ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: DISEÑO Y DESARROLLO DE UNA APP PARA EL ENTRENAMIENTO MATEMÁTICO DE NIÑOS

AUTOR: RAQUEL DE LOS FRAILES HARRISON

TITULACIÓN: GRADO EN INGENIERÍA DE SONIDO E IMAGEN

TUTOR: VÍCTOR JOSÉ OSMA RUÍZ

DEPARTAMENTO: TEORÍA DE LA SEÑAL Y COMUNICACIONES

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: ELOY PORTILLO ALDANA

VOCAL: VÍCTOR JOSÉ OSMA RUÍZ

SECRETARIO: RUBÉN FRAILE MUÑOZ

Fecha de lectura:

Calificación:

El Secretario,

AGRADECIMIENTOS

Primeramente, me gustaría agradecer a mi tutor Víctor José Osma por darme la oportunidad de ser participe en este proyecto, el cual me hace mucha ilusión participar. Siempre he creído en la necesidad de educar de forma correcta y poder aportar mi grano de arena es y ha sido todo un honor.

A los profesores que ponen toda su ilusión en la enseñanza que hacen que sus alumnos recuperen la ilusión y el amor por lo que se están formando.

A mis padres, por darme la oportunidad de educarme en la Universidad Politécnica de Madrid y aguantarme todos estos años para por fin terminar mi carrera sin meterme presión en ningún momento.

A mi hermana Michelle, por acompañarme en esta etapa final que juntas hemos superado.

A mi hermana Laura, por insistir en que terminara de una vez y ser ejemplo de superación en su etapa educativa, profesional y personal.

A mi hermano Alex, por ser símbolo de dedicación, autosuperación y enseñarme a que se puede sentir verdadera pasión por un trabajo.

A mis muy mejores amigos, que me han apoyado en muchos momentos de desilusión por mi futuro profesional y animarme a seguir formándome y no rendirme.

A Andrés y su familia, por vivir junto a mí todos los exámenes complicados y que tanto me han costado aprobar. Por las velitas.

A Ron, Pongo, Lolo, Duna y Sally, por darme la compañía que necesitaba todas esas noches largas estudiando y finalizando trabajos.

A toda la gente que pone su conocimiento de manera desinteresada a disposición de los demás que hacen que adquirir cualquier tipo de conocimiento sea muy sencillo y rápido.

A todos ellos, gracias.

La diferencia entre ganar y perder a menudo consiste en abandonar (Walt Disney)

RESUMEN

Actualmente, las nuevas tecnologías son un elemento imprescindible en las escuelas. Al igual que hace 20 años no faltaban diccionarios y enciclopedias en las aulas, raro es que ahora no se encuentre un ordenador con conexión a internet en el que poder consultar. Incluso las pizarras que llenaban entonces todo de tiza, ahora son electrónicas y necesarias para la enseñanza actual.

Bien es cierto que el exceso de las nuevas tecnologías en niños y niñas de primaria puede suponer un riesgo si no se utilizan correctamente. En este proyecto, se quiere aprovechar el auge de las TIC (Tecnologías de la Información y la Comunicación) en el ámbito de la educación para desarrollar unos juegos que no solo aporten diversión a los alumnos, sino que además les permitan aprender conceptos que no pueden ser adquiridos mediante ejercicios clásicos en las aulas. Es por todo esto, que se ha diseñado y programado un primer juego de operaciones donde el usuario deba introducir no solamente el resultado de una ecuación, sino que pruebe niveles con mayor dificultad donde falte un operando u operador y aplique y asienta de forma distinta el aprendizaje de las operaciones matemáticas; y un segundo y tercer juego, donde el alumno aprenda a ser capaz de, a simple vista, determinar la longitud de un segmento a tamaño real.

En este documento, se incluyen las herramientas de desarrollo utilizadas como son el programa Visual Studio, el lenguaje C# y el entorno de desarrollo Xamarin.

Después, se incluyen los diagramas de flujo que el usuario espera encontrar, diagramas UML (Lenguaje Unificado de Modelado) de las clases y librerías que han sido necesarias y una matriz de verificación. De manera genérica, se proporciona al lector el proceso que se ha elegido para la creación de la APP (aplicación) y, más en detalle, cada uno de los juegos indicando las librerías que han sido necesarias, el diseño que se ha elegido y los problemas que se han ido encontrando junto con la solución desarrollada. Además, se incluye una explicación sobre el manejo y uso de la librería *MPAndroidChart*, utilizada para la muestra de datos almacenados en distintos tipos de gráficas, y una explicación de la base de datos interna utilizada y los elementos que se han decidido almacenar para posibles versiones posteriores de la aplicación.

Para finalizar, se describen las posibles líneas futuras que se pueden tomar a partir de esta aplicación y mejoras que podrían ser beneficiosas para posteriores juegos educativos.

ABSTRACT

Nowadays, new technologies are an indispensable element at the schools. Same as 20 years ago dictionaries and encyclopedias remained in the classrooms, now it's difficult to find a classroom without a computer with internet connection where all students can consult their questions. Even the blackboards that used to fill everything with chalk, now are electronic and necessary for the current education.

Although it is true that the excessive use of new technologies in primary-school children my pose a risk if they don't use them correctly. In this project, we want to take advantage of the increasing use of ICT (Information and Communication Technology) developments in education to develop three games that not only amuse the students, but also teach them concepts that could not be acquired with classic exercises at school. It is for all that, which has been designed and programmed a first game of operations where the user should introduce not only the result of an equation, but also tries different levels with major difficulty where one operand or the operator are missing so he or she applies and learns in a different way their knowledge of mathematics; and a second and third games where the students can learn to determine the real length of a segment with the naked eye.

In this document, the development tools used to perform the games are included, such as Visual Studio, C# programming language or the development environment Xamarin.

After that, you can find flow charts, UML (Unified Modeling Language) diagrams and a verification matrix where all the classes, libraries, and everything that the user could find during his or her experience with the APP (application) are explained. In a general way, the reader of this project can understand how the APP has been created and, more in details for each game, the design chose, the libraries and all the problems that have appeared programming with their corresponding solutions. Also, an explanation of the *MPAndroidChart* library is included, a library that has been used for the internal data storage (explaining which elements of each game have been saved) and the show of the user's results in different types of graphs.

In conclusion, the possible future lines from this application and some improvements that could enhanced further educational games are defined at the end of this document.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTA DE ECUACIONES

LISTA DE SIGLAS Y ACRÓNIMOS

API: Interfaz de Programación de Aplicaciones (Application Programming Interface)

APP: Abreviatura de la palabra en inglés *application* (aplicación)

BBDD: Base de Datos

IDE: Entorno de Desarrollo Integrado (Integrated Development Environment)

IEC: Comisión Electrotécnica Internacional (International Electrotechnical Commission)

ISO: Organización Internacional para la Estandarización (International Organization for Standardization)

PC: Computadora Personal (Personal Computer)

RAM: Memoria de Acceso Aleatorio (Random Access Memory)

SLAT: Traducción de Direcciones de Segundo Nivel (Second Level Address Translation)

SSD: Unidad de Estado Sólido (Solid-State Drive)

TIC: Tecnologías de la Información y la Comunicación

UML: Lenguaje Unificado de Modelado (Unified Modeling Language)

1. INTRODUCCIÓN

La necesidad de adaptar la educación actual al siglo XXI, no implica sustituir a los profesores de los centros por un ordenador o Tablet que muestre a los alumnos tutoriales con los que aprender o juegos con los que adquirir conceptos, sino que implica utilizar los recursos tecnológicos de los que disponemos, a los que la mayoría de los alumnos están más que acostumbrados, de tal forma que las clases magistrales se apoyen en estas herramientas y, además, enseñe a los alumnos el correcto uso de las mismas y todos sus beneficios.

Es posible que los profesores que lleven más años en la profesión sean más reacios a este inminente y necesario cambio. Es por eso por lo que la integración de las TIC en las aulas no debe hacerse a la ligera e implantarse sin más, sino que es imprescindible que se forme a los profesores y se explique detalladamente todos los beneficios que estas herramientas pueden ofrecerles, sobre todo, la mejoría en cuanto a la evaluación y formación personalizada que se puede realizar de cada alumno y alumna de tal forma que se adapte a las necesidades de cada uno de ellos [1]. Con estos recursos, se facilita que los centros escolares atiendan a la diversidad del ser humano, tanto para alumnos que presenten más dificultades a la hora de adquirir conceptos como para alumnos que adquieran conceptos de manera más rápida y se pueda aprovechar su capacidad de aprendizaje [2].

No obstante, un mal uso de las TIC puede ocasionar desventajas como son la cantidad de información errónea que puede encontrarse en internet o las adicciones que puede acarrear un uso excesivo de ellas, de ahí la necesidad de formar correctamente a los docentes y, por consecuencia, a los alumnos [3].

Una de las múltiples ventajas que existen al utilizar las TIC en el ámbito educativo, es que pueden diseñarse juegos para el aprendizaje, lo cual resulta muy atractivo para los niños y niñas de educación primaria. Actualmente, existen numerosos juegos de aplicaciones matemáticas que pueden descargarse, pero, realizando un breve estudio de los encontrados, detectamos que entorno al 90% de los primeros 30 juegos que se pueden descargar, consisten en introducir el resultado de operaciones numéricas, lo cual puede ser un buen instrumento de repaso, pero refleja la poca variedad de ejercicios disponibles para todos los conceptos que engloban las matemáticas.

El objetivo principal de este proyecto es diseñar y desarrollar aplicaciones software para dispositivos portátiles tipo Tablet que permitan, a alumnos de primeros cursos de Educación Primaria, la realización de actividades que contribuyan al desarrollo de funciones ejecutivas y del sentido numérico. Para conseguirlo, se divide el objetivo en los siguientes objetivos específicos de cara a que la realización del proyecto se realice de manera productiva:

1. Estudiar la programación Android y sus posibilidades.
2. Definir los requisitos de usuario y las características que deben tener cada uno de los juegos.
3. Desarrollar la aplicación:
    i. Diseñar una apariencia atractiva para niños.
    ii. Programar los juegos y probar su funcionamiento.
4. Comprobar y resolver problemas de diseño mediante el uso de personas ajenas al proyecto.

