantidad de operaciones que realiza internamente, se ha conseguido tener un buen rendimiento que permita una muy buena fluidez visual.

Por último, **el resultado de la fusión de términos como la factorización numérica y la recursividad reside en una herramienta didáctica completamente funcional**. Además, es una opción gratuita y completamente al alcance de cualquier usuario que se pueda permitir un dispositivo Android.

6.2. Líneas futuras

La aplicación en este momento se da por finalizada en su primera versión, no obstante siempre queda abierta a incorporar nuevas mejoras y funcionalidades en el futuro. Además, durante todo el proceso de desarrollo y durante el proceso de testeo, los usuarios no han sido el público objetivo, por lo que sería interesante que tanto un especialista en didáctica como alumnos de centros educativos probaran la aplicación. De esta manera, las reseñas que se obtendrían serían mucho más relevantes para aplicar cualquier cambio en el proyecto. Sin embargo, se ha elaborado una pequeña lista de mejoras futuras que podrían mejorar la interactividad con el usuario:

- **Añadir panel de botones**. Aunque los gestos sean la manera más cómoda de interactuar con la aplicación, se podría incorporar una serie de botones que plasmen

el comportamiento de los gestos que hemos definido de ”deslizar”. De esta manera, el usuario tendría diversas maneras de navegar por las distintas factorizaciones.

- **Añadir un botón de cambio de velocidad**. El poder aplicar cambios de velocidad en un sistema que traza un movimiento armónico siempre genera un gran interés en los más jóvenes. Por ello, se podría añadir esta opción para que el usuario se mantenga más tiempo interactuando con la aplicación.
- **Añadir un modo de juego**. Los juegos siempre han sido grandes pasatiempos, y qué mejor que realizarlo sobre las matemáticas. Lo que se pretende es generar una nueva pantalla que se comporte como una especie de trivial. Aquí, al usuario le van apareciendo preguntas relacionadas con las diferentes descomposiciones de los números. En dicha pregunta el usuario tendrá tres opciones diferentes para responder. De esta manera, en las aulas se podrá utilizar este método para realizar competiciones entre alumnos incentivando así el aprendizaje.

Capítulo 7

Conclusions and future development

7.1. Conclusions

The application, in its latest version(25), achieves all the objectives proposed at the beginning of the project. In addition, the initial expectations have been improved both at the design level, such as with the implementation of the splash screen or the instructions screen, and at the functional level, since the information screen was not contemplated from the beginning.

On the other hand, although it is a short-term project, the application's result is quite harmonious in terms of design. In addition, despite the number of operations it performs internally, it has managed to have a good performance that allows a very good visual fluidity.

Finally, the result of the fusion of terms such as numerical factorization and recursion lies in a fully functional teaching tool. In addition, it is a free option and completely available to any user who can afford an Android device.

7.2. Future development

The application at this time is terminated in its first version; however, it is always open to incorporate new improvements and features in the future. In addition, throughout the development and testing process, users have not been the target audience, so it would be interesting for both didactics specialist and students of educational centers to test the application. In this way, the reviews that would be obtained would be much more valuable to apply any changes in the project. Here is a small list of future implementations that could improve user interactivity:

- **Add button panel**. Although gestures are the most comfortable way to interact with the application, a few buttons that mimic the behavior of the "slide" gestures could be incorporated. In this way, the user would have different ways to navigate through the different factorizations.

- **Add a speed change button**. Being able to apply speed changes in a system that traces a harmonic movement always generates great interest in the youngest. Therefore, this option could be added so that the user stays longer interacting with the application.
- **Add a game mode**. Games have always been great hobbies, and what a better way to play them than in a math app. What is intended is to generate a new screen that behaves as a kind of trivial. Here, the user will appear questions related to the different decompositions of the numbers. In each question the user will have three different options to answer; in this way, this method can be used in classrooms to carry out competitions between students, thus encouraging learning.