Siguiendo los objetivos específicos planteados, lo primero que se describe en este proyecto son los antecedentes y el estado actual del uso de las TIC en las aulas, así el lector conoce la situación en la que nos encontramos.

Después, se describen las herramientas de desarrollo utilizadas que son el IDE (Entorno de Desarrollo Integrado) Visual Studio, el lenguaje C# y Xamarin, un entorno de desarrollo de aplicaciones multiplataforma.

En la sección "descripción de la solución propuesta", se recoge todo el desarrollo realizado para la correcta funcionalidad de los juegos. Primero se indican las características que se han utilizado en la elaboración de la aplicación y las necesidades por parte de un usuario para la utilización de la misma. Además, se incluyen diagramas de flujo y UML de la estructura de la aplicación para una mejor comprensión de la arquitectura y librerías utilizadas. Después se describen, en términos generales, la estructura y los pasos seguidos durante el tiempo de desarrollo del proyecto y, uno por uno, cada juego que contiene la aplicación, explicando el diseño utilizado y las principales clases y parámetros para su correcto funcionamiento. Al final, se enumeran las librerías utilizadas para el almacenamiento de los resultados en una base de datos interna y las gráficas de resultados que aparecen cada vez que se finaliza una sesión en cada uno de los juegos.

Una vez definida con detenimiento la aplicación, se finaliza con una breve conclusión y las posibles líneas futuras que se pueden seguir para que el objetivo y el contexto en los que se sustenta este proyecto, no finalicen y se sigan desarrollando aplicaciones que mejoren la situación actual de la educación primaria.

Se anexa junto con la memoria, un manual de instalación y uso de la aplicación de cara a un posible estudio real de funcionalidad y los presupuestos que han ocasionado la realización del proyecto.

2. ANTECEDENTES

A lo largo de los últimos años, la aplicación de las TIC en la educación ha ido aumentando notablemente. De hecho, ya desde el año 2015, docentes y expertos coincidían absolutamente en afirmar que antes del 2018 la mayoría de las aulas dispondrían, si no disponían ya, de Wifi, de proyector y/o pizarra digital (por tanto, de como mínimo un PC-Computadora Personal) y, a partir del curso 2018-19, se iría generalizando el uso de sistemas de producción audiovisual y de tabletas [4]. Estos últimos, de tamaño y precio más reducido que los tradicionales ordenadores personales, ha posibilitado el tránsito del concepto de "aula de ordenadores" al de "ordenadores en el aula" [5]. Además, cuando se han conseguido incorporar en las aulas de Educación Primaria, han producido un impacto positivo sobre el aprendizaje colaborativo y el intercambio de información [6] gracias a la pantalla táctil, que proporciona una interfaz en la que el usuario tiene una sensación mucho más próxima a la manipulación de objetos. Este hecho, unido a un diseño intuitivo de la parte gráfica de la interfaz, ha hecho de los dispositivos portátiles tipo Tablet sistemas mucho más usables por niños pequeños [7].

Actualmente, existen numerosos estudios sobre si el uso de estas nuevas tecnologías afecta o no al aprendizaje de niños, pero, en general, no se han obtenido más que contradicciones y ningún resultado claro. Por tanto, sería razonable pensar que no es el uso de la tecnología en sí lo que afecte en el aprendizaje, sino las actividades desarrolladas en ellos lo que pueda provocar que un alumno, a través del dispositivo, aplique una metodología de trabajo y ponga en juego unas capacidades concretas para desarrollar tales actividades [8].

En este sentido metodológico, una técnica muy utilizada es la denominada *Gamification*, que consiste en la utilización de videojuegos mediante los cuales un alumno pueda aprender inconscientemente, es decir, el alumno juega únicamente por diversión, pero en el proceso adquiere todas las capacidades educativas programadas sin apenas darse cuenta. Para ello, existen ciertas características que pueden influir en el incremento de la motivación [9]:

- La adquisición de puntos que actúan como refuerzos positivos
- La selección de avatares o la posibilidad de configuración del personaje que influyen en la implicación personal en el objetivo y sentimiento de autonomía.
- Tablas con las mejores puntuaciones que despierta competitividad, sentimientos de mejora de la capacidad individual o de grupo o colaboración. No obstante, esta característica puede causar desmotivación para los que se encuentren al final de la lista por lo que hay que ser muy cuidadosos en su utilización.
- Otros elementos como recompensas, barras de progreso, gráficos de evolución, retos; pueden hacer que el usuario sea consciente de su progreso y desarrollo.

En este proyecto, se pretenden desarrollar aplicaciones Android con ciertos elementos de *Gamification* como los descritos anteriormente, que puedan ayudar a los niños en su proceso del aprendizaje de las matemáticas en los primeros años de la Educación Primaria (6-7 años). Una de esas técnicas se puede apreciar al finalizar cada uno de los juegos, donde se representa mediante gráficas los resultados de, únicamente, la última sesión realizada, así un alumno podría sacar una nota alta aun habiendo realizado el ejercicio con anterioridad y obteniendo un resultado negativo. Además, cuando un alumno falla en algún ejercicio de distancias, aparece por pantalla la distancia que ha dibujado para que pueda ir adquiriendo el concepto de forma progresiva.

El aprendizaje de las matemáticas está muy condicionado por el grado de desarrollo previo de dos tipos de capacidades: las *funciones ejecutivas* y el *sentido numérico* [10]. Se conoce con el nombre de "funciones ejecutivas" a un conjunto de funciones mentales genéricas que regulan los procesos cognitivos.

Frecuentemente se identifican tres funciones ejecutivas básicas [11]:

- La memoria de trabajo, que permite el mantenimiento y actualización de los datos con que se trabaja.
- La capacidad de adaptación a diferentes tareas, datos y requisitos.
- El autocontrol o capacidad de inhibición de respuestas automáticas.

Por su parte, las capacidades que conforman el sentido numérico se pueden agrupar en las siguientes áreas [12]:

- Cuenta: conocer la secuencia de números, los conceptos de cardinalidad, etc.
- Conocimiento de los números: discriminar cantidades y comparar magnitudes.
- Transformación de números, mediante operaciones de suma y resta, cálculo en contexto verbal.
- Estimación de tamaños, uso de puntos de referencia.
- Patrones numéricos: copia y compleción de patrones, identificación de relaciones numéricas.

Estudios recientes, como el de Geary [13], son coincidentes en señalar este grupo de capacidades como básicas para el desarrollo posterior de competencias matemáticas, tal vez añadiendo alguna otra como la velocidad de proceso mental de la información y la capacidad para realizar representaciones visuales-espaciales de los datos. Una falta de atención en el ámbito escolar al desarrollo de estas capacidades básicas, con carácter previo a la introducción de los formalismos propios de las matemáticas, y una presentación de éstos alejada de las experiencias de los alumnos, puede producir paradojas como que haya jóvenes más capaces de resolver problemas de tipo matemático en la vida diaria que de resolver los mismos problemas o similares enunciados en el ámbito académico [14]. Es deseable, por tanto, asegurar el desarrollo de estas capacidades básicas antes de la introducción de los aspectos formales del cálculo matemático y realizar esta introducción de modo que esté estrechamente ligada a las experiencias de los alumnos [15].

En este proyecto se realiza un juego de cálculo numérico donde, por niveles, el alumno debe resolver ecuaciones básicas donde no solo tenga que escribir o elegir un resultado, sino que tenga que completar ecuaciones donde pueda faltar un operando o el operador para completar la ecuación. Este ejercicio fomenta las funciones ejecutivas básicas ayudando al alumno a que memorice las tablas numéricas aprendidas, se adapte a diferentes cambios en cada nivel y deba pensar en varias opciones posibles en algunas ocasiones. En cuanto al sentido numérico, este juego potencia las capacidades de cuenta, conocimiento y transformación de los números que adquieren los alumnos en las aulas. Echando un vistazo a los juegos ya existentes mencionados anteriormente, un gran número de ellos solo trabaja el resultado de la ecuación lo que puede ocasionar que un alumno no piense en la respuesta, sino que la memorice sin comprender el cálculo realizado. Además, en el juego de operaciones diseñado en este proyecto, existe la posibilidad de que en algunos casos haya múltiples posibles respuestas, lo que fomenta el aprendizaje del alumno.

A parte del juego de cálculo numérico, se diseñan dos ejercicios donde el alumno puede ganar perspectiva y aplicar el concepto de distancia a tamaño real. Estos son el juego DIBUJA y el juego ¿CUÁNTO MIDE? donde el alumno debe dibujar o calcular la longitud de una línea indicada por el juego. La estimación de tamaños o uso de puntos de referencia se imparten en las aulas, pero no de una forma realista. Estos ejercicios mejoran las funciones ejecutivas básicas de los alumnos ya que deben memorizar el tamaño que se indica con el mostrado por pantalla, adaptarse a que no siempre puede que tengan que dibujar o calcular la misma distancia ni la línea aparece en la misma dirección, y razonar la respuesta con respecto a sus anteriores

resultados. En cuanto a las capacidades que conforman el sentido numérico, los alumnos potencian la estimación de tamaños, el uso de puntos de referencia y los patrones numéricos con ayuda de estos dos juegos. Además, realizando una búsqueda por la librería de aplicaciones, no se ha encontrado ningún juego ya existente donde se practique este tipo de concepto.

Por último, es importante señalar que este proyecto forma parte de un proyecto mayor cuyo objetivo principal es evaluar si la realización de actividades orientadas al desarrollo del sentido numérico y de funciones ejecutivas utilizando dispositivos portátiles tipo Tablet, contribuye al desarrollo de capacidades relacionadas con las matemáticas en niños de edades correspondientes a los primeros cursos de Educación Primaria. Este proyecto, como se ha mencionado anteriormente, se va a centrar en el diseño y desarrollo de algunas actividades informatizadas basadas en funciones ejecutivas y del sentido numérico que, posteriormente, puedan ser utilizadas para llevar a cabo la citada evaluación global.

3. HERRAMIENTAS DE DESARROLLO

Antes de comenzar con el desarrollo de la solución propuesta, es necesario resaltar el tiempo y estudio previo que es necesario realizar sobre el programa Visual Studio y el lenguaje C# (pronunciado *si sharp* en inglés) para realizar correctamente un proyecto de estas características.