Capítulo 8

Presupuesto

8.1. Licencias de software

Nombre del software	Coste	Anotaciones
Android	0.00 €	Gratuito. Código libre
Android Studio	0.00 €	Gratuito. Código libre
Git	0.00 €	Gratuito. Código libre
Java (OpenJDK)	0.00 €	Gratuito. Código libre
Kotlin	0.00 €	Gratuito. Código libre
LaTeX (pdfLaTeX)	0.00 €	Gratuito. Código libre
Overleaf	0.00 €	Gratuito. Código libre
InkScape	0.00 €	Gratuito. Código libre
Trello	0.00 €	Plan Gratuito
Widnows 10	0.00 €	Coste incluido en el precio del equipo

Cuadro 8.1: Coste de las licencias de software utilizadas.

8.2. Materiales

Material	Valor	Factor de amortización	Coste
Equipo sobremesa	1.057,38€ €	5 %	52,87 €
Samsung Tab A7	179 €	5 %	8.95 €
Total:			61.82 €

Cuadro 8.2: Coste de los materiales utilizados

8.3. Costes de personal

Asumiendo un coste por hora de 20 €.

Tarea	Horas	Coste
Investigación sobre la factorización y buscar apps en el mercado	6	120,00 €
Desarrollo de la parte lógica	150	3.000,00 €
Desarrollo de la GUI	90	1.800,00 €
Testeo de la aplicación	32	640,00 €
Elaboración de la memoria	30	600,00 €
Total	300	6.040,00 €

Cuadro 8.3: Costes de personal.

8.4. Presupuesto final del proyecto

Motivo	Coste
Licencias de software	0.00 €
Materiales	61.82 €
Mano de obra	6.040,00 €
Coste	6.101,82 €
Beneficio (6 %)	366,11 €
Presupuesto total del proyecto	6.467,93 €

Cuadro 8.4: Presupuesto final del proyecto.