Primero se explica qué es Visual Studio y tres de sus principales características que permiten programar con rapidez y facilidad y los requisitos de sistema necesarios para poder instalar el programa y utilizar algunas de sus características como son el emulador o Xamarin. Después, se describe el lenguaje C# y sus ventajas en su uso como lenguaje para la programación de juegos y, para finalizar, se realiza un breve resumen de Xamarin y sus beneficios.

3.1. IDE Visual Studio 2017

El IDE de Visual Studio es un entorno de desarrollo integrado que posee todas las características para programar en Android, iOS[1], Windows, la Web y la nube. Permite escribir código de forma rápida sin perder el contexto del archivo actual de tal forma que asista en tiempo real a medida que se escribe código, sin importar el lenguaje. Desde su página web[2] puede obtenerse ayuda e información sobre todas sus posibilidades y ventajas de entre las que se destacan las siguientes.

1. **IntelliSense**. Esta herramienta describe las API (Interfaz de Programación de Aplicaciones) mientras el programador escribe y las autocompleta aumentando la velocidad y la precisión. Son muchas las opciones con las que autocompletar ya que puede mostrar métodos, variables, clases creadas, colores, identificadores, propiedades y otras muchas más opciones que se pueden configurar como se observa en la figura 1 [16].

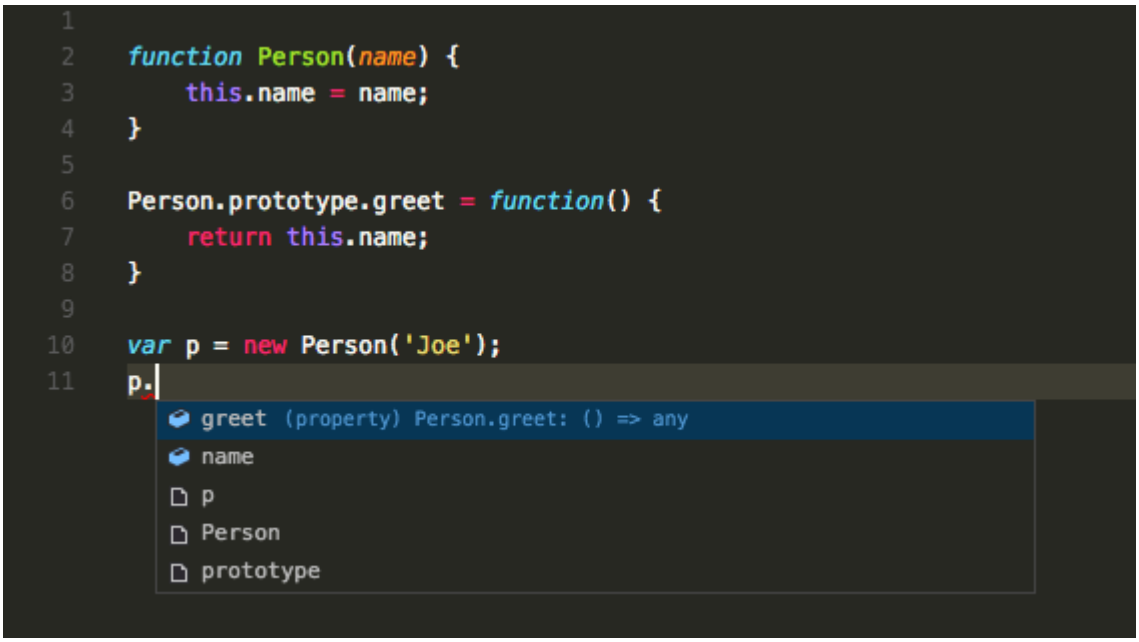

Figura 1. Ejemplo 1 Intellisense obtenido de la referencia [16]

[1] iOS es un sistema operativo móvil de Apple Inc desarrollado para sus dispositivos iPhone, iPod touch y iPad.
[2] https://visualstudio.microsoft.com/es/vs/

Además, a medida que se va escribiendo, se mueve entre las opciones posibles para que aparezcan las más parecidas a los caracteres escritos. Por ejemplo, suponemos que el programador quiere utilizar la variable *sizeOfABuffer*, tan solo con escribir las tres primeras letras, tal y como se muestra en la figura 2, la API se mueve directamente a las variables que comienzan por dichas letras encontrando las opciones con mayor velocidad, lo que proporciona rapidez a la hora de programar [17].

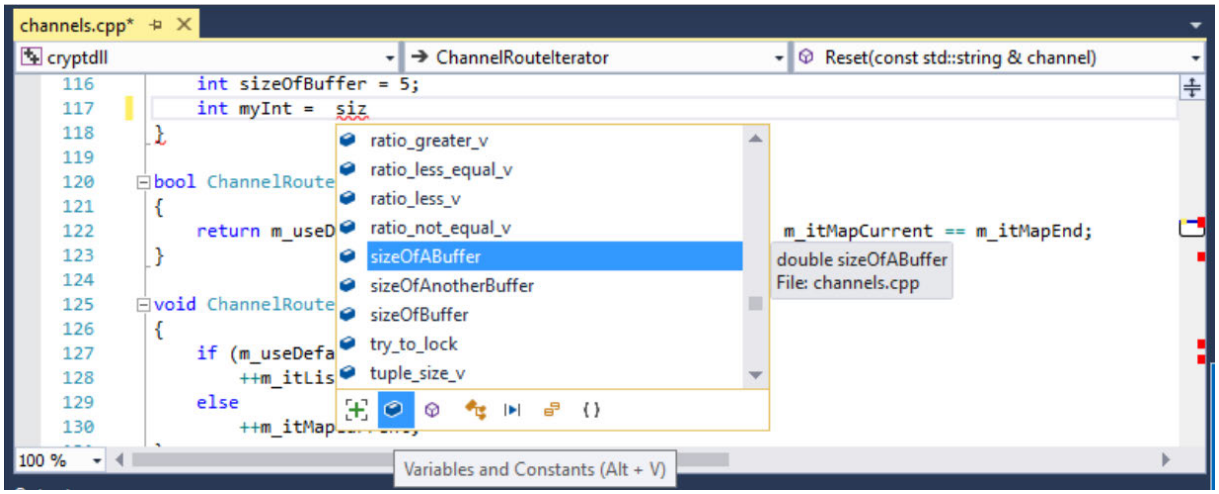

Figura 2. Ejemplo 2 *IntelliSense* obtenido de la referencia [17]

Otra de las opciones que permite *Intellisense* es la que puede apreciarse en la figura 3, a la hora de escribir el nombre de una clase te muestra las variables que debes añadir y, en el caso de que haya varias opciones, te permite seleccionar la que desees de tal forma que los parámetros que añadas sean correctos [18].

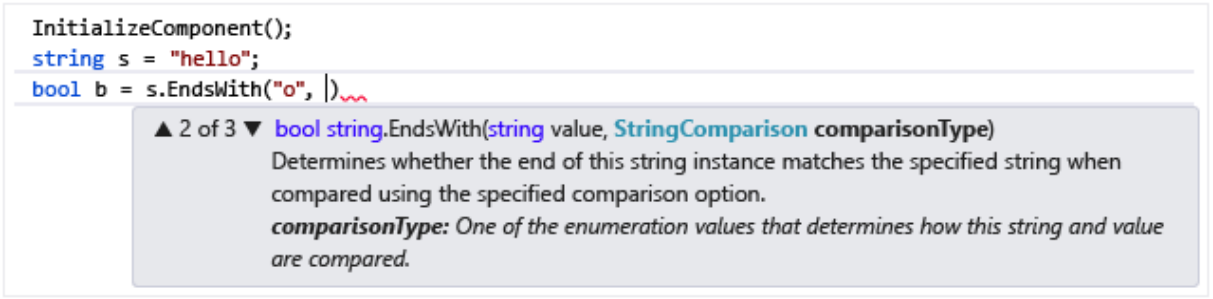

Figura 3. Ejemplo 3 *Intellisense* obtenido de la referencia [18]

2. Los iconos de bombilla como el que se observa en la figura 4 ayudan a identificar y corregir problemas de codificación y se muestran a medida que se programa. Además, permite realizar acciones rápidas en el código desde el editor, como la refactorización y la implementación de interfaces [17].

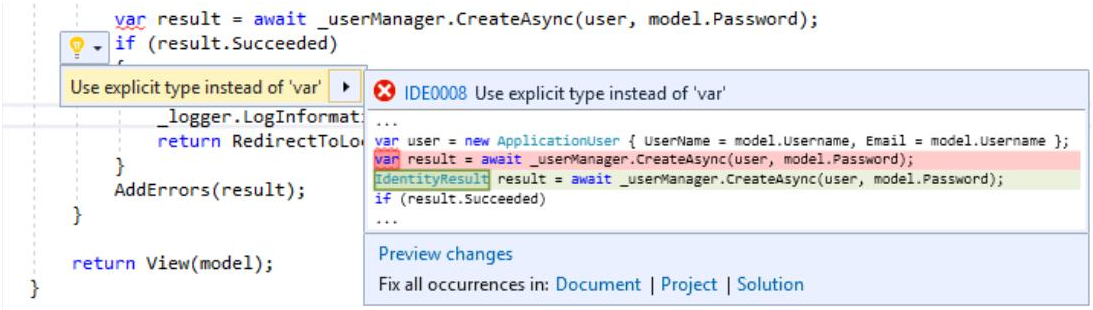

Figura 4. Ejemplo de iconos bombilla obtenido de la referencia [17]

3. En proyectos de gran tamaño, es posible que se tenga que reestructurar y refactorizar código de uno mismo o incluso de otra persona que escribiese anteriormente en el proyecto. Para facilitar este trabajo, Visual Studio, a través del menú de acciones rápidas del editor, admite potentes opciones de refactorización como extraer métodos o incluso cambiar nombres como se muestra en la figura 5. No obstante, esta característica depende del lenguaje utilizado, en este caso está disponible para C#, Visual Basic y C++ [17].

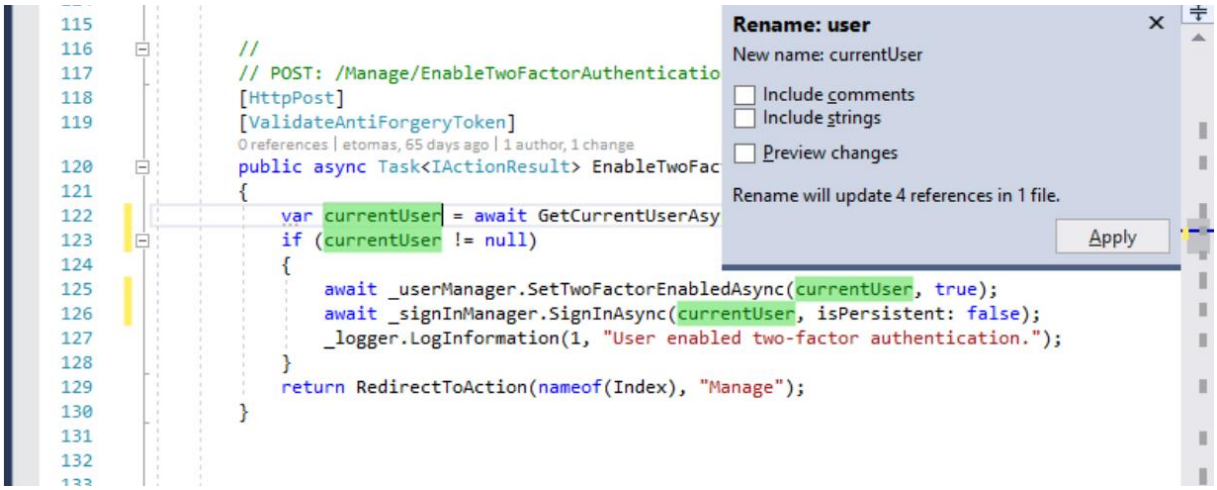

Figura 5. Ejemplo de renombrado obtenido de la referencia [17]

4. Existe además una ventana independiente que lista los errores de compilación y muestra problemas de análisis del código como en la figura 6. Esta permite que el programador pueda buscar contenido web que le ayude a resolver sus problemas simplemente presionando F1 encima del error[3] y filtrar entre todos los errores para localizar sus problemas y dirigirse exactamente a las líneas y actividades donde se encuentren los fallos [19].

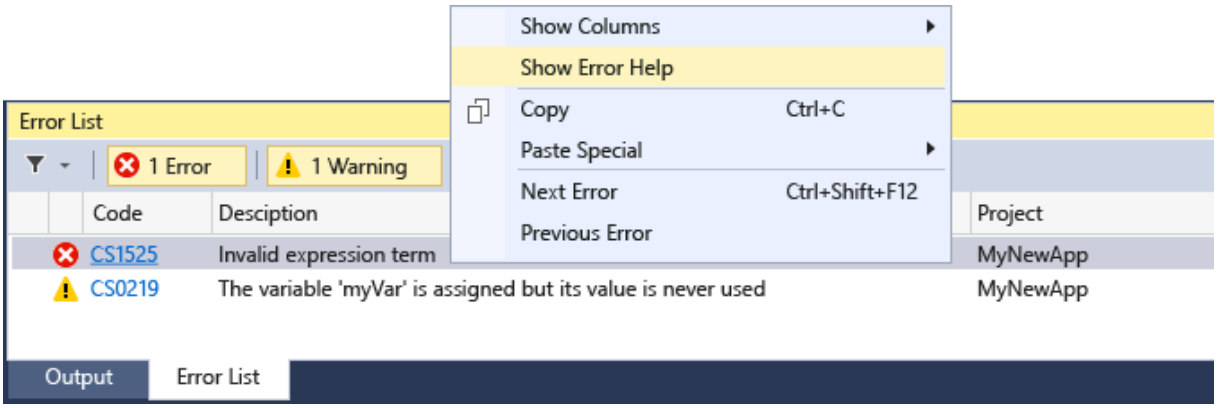

Figura 6. Ejemplo listado de errores obtenido de la referencia [19]

[3] Estos contenidos web se encuentran en la página https://msdn.microsoft.com/es-es/library/8x5x43k7.aspx

Para poder instalar el programa, hay que cumplir con los siguientes requisitos del sistema que se muestran en la tabla 1 [20].

Tabla 1. Requisitos del sistema para instalar Visual Studio

Requisitos del sistema	
Sistemas operativos	Windows 10 versión 1507 y posteriores: Home, Professional, Education y Enterprise
	Windows Server 2016: Standard y Datacenter
	Windows 8.1 (con actualización 2919355): Core, Professional y Enterprise
	Windows Server 2012 R2 (con actualización 2919355): Essentials, Standard y Datacenter
	Windows 7 SP1 (con las actualizaciones más recientes de Windows): Home, Premium, Professional, Enterprise y Ultimate
Hardware	Procesador de 1,8 GHz o superior. Doble núcleo o superior recomendado.
	2 GB de RAM (Memoria de Acceso Aleatorio); 4 GB de RAM recomendado (mínimo de 2,5 GB si se ejecuta en una máquina virtual)
	Espacio en disco duro: hasta 130 GB de espacio disponible, en función de las características instaladas; las instalaciones típicas requieren entre 20 y 50 GB de espacio libre.
	Velocidad del disco duro: para mejorar el rendimiento, instale Windows y Visual Studio en una SSD (Unidad de Estado Sólido).
	Tarjeta de vídeo que admita una resolución de pantalla mínima de 720p (1280 x 720); Visual Studio funcionará mejor con una resolución de WXGA[4] o superior.
Internet	Internet Explorer 11 o Microsoft Edge son necesarios para los escenarios relacionados con Internet. Algunas características podrían no funcionar a menos que se instalen estas soluciones o versiones posteriores.
Emulador	Para la compatibilidad con el emulador, se requieren las ediciones de Windows 8.1 Pro o Enterprise (x64). También es necesario un procesador que admita SLAT (Traducción de Direcciones de Segundo Nivel).
Xamarin	Xamarin.Android requiere una edición de 64 bits de Windows y JDK[5] de 64 bits.

3.2. Lenguaje C#

C# es un lenguaje de programación orientado a objetos diseñado para la infraestructura de lenguaje común. En un principio fue desarrollado y estandarizado por Microsoft como parte de su plataforma .NET[6], aunque después fue aprobado como un estándar por Ecma International (ECMA-334) y las organizaciones ISO (Organización Internacional para la Estandarización) e IEC (Comisión Electrotécnica Internacional) en la norma 23270.

[4] WXGA (Wide eXtended Graphics Array) es una norma de visualización de gráficos de ordenador de resolución 1366x768

[5] JDK (Java Development Kit) es un software que provee herramientas de desarrollo para la creación de programas en Java

[6] La plataforma .NET es un framework de Microsoft que permite un rápido desarrollo de aplicaciones debido a sus múltiples bibliotecas de desarrollo y su entorno de ejecución gestionado.

Su sintaxis básica deriva de C/C++ por lo que es fácilmente reconocida por cualquier persona familiarizada con dichos lenguajes. Utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras que no se encuentran en este lenguaje como tipos de valor que admiten valores nulos, enumeraciones, delegados y acceso directo a memoria.

Como lenguaje orientado a objetos, C# admite los conceptos de encapsulación, herencia y polimorfismo. Todas las variables y métodos, incluido el método *Main* que es el punto de entrada de la aplicación, se encapsulan dentro de definiciones de clase. Una clase puede heredar directamente de una clase primaria, pero puede implementar cualquier número de interfaces. [21]

El nombre "C Sharp" fue inspirado por la notación musical del signo '#', el cual indica que una nota debe realizarse un semitono más alto al original. En el caso de C#, la almohadilla simboliza una mejora del lenguaje C++, es decir, pasaría a convertirse en el lenguaje C++++ donde, si se colocan los signos '+' en una cuadricula de dos por dos, se puede generar el símbolo '#'. [22]

Aunque C# forma parte de la plataforma .NET (una API), C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma.

3.3. Xamarin y sus beneficios

Xamarin es un entorno de desarrollo de aplicaciones móviles que permite crear eficazmente diseños de interfaz de usuario nativas comunes para iOS, Android y Windows a partir de una base de código C#/.NET común. Las aplicaciones escritas con Xamarin pueden lograr un grado de reutilización de código entre plataformas de entre un 75% y casi un 100% lo que minimiza en gran cantidad los tiempos de desarrollo de aplicaciones multiplataforma. [23]

A pesar de la ventaja que supone la reutilización del código, es necesario conocer detalles de todas las plataformas en las que se quiera desarrollar la aplicación ya que las configuraciones no son iguales para todas y, para completar una correcta funcionalidad en todas las plataformas como se puede apreciar en la figura 7, habrá una parte exclusiva para cada una de las plataformas diseñadas, generándose así un archivo '.apk' para Android y un archivo '.ipa' para iOS. [25]

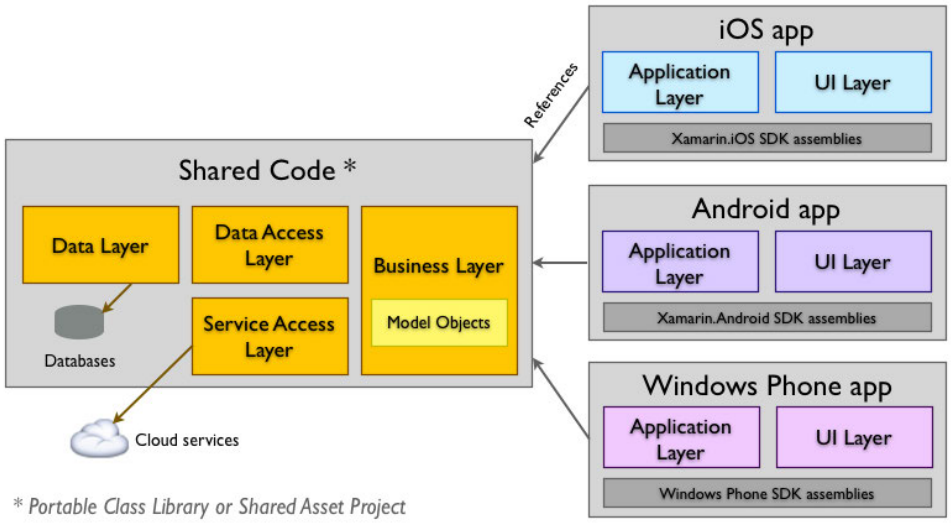

Figura 7. Arquitectura típica de una aplicación creada con Xamarin obtenido de la referencia [24]

Para crear una aplicación con Xamarin es necesario utilizar el programa Visual Studio. En este proyecto se utiliza Xamarin, aunque sea una aplicación exclusivamente Android, de cara a que, si en futuras versiones se quiere adaptar a otro tipo de plataforma, sea más sencillo que realizarla desde cero.

4. DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

En este apartado se explica todo el desarrollo de la aplicación diseñada. Primero, se detallan las especificaciones, restricciones y requisitos de usuario para el correcto uso de la aplicación, definiendo todas las características necesarias para su uso e incorporando diagramas que definen lo que el usuario se va a encontrar en la aplicación. También se incluyen diagramas UML y una tabla que indica qué clase utiliza cada librería descrita y una matriz de verificación que relaciona las clases desarrolladas con los requisitos de usuario definidos.

Después de definir las características en cuanto a la aplicación de forma genérica, se definen las clases programadas más importantes y los parámetros que han servido para el correcto funcionamiento de la APP. Lo más importante que se ha tenido en cuenta durante el desarrollo del proyecto ha sido la correcta funcionalidad de los juegos que se han desarrollado. Es por eso, que la metodología que se ha seguido es, primero, crear las páginas necesarias para el flujo de la aplicación, seguir con el desarrollo de las funcionalidades de cada juego individualmente, continuar con el diseño global de la aplicación y finalizar con la creación de la base de datos y la representación de resultados al finalizar cada uno de los juegos.

Todos los problemas y soluciones que se comentan en este apartado han sido detectados y probados por personas ajenas al proyecto compilándolos en la Tablet Alcatel Onetouch Pixi modelo 8079.

4.1. Especificaciones previas

Existen ciertas especificaciones previas a la realización del proyecto donde se indica el entorno en el que está desarrollada la APP. Estas son:

- El programa se ejecutará sobre un entorno Android.
- El diseño de la APP se realizará para un desarrollo en dispositivo Tablet
- El tamaño de los números y la duración de los mensajes que aparezcan por pantalla deben ser del tamaño y duración acordes a la edad de los alumnos y alumnas que realicen la actividad.
- Cada vez que un usuario realice los ejercicios, se guardarán sus resultados en una base de datos interna que el profesor podrá consultar y se mostrarán los datos en forma de gráfica al finalizar cada una de las sesiones.
- El diseño debe ser atractivo para el usuario de tal forma que perciba que no es un ejercicio matemático al uso, sino más bien una actividad entretenida y motivante.
- Si algún ejercicio dispone de niveles, estos se irán superando, aunque el usuario no acierte la respuesta.
- El usuario debe poder cerrar la aplicación en cualquier instante y grabar los resultados de los ejercicios realizados.

4.2. Restricciones del diseño

Existen ciertas restricciones y necesidades por parte del usuario para poder hacer uso de la aplicación.

En cuanto al dispositivo en el que se vaya a utilizar la APP:

- El espacio de memoria necesario es de 35 Mb
- La memoria RAM necesaria es de al menos 2 Gb
- Versión mínima del sistema operativo Android es: 5.0 Lollipop

- Es recomendable que la aplicación se ejecute en una Tablet de mínimo 11” para que los juegos de segmentos dispongan de un número razonable de valores posibles.

Como la aplicación de ejemplo no se encuentra en el Play Store, existen dos maneras de copiar la aplicación al dispositivo:

- Compilándola mediante Visual Studio. Una vez se ha ejecutado este proceso, la aplicación se mantendrá en el dispositivo hasta que el usuario la borre o hasta que se compile una nueva versión.
- Se podría crear el fichero ejecutable (apk) a través de las herramientas que proporciona Visual Studio, pero es un proceso largo que realmente no merece la pena, ya que el usuario que reciba la aplicación tendría que activarse las opciones de desarrollador.

4.3. Requisitos de usuario y diagrama de flujo

La aplicación “Mat3m4t1cas” que se desarrolla en este proyecto, consiste en la realización de tres ejercicios diferentes por parte de un alumno de primaria en el que poder incrementar y afianzar sus conocimientos numéricos y desarrollar funciones ejecutivas de distancia y dimensión. Los requisitos que el usuario debe realizar para el correcto funcionamiento de la aplicación son:

1. Escribir su nombre al iniciar la aplicación
2. Para elegir qué juego quiere practicar, primero elegir entre SEGMENTOS u OPERACIONES y, si elige SEGMENTOS, elegir entre el juego DIBUJA o ¿CUÁNTO MIDE?
3. En el juego OPERACIONES, rellenar los recuadros rojos vacíos con los números u operandos correspondientes. Los operandos posibles son ‘+’, ‘-’, ‘*’ o ‘/’ y los números posibles son entre ‘0’ y ‘100’ (no existe la posibilidad de introducir números negativos). Se pueden modificar los valores introducidos hasta antes de pulsar sobre el botón COMPROBAR.
4. En el juego ¿CUÁNTO MIDE?, rellenar en el recuadro de la derecha de la pantalla el valor en centímetros de lo que crea que mide el segmento.
5. En el juego DIBUJA, dibujar una línea en el lienzo disponible de longitud indicada por el enunciado.
6. El usuario puede volver a dibujar en el lienzo si la línea dibujada cree que no es correcta en el juego DIBUJA borrándose instantáneamente la línea anterior.
7. Para comprobar, es necesario que el usuario pulse sobre el botón COMPROBAR.
8. Al salir de un ejercicio, pulsando el botón SALIR o al finalizar la actividad, se muestran los resultados de la última sesión.
9. Pulsar OK una vez revisados los resultados mostrados al salir o terminar un juego para volver al menú anterior.

En la figura 8, se puede ver el diagrama de flujo correspondiente a la aplicación definido por los requisitos mencionados anteriormente. Como puede observarse, si el usuario decide realizar el juego de operaciones numéricas, este no se finaliza hasta que el alumno realice el último ejercicio del nivel 6, es decir, haya completado los 3 juegos de cada uno de los 6 niveles, o pulse en el botón SALIR. Sin embargo, en el caso de los juegos relacionados con segmentos, el alumno solo finaliza el ejercicio cuando pulse el botón SALIR, es decir, hará todos los ejercicios que desee.

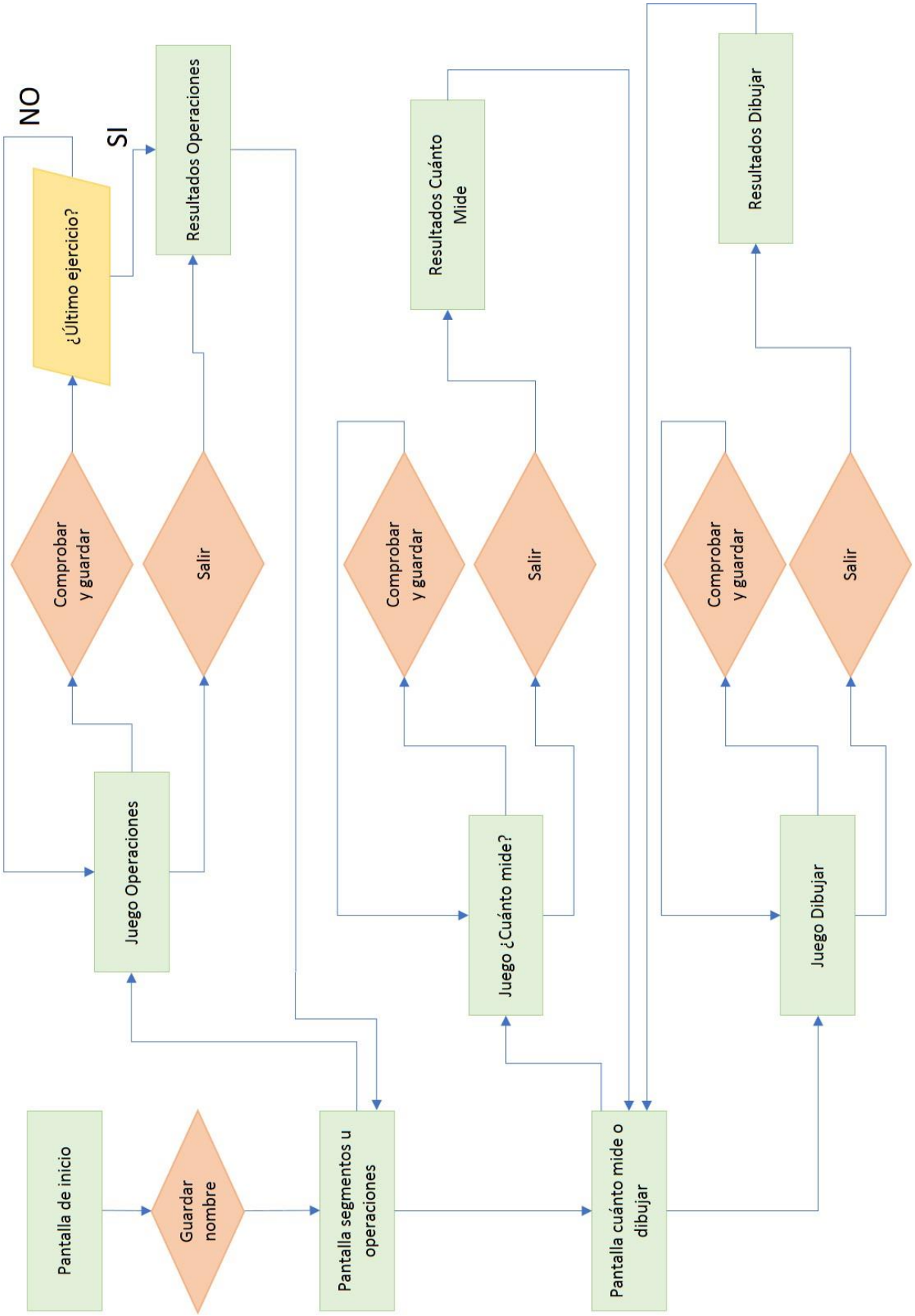

Figura 8. Diagrama de flujo de la aplicación

4.4. Diagramas UML

En este apartado se explica, a través de diagramas UML, el diseño estructural de la aplicación desde un punto de vista técnico. La aplicación está formada por clases desarrolladas por el programador las cuales cada una tiene asignadas pantallas de diseño específico (denominadas a partir de ahora como *layouts*) con todos los botones y cuadros necesarios para el correcto funcionamiento de la APP. Además, se utilizan librerías proporcionadas por otros programadores y Visual Studio que permiten el desarrollo de determinadas funciones. En la figura 9, se explica la distribución de la aplicación.