Apéndice A

Códigos

A.1. Clase Gear

```
1  package com.ull_tfg.diagramfactorization
2
3  import android.content.Context
4  import android.graphics.*
5  import kotlin.math.cos
6  import kotlin.math.sin
7
8  class Gear(
9    private var context: Context,
10   center: Point,
11   radius: Float,
12   speed: Float,
13   isTerminal: Boolean,
14   drawPerimeter: Boolean,
15   level: Int,
16 ) {
17
18   companion object {
19     private const val MAX_LEVEL_INCREASE_SPEED = 2
20     const val PERCENTAGE_SIZE = 0.60F
21     private const val DRAW_PERIMETER = false
22   }
23
24   var center: Point = Point(0F, 0F)
25   var radius: Float = 0F
26   private var speed: Float = 0F
27   private var phase: Float = 0F
28   var isTerminal: Boolean = false
29   private var drawPerimeter: Boolean = false
30   var children = ArrayList<Gear>()
31   private var level: Int = 0
32   private var elementSize = 0F
33
34   init {
35     this.center = center
36     this.radius = radius
37     this.speed = speed
```

```
38      this.isTerminal = isTerminal
39      this.drawPerimeter = drawPerimeter
40      this.level = level
41      this.elementSize = radius * PERCENTAGE_SIZE
42    }
43
44    private fun drawPhase(canvas: Canvas, originPoint: Point, destinyPoint: Point)
      ↪ {
45      canvas.drawLine(originPoint.x, originPoint.y, destinyPoint.x, destinyPoint.y,
        ↪ Paint())
46    }
47
48    private fun drawPerimeter(canvas: Canvas) {
49      val paint = Paint()
50      paint.strokeWidth = 2F
51      paint.style = Paint.Style.STROKE
52      paint.pathEffect = DashPathEffect(floatArrayOf(20f, 40f), 0f)
53      canvas.drawCircle(center.x, center.y, radius, paint)
54    }
55
56    private fun drawCenter(canvas: Canvas) {
57      if (isTerminal) {
58        val pointerCenter = CircumferenceElement(center, elementSize)
59        pointerCenter.printCircle(canvas)
60      }
61    }
62
63    private fun calculatePhaseNewPosition(): Point {
64      return Point(
65        /* X */ center.x + radius * cos(phase.toDouble()).toFloat(),
66        /* Y */center.y - radius * sin(phase.toDouble()).toFloat()
67      )
68    }
69
70    private fun gearTrajectory(frame: Int) {
71      phase = frame * speed
72    }
73
74    fun updateGear(frame: Int) {
75      gearTrajectory(frame)
76      calculateChildNewPosition()
77      for (gearChild in children) {
78        gearChild.updateGear(frame)
79      }
80    }
81
82    fun draw(canvas: Canvas, frame: Int) {
83      if (drawPerimeter) {
84        drawPerimeter(canvas)
85        drawPhase(canvas, center, calculatePhaseNewPosition())
86      }
87      drawCenter(canvas)
88      for (gearChild in children) {
89        gearChild.draw(canvas, frame)
90      }
91    }
92
93    private fun calculateChildRadius(): Float {
```

```
94      val alpha = Math.PI / children.size
95      return (sin(alpha) * radius / (1 + sin(alpha))).toFloat()
96    }
97
98    private fun calculateCenterChild(iterator: Int, childRadius: Float): Point {
99      return Point(
100       /* X */ (center.x + (radius - childRadius) * cos(iterator * 2 * Math.PI /
          ↪  children.size)).toFloat(),
101       /* Y */ (center.y - (radius - childRadius) * sin(iterator * 2 * Math.PI /
          ↪  children.size)).toFloat()
102     )
103   }
104
105   private fun addChild(terminalGear: Boolean) {
106     children.add(Gear(context, Point(0F, 0F), 0F, 0F, terminalGear,
        ↪  DRAW_PERIMETER, 0))
107     val childRadius = calculateChildRadius()
108     for (iterator in children.indices) {
109       val childCenter = calculateCenterChild(iterator, childRadius)
110       val newLevel = level + 1
111       val newSpeed = calculateChildSpeed(newLevel)
112       children[iterator] = Gear(
113         context,
114         childCenter,
115         childRadius,
116         newSpeed,
117         terminalGear,
118         DRAW_PERIMETER,
119         newLevel
120       )
121     }
122   }
123   private fun calculateChildSpeed(newLevel: Int): Float {
124     val newSpeed = if (newLevel < MAX_LEVEL_INCREASE_SPEED){
125       -(speed * 2)
126     } else {
127       -speed
128     }
129     return newSpeed
130   }
131
132   private fun calculateChildNewPosition() {
133     for ((id, child) in children.withIndex()) {
134       child.center = Point(
135         /* X */ (center.x + (radius - child.radius) * cos(phase + id * 2 *
            ↪  Math.PI / children.size)).toFloat(),
136         /* Y */ (center.y - (radius - child.radius) * sin(phase + id * 2 *
            ↪  Math.PI / children.size)).toFloat()
137       )
138     }
139   }
140
141   fun createGearFactorization(numberFactorization: ArrayList<Int>) {
142     val childNumberIterator = 0
143     isTerminal = false
144     children.clear()
145     if (numberFactorization.first() != 1) {
146       recursiveAddGearChild(childNumberIterator, this, numberFactorization)
```

```
147       } else {
148         isTerminal = true
149       }
150     }
151
152     private fun recursiveAddGearChild(childNumberIterator: Int, currentGear: Gear,
        ↪  numberFactorization: ArrayList<Int>) {
153       var terminalGear = false
154       if (childNumberIterator >= numberFactorization.size) {
155         return
156       }
157       if (childNumberIterator == numberFactorization.size - 1) {
158         terminalGear = true
159       }
160       for (iterator in 1..numberFactorization[childNumberIterator]) {
161         currentGear.addChild(terminalGear)
162       }
163       for (gearChild in currentGear.children) {
164         recursiveAddGearChild(childNumberIterator + 1, gearChild,
          ↪  numberFactorization)
165       }
166     }
167
168     fun getCurrentCoordinates(list: ArrayList<Gear>): ArrayList<Gear> {
169       for (gearChild in children) {
170         gearChild.getCurrentCoordinates(list)
171       }
172       if (isTerminal) {
173         list.add(this)
174       }
175       return list
176     }
177
178     fun getPositionOfFirstLeaf(): Point {
179       if (isTerminal) {
180         return center
181       }
182       return children[0].getPositionOfFirstLeaf()
183     }
184
185 }
186
```