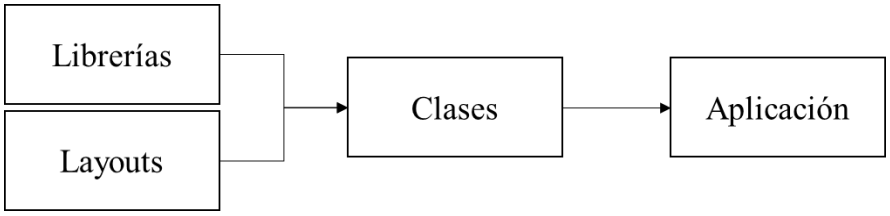

Figura 9. Diagrama genérico de la aplicación

Las librerías utilizadas, se han dividido según su función en la aplicación de la siguiente forma:

- Gráficas (*MPAndroidChart*): es una librería proporcionada por el desarrollador australiano Philipp Jahoda para la programación en Android de distintos tipos de gráficas. Permite realizar gráficas de resultados de múltiples formas, en esta aplicación se utilizan dos tipos:
  - Lineales: utilizada para mostrar los resultados de los ejercicios de segmentos.
  - De barras: utilizada para mostrar los resultados por niveles del juego de OPERACIONES.
- Base de datos (*SQLite.net*): permite la comunicación con una base de datos interna organizada en tablas que se encuentra en la memoria del dispositivo. En ella se almacenarán los resultados cada vez que se compruebe un ejercicio y, con los datos guardados, se podrá obtener la última sesión de un usuario y el listado de las variables a mostrar en las gráficas al salir o finalizar un juego.
- Sistema: son las librerías necesarias para iniciar pantallas y objetos.
- Diseño: son utilizadas para el diseño de la aplicación de tal forma que puedan solaparse unas pantallas encima de otras, reinicializarlas, tener un menú o desarrollar estilos.

En la figura 10 se encuentran por bloques las distintas librerías descritas anteriormente. Las librerías utilizadas para la base de datos, el sistema y el diseño son proporcionadas por Visual Studio para el desarrollo de aplicaciones.

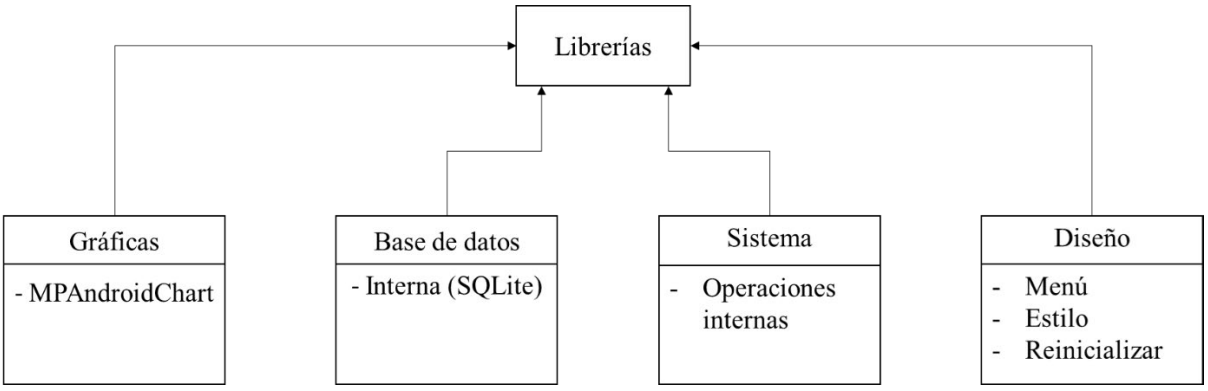

Figura 10. División por bloques de librerías utilizadas en la aplicación

Todas las clases de esta aplicación se han programado apoyándose en librerías proporcionadas por Visual Studio o por terceros, pero han sido todas creadas por el programador. Cada una de las pantallas tiene como mínimo un *layout* específico, por lo que en el diagrama UML de la figura 11, se relacionan las pantallas (en colores) con sus correspondientes *layouts* y las clases utilizadas (en gris).

- Pantallas
    - MainActivity: es la primera pantalla que se encuentra el usuario. En ella se almacena el nombre que introduzca el usuario como variable global y permite continuar a la pantalla "PrincipalActivity".
    - PrincipalActivity: en esta pantalla se muestran las dos opciones genéricas de juegos al usuario: SEGMENTOS u OPERACIONES y conduce a la pantalla elegida.
    - OperacionesActivity: en esta pantalla se muestran los *layouts* del 1 al 6 correspondientes a los 6 niveles de juegos de OPERACIONES.
    - SegmentosActivity: en esta pantalla se muestran las dos opciones específicas de juegos relacionados con segmentos: ¿CUÁNTO MIDE? o DIBUJA.
    - CuantoMideActivity: en esta pantalla se realiza el juego ¿CUÁNTO MIDE?
    - DibujaActivity: en esta pantalla se realiza la actividad DIBUJA
- Base de datos:
    - BaseDatosFunciones: clase que contiene métodos con los que comunicarse con la base de datos interna. Crea la BBDD (Base de Datos) en caso de no existir, guarda los resultados en la base de datos al finalizar un juego, obtiene el listado de todos los resultados para cada juego en una sesión y devuelve la última sesión para un usuario en concreto.
- Finger Paint:
    - FingerPaintCanvasView: clase que detecta cuando un usuario toca la pantalla y levanta el dedo de la misma de tal forma que se pueda dibujar en el lienzo del ejercicio DIBUJA.
    - FingerPaintPolyline: permite definir y obtener el color y el grosor de la línea y almacenar su trayectoria.
- Tablas de la base de datos:
    - ResultadosJD: esta clase contiene las columnas que conforman la tabla del juego DIBUJA en la base de datos. Está formada por: un número identificativo que autoincrementa para cada línea guardada, el nombre del usuario, la sesión actual, la hora y el día actual, el resultado correcto del ejercicio, el resultado que introduce el usuario y una variable que indica si el ejercicio es correcto o no.
    - ResultadosJM: esta clase contiene las columnas que conforman la tabla del juego ¿CUÁNTO MIDE? en la base de datos. Está formada por: un número identificativo que autoincrementa para cada línea guardada, el nombre del usuario, la sesión actual, la hora y el día actual, el resultado correcto del ejercicio, el resultado que introduce el usuario y una variable que indica si el ejercicio es correcto o no.

  - ResultadosJO: esta clase contiene las columnas que conforman la tabla del juego OPERACIONES en la base de datos. Está formada por: un número identificativo que autoincrementa para cada línea guardada; el nombre del usuario; la sesión actual; la hora y el día actual; el nivel del ejercicio; el operador, operando 1, operando 2 y resultado que son mostrados en el ejercicio; el operador, operando 1, operando 2 y resultado que introduce el usuario, y una variable que indica si el ejercicio es correcto o no. En caso de que para ese nivel alguna de las variables sea vacía, se almacenará en el caso de números un "-1" y en el caso de un operando la letra "a".

- Variables globales:
  - Variables: En esta clase se almacenan las variables globales de la aplicación: el nombre del usuario introducido en la pantalla "MainActivity", el número de ejercicios por nivel que debe realizar el usuario (en este caso es 3) y una variable que se utiliza para identificar si un usuario ha terminado un ejercicio o no. Estas variables se explican con más detenimiento en el apartado 4.10.

- Layouts
  - CuantoMideLayout: diseño de la pantalla de juego ¿CUÁNTO MIDE? Incluye un lienzo donde aparece la línea, un cuadro de texto numérico donde el usuario poder introducir números, un título y dos botones (COMPROBAR y SALIR)
  - Custom_Toast: diseño en forma de nube para los mensajes por pantalla.
  - DibujaLayout: diseño de la pantalla de juego DIBUJA. Incluye un lienzo donde poder dibujar la línea, un título y dos botones (COMPROBAR y SALIR)
  - Main: diseño de la pantalla principal. Incluye un cuadro de texto donde poder escribir el usuario su nombre y un botón (JUGAR) para continuar.
  - MostrarResultadosJD: diseño donde se muestran los resultados del juego DIBUJA al presionar el botón SALIR en la pantalla "DibujaLayout". Incluye un título, un cuadro donde poner los resultados, un texto donde poder introducir los aciertos totales y un texto donde poder indicar la efectividad.
  - MostrarResultadosJM: diseño donde se muestran los resultados del juego ¿CUÁNTO MIDE? al presionar el botón SALIR en la pantalla "CuantoMideLayout". Incluye un título, un cuadro para poner los resultados, un texto donde poder introducir los aciertos totales y un texto donde poder indicar la efectividad.
  - MostrarResultadosJO: diseño donde se muestran los resultados del juego OPERACIONES al presionar el botón SALIR en las pantallas "OperacionesLayout_L1", "OperacionesLayout_L2", "OperacionesLayout_L3", "OperacionesLayout_L4", "OperacionesLayout_L5" o "OperacionesLayout_L6", o al finalizar el último ejercicio del nivel 6. Incluye un título, un cuadro para poner los resultados, un texto donde poder introducir los aciertos totales y un texto donde poder indicar la efectividad.
  - OperacionesLayout_L1: diseño de la pantalla de nivel 1 del juego OPERACIONES. Incluye cuadros donde mostrar los operandos 1 y 2 y el operador, un cuadro de texto donde poder introducir el resultado, un título y dos botones (COMPROBAR y SALIR)

- OperacionesLayout_L2: diseño de la pantalla de nivel 2 del juego OPERACIONES. Incluye cuadros donde mostrar el operando 2, el resultado y el operador, un cuadro de texto donde poder introducir el operando 1, un título y dos botones (COMPROBAR y SALIR)
- OperacionesLayout_L3: diseño de la pantalla de nivel 3 del juego OPERACIONES. Incluye cuadros donde mostrar el operando 1 y el operador, dos cuadros de texto donde poder introducir el operando 2 y el resultado, un título y dos botones (COMPROBAR y SALIR)
- OperacionesLayout_L4: diseño de la pantalla de nivel 4 del juego OPERACIONES. Incluye cuadros donde mostrar los operandos 1 y 2 y el resultado, un cuadro de texto donde poder introducir el operador, un título y dos botones (COMPROBAR y SALIR)
- OperacionesLayout_L5: diseño de la pantalla de nivel 5 del juego OPERACIONES. Incluye cuadros donde mostrar el resultado y el operador, dos cuadros de texto donde poder introducir el operador 1 y el operador 2, un título y dos botones (COMPROBAR y SALIR)
- OperacionesLayout_L6: diseño de la pantalla de nivel 6 del juego OPERACIONES. Incluye cuadros donde mostrar el operando 2 y el resultado, dos cuadros de texto donde poder introducir el operando 1 y el operador, un título y dos botones (COMPROBAR y SALIR)
- PrincipalLayout: diseño de la pantalla “PrincipalActivity”. Incluye un título y dos botones (SEGMENTOS u OPERACIONES)
- SegmentosLayout: diseño de la pantalla “SegmentosActivity”. Incluye un título y dos botones (¿CUÁNTO MIDE? y DIBUJA)

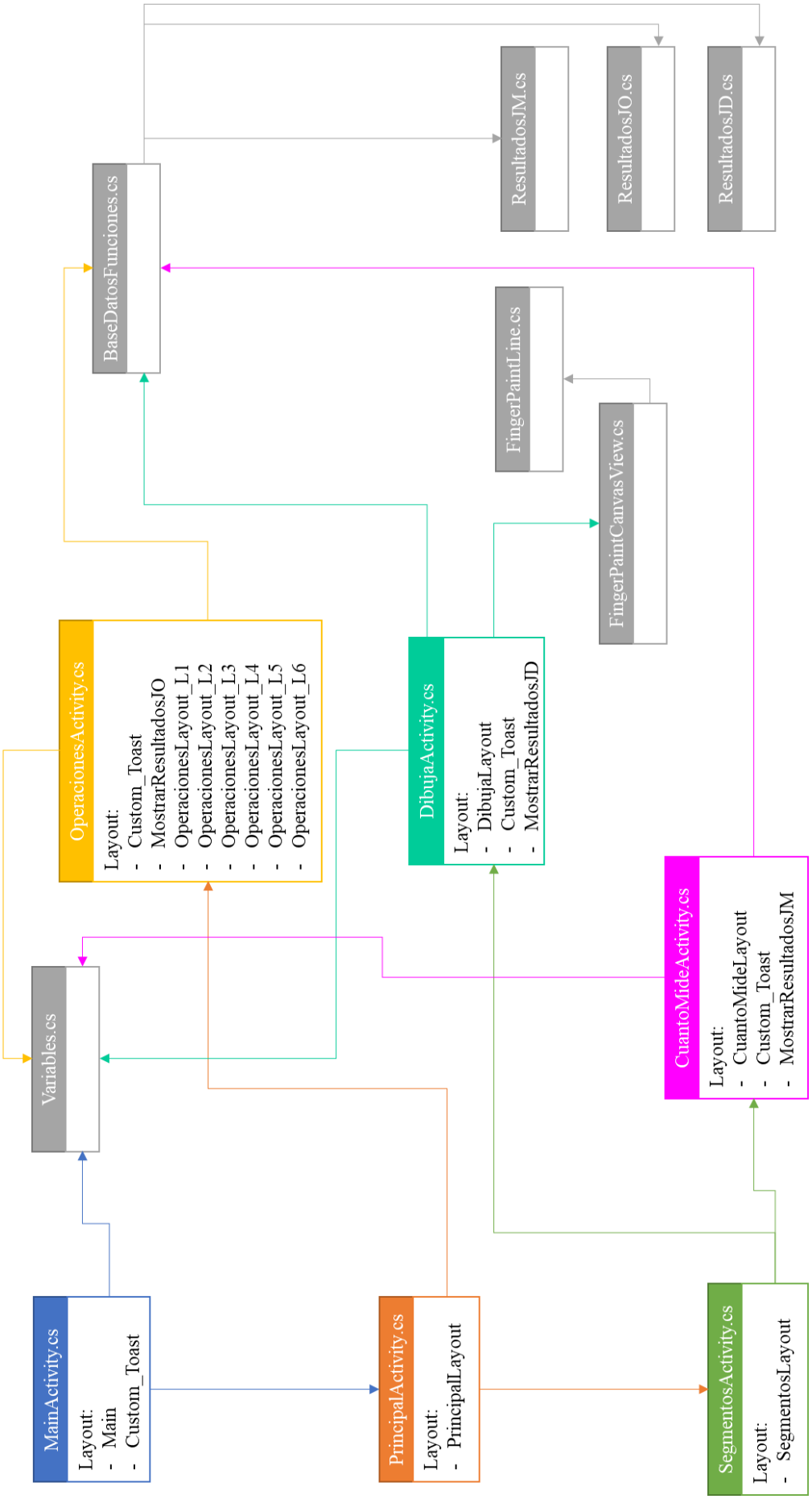

Figura 11. Diagrama UML de la aplicación

La relación entre las clases definidas y las librerías utilizadas, se describen en la tabla 2.

Tabla 2. Relación entre librerías y clases.

Librerías / Clases	BaseDatosFunciones	CuantoMideActivity	DibujaActivity	FingerPaintCanvasView	FingerPaintPolyline	MainActivity	OperacionesActivity	PrincipalActivity	ResultadosJD	ResultadosJM	ResultadosJO	SegmentosActivity	Variables
MPAndroidChart													
MikePhil.Charting.Charts		x	x				x						
MikePhil.Charting.Data	x	x	x				x						
MikePhil.Charting.Interfaces.Datasets	x												
MikePhil.Charting.Components		x	x				x						
MikePhil.Charting.Formatter							x						
MikePhil.Charting.Components.XAxis		x	x				x						
Base de datos													
SQLite	x												
SQLite.Net.Attributes									x	x	x		
Sistema													
System	x	x	x	x			x		x	x	x		
System.Linq	x												
System.Collections.Generic	x						x						
Diseño													
Android.App		x	x			x	x	x				x	
Android.Graphics	x	x	x	x	x	x	x	x				x	
Android.Content				x									
Android.Content.PM		x	x			x	x	x				x	
Android.OS		x	x			x	x	x				x	
Android.Widget		x	x			x	x	x				x	
Android.Views		x	x	x		x	x						
Android.Util				x									

4.5.Matriz de verificación

En este apartado se van a relacionar los requisitos de usuario y las clases utilizadas en la aplicación, formando así la matriz de verificación que se puede ver en la tabla 3: todos los requisitos de usuario han sido implementados por al menos una clase.

Tabla 3. Matriz de verificación

Clases / Requisitos de usuario	BaseDatosFunciones	CuantoMideActivity	DibujaActivity	FingerPaintCanvasView	FingerPaintPolyline	MainActivity	OperacionesActivity	PrincipalActivity	ResultadosJD	ResultadosJM	ResultadosJO	SegmentosActivity	Variables
1						x							x
2								x				x	
3							x						
4		x											
5			x	x	x								
6			x	x	x								
7		x	x				x						
8	x	x	x				x		x	x	x		x
9		x	x				x						

4.6. Parámetros de diseño global

Como se ha definido anteriormente, los *layouts* son archivos XML donde se especifican los diseños de las páginas que el usuario va a encontrarse. En el apartado 4.3 de diagramas UML se pueden encontrar definidos todos los *layouts* diseñados, pero cabe destacar ciertos parámetros que se han utilizado de manera global para obtener el diseño deseado.

4.6.1. Parámetro weightSum

Todas las páginas se han desarrollado en porcentajes de tal forma que se ajusten a distintos dispositivos. Esto se consigue mediante los parámetros *weightSum* y *layout_weight*. Asignando un valor de 100 al parámetro *weightSum*, indicamos que todo lo que componga ese *layout* debe sumar 100 y, lo que ocupa cada uno de los *layout* inferiores, se asigna con el parámetro *layout_weight*. En el ejemplo de la figura 12, se observa cómo se le asigna a un *layout* el valor 100 al parámetro *weightSum* y a los *layout* por debajo valores de 20 y 80 en los parámetros *layout_weight*, sumando así un total de 100.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:weightSum="100"
    android:background="#ff008000">
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="20"
        android:id="@+id/descripcion"
        android:gravity="center_vertical">
    </LinearLayout>
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="80"
        android:id="@+id/linearLayout2"
        android:weightSum="100">
        <LinearLayout
            android:orientation="horizontal"
            android:layout_width="0dp"
            android:layout_height="fill_parent"
            android:layout_weight="80"
            android:background="#ffe65100"
            android:id="@+id/linearLayout1">
        <LinearLayout
            android:orientation="vertical"
            android:layout_width="0dp"
            android:layout_height="fill_parent"
            android:layout_weight="20"
            android:weightSum="100"
            android:id="@+id/linearLayout4"
            android:gravity="center">
```

Figura 12. Ejemplo parámetros *layout_weight* y *weightSum*

El parámetro *weightSum* solo afecta a los *layout* que haya por debajo como primer nivel, es decir, si el *layout* 1 se divide en los *layouts* 2 y 3 y el *layout* 2 se divide en los *layout* 4 y 5, el parámetro *weightSum* del *layout* 1 solo afecta al parámetro *layout_weight* de los *layout* 2 y 3. Para poder ajustar en porcentajes los *layout* 4 y 5 es necesario asignarle un parámetro *weightSum* al *layout* 2.

4.6.2. Botones

Los botones que se encuentran en la aplicación se han diseñado en un programa externo y son imágenes PNG. Para poder asignar como fondo una imagen a un botón, es necesario que la imagen esté guardada dentro del apartado de *Resources* de la aplicación, en este caso se ha decidido crear una carpeta destinada a todas las imágenes llamada *drawable*. Para asignar una de las imágenes, hay que utilizar el parámetro *src*, tal y como se muestra en la figura 13.