Bibliografía

[1] I. C. P. y Ángel Puentes Puente, "Innovación educativa: uso de las tic en la enseñanza de la matemática básica," 2012. https://helvia.uco.es/handle/10396/11641.

[2] U. E. F. Valencia, "Utilización de recursos didácticos interactivos a través de las tic en el proceso de enseñanza aprendizaje en el área de matemática," 2017. https://dialnet.unirioja.es/servlet/articulo?codigo=6119349.

[3] "Software libre y matemáticas." http://www.educacontic.es/blog/software-libre-y-matematicas.

[4] Wikipedia, "Factorización de números enteros." https://es.wikipedia.org/wiki/Factorizaci%C3%B3n_de_enteros#:~:text=En%20teor%C3%ADa%20de%20n%C3%BAmeros%2C%20la,multiplican%20dan%20el%20n%C3%BAmero%20original.

[5] "Link descarga prime numbers and factors." https://play.google.com/store/apps/details?id=com.jns.PrimeNumbers.

[6] "Página oficial de android studio." https://developer.android.com/studio?hl=es&gclid=Cj0KCQjwjvaYBhDlARIsAO8PkE2mctTNqbBLPspvWxL2ZrtfZVbPArQOEIXwcB&gclsrc=aw.ds.

[7] "Patrón de arquitectura de software mvc." https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador.

[8] "Documentación clase view, kotlin." https://developer.android.com/reference/kotlin/android/view/View.

[9] "Documentación clase gesturedetector, kotlin." https://developer.android.com/reference/android/view/GestureDetector.

[10] "Página oficial de latex y overleaf." https://es.overleaf.com/.

[11] "Página oficial de inkscape." https://inkscape.org/es/.

[12] "Imágenes vectoriales." https://es.wikipedia.org/wiki/Gr%C3%A1fico_vectorial.

[13] "Página oficial de telegram." https://telegram.com.es/.

[14] S. G. Sotomayor, "Metodologías ágiles." https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/.

[15] "Página oficial de bitbucket." https://bitbucket.org/product?&aceid=&adposition=&adgroup=92542398455&campaign=9128560695&creative=414608949975&device=c&keyword=bitbucket&matchtype=e&network=g&placement=&ds_kids=p51241296666&ds_e=GOOGLE&ds_eid=700000001551985&ds_e1=GOOGLE&gclid=Cj0KCQjwjvaYBhDlARIsAO8PkE3vBq4xJ3WZ9cLCIGuwtFF3XzFaEgGXfNObZFp4kesIhqqwcB&gclsrc=aw.ds.

[16] "Página oficial de git." https://git-scm.com/.

[17] "Página oficial de trello." https://trello.com/es.

[18] "Documentación clase assetsmanager, kotlin." https://developer.android.com/reference/android/content/res/AssetManager#open(java.lang.String).

[19] "Documentación clase bufferedreader, kotlin." https://developer.android.com/reference/kotlin/java/io/BufferedReader.

[20] Wikipedia, "Frame." https://es.wikipedia.org/wiki/Fotograma.

[21] Wikipedia, "Movimiento circular." https://en.wikipedia.org/wiki/Circular_motion.

[22] Wikipedia, "Rgba." https://es.wikipedia.org/wiki/Espacio_de_color_RGBA.

[23] Wikipedia, "Lerp." https://en.wikipedia.org/wiki/Linear_interpolation.

[24] Wikipedia, "Espirógrafo." https://es.wikipedia.org/wiki/Espir%C3%B3grafo.

[25] "Repositorio con la apk." https://github.com/alu0101014319/apkFactorizationDiagram.git.