```
<ImageButton
    android:src="@drawable/COMPROBAR"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@+id/comprobar"
    android:scaleType="fitCenter"
    android:background="#ff008000" />
```

Figura 13. Ejemplo asignar una imagen de fondo a un botón

4.6.3. Ocultar teclado virtual

De manera automática, aparecen los teclados una vez se inicia una nueva pantalla. Para evitar que aparezca, es necesario introducir el siguiente comando dentro de las clases *onCreate* de cada una de las *activities*:

Window.SetSoftInputMode(SoftInput.StateAlwaysHidden);

4.6.4. Orientación horizontal

Para un mejor uso de la aplicación, se ha decidido fijar la orientación en posición horizontal en todas las pantallas. Para ello es necesario introducir al principio de cada clase *ScreenOrientation = ScreenOrientation.Landscape* tal y como se muestra en la Figura 14.

```
namespace App1
{
    [Activity(Label = "DibujaActivity", ScreenOrientation = ScreenOrientation.Landscape)]
    public class DibujaActivity : Activity
    {
}
```

Figura 14. Ejemplo para fijar la orientación horizontal

4.6.5. Gama de colores

En cuanto a los colores utilizados, se ha optado por elegir un tono verde oscuro y letras blancas que simulen una pizarra de clase para que resulte familiar a los alumnos. Se ha decidido por el sistema hexadecimal para la definición de los colores. A continuación, se definen los colores principales utilizados en la aplicación:

- Color de fondo principal: #ff008000
- Color de letra enunciados: #ffffff
- Color de fondo color claro: #ff43a047
- Márgenes y números: #ffe65100

4.6.6. Tipografía

En esta aplicación, se ha decidido utilizar una tipografía diferente a las que vienen por defecto en el programa, en concreto, la fuente “Berlin Sans FB Demi Negrita”, a partir de ahora se denominará fuente BSDB.

Para poder introducir una tipografía específica a una aplicación, es necesario descargarse el archivo .TTF[7] de la tipografía en cuestión[8] y dejarlo en la carpeta *Assets* del proyecto. Además, para poder asignar una tipografía en concreto a un texto, es necesario el uso del parámetro *Typeface* encontrado en la librería *Android.Graphics*. Con este parámetro se puede generar un objeto tipo *typeface* de la siguiente forma:

Typeface tf = Typeface.CreateFromAsset(Assets, "FuenteBSDB.TTF");

[7] La extensión de archivo TTF (True Type Font) fue creada originalmente por Apple para permitir a sus desarrolladores controlar la apariencia de sus textos tanto en páginas impresas como en pantallas de ordenador.
[8] Desde cualquier navegador se pueden encontrar múltiples tipografías que pueden descargarse de manera gratuita si se dispone de conexión a internet.

Y, posteriormente, se le puede asignar la fuente a cada uno de los textos de la siguiente forma, pudiendo elegir entre los estilos normal, negrita (*Bold*), cursiva (*Italic*) o ambas (*Bold_Italic*):

titulo.SetTypeface(tf, TypefaceStyle.Normal);

4.7. Juego DIBUJA

El juego DIBUJA consiste en que el alumno dibuje con sus dedos tocando en la pantalla una línea de una cierta longitud. En la siguiente figura 15, se muestra un ejemplo de lo que el alumno se va a encontrar al iniciar la actividad.

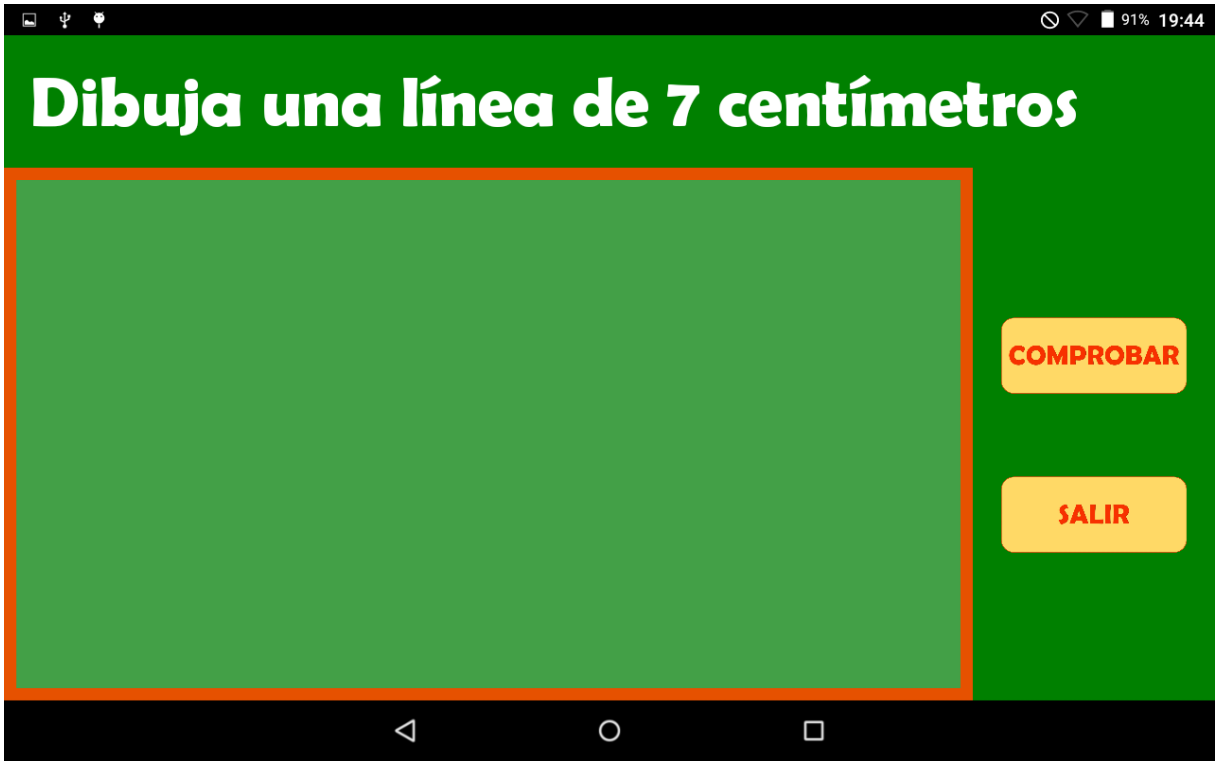

Figura 15. Pantalla inicial juego DIBUJA

En los siguientes apartados se explica en detalle las acciones principales que se han desarrollado a la hora de implementar esta actividad, así como las funciones necesarias para el correcto funcionamiento de la misma.

4.7.1. Descripción

La pantalla se divide en tres partes: el enunciado superior donde se especifica la longitud de la línea a dibujar, el lienzo donde poder dibujar y realizar el ejercicio y la zona derecha de botones donde poder comprobar el resultado o salir del juego.

Para el correcto desarrollo de este juego, ha sido necesario crear las clases *FingerPaintCanvasView* y *FingerPaintPolyline*. Para su diseño, se han utilizado de ayuda juegos de dibujar gratuitos descargados de la página de soporte de Visual Studio que han permitido el correcto desarrollo del panel de dibujo, aunque se ha prescindido de numerosos detalles que no eran necesarios en este juego tales como la posibilidad del cambio de color de línea o tamaño las cuales se pueden modificar en el código de la Figura 16 desde la actividad *FingerPaintCanvasView*.

```
// External interface accessed from MainActivity
public Color StrokeColor { set; get; } = Color.White;

public float StrokeWidth { set; get; } = 5;
```

Figura 16. Ejemplo para modificar parámetros de línea con *canvas*

La clase *FingerPaintCanvasView* es un objeto tipo *View* al que se le ha asignado como parte del *layout* el trozo de color verde claro que se puede ver en la figura 15. Al haberse asignado ese trozo exclusivo, el alumno solo dibujará si pulsa sobre ese recuadro, si no, no activará la clase. En la figura 17 se muestra cómo asignar un objeto tipo *view* a un *layout* en concreto y cómo se le pueden asignar valores como si se tratase de otro *layout* más tales como el margen (*layout_margin*) o el color de fondo (*background*).

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="0dp"
    android:layout_height="fill_parent"
    android:layout_weight="80"
    android:background="#ffe65100"
    android:id="@+id/linearLayout1">
    <App1.FingerPaintCanvasView
        android:background="#ff43a047"
        android:layout_margin="10dp"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:id="@+id/pantalladibujar" />
```

Figura 17. Ejemplo de objeto tipo *view* en un *layout*

La clase *FingerPaintPolyline* se utiliza para obtener y asignar los datos de la línea que se va a dibujar, tales como el color (*Color*), el espesor (*StrokeWidth*) o la situación de la línea (*Path*).

Cada vez que el alumno dibuja en el lienzo, la línea anterior se elimina de tal forma que puede intentar dibujar la línea tantas veces como quiera antes de comprobarla. Además, para la obtención de la distancia dibujada, el juego calcula la longitud en línea recta desde el primer punto donde se pulsa hasta el último donde el alumno levanta el dedo, por lo que no afecta si el alumno no dibuja exactamente una línea recta.

Una vez el alumno esté convencido de que la línea está dibujada correctamente, debe pulsar el botón COMPROBAR, saldrá un mensaje con el resultado y se iniciará un nuevo ejercicio hasta que decida salir. Si decide salir y no realizar más ejercicios, puede pulsar el botón SALIR sin que cuente esa última actividad.

Al salir del juego se mostrará por pantalla una gráfica lineal con los resultados. Para más información puede consultar el apartado 4.12.1.

4.7.2. FingerPaint

Anteriormente, se ha comentado que hay ciertos parámetros que pueden modificarse para modificar la estética de la línea que se va a dibujar en este juego, pero también ha sido necesario programar ciertas clases para que el usuario pudiera “dibujar” en la pantalla del dispositivo.

4.7.2.1. OnTouchEvent

La clase *OnTouchEvent* es un evento que se activa cuando el usuario pulsa sobre el lienzo de la pantalla y, dependiendo de lo que haga el usuario, el código de atención al evento realiza unas acciones u otras.

Para conseguirlo, se ha utilizado la clase *MotionEvents*, la cual puede detectar cinco tipos distintos de movimientos de los cuales dos, ACTION_DOWN y ACTION_UP, son para el primer punto que se detecta en la pantalla nada más pulsar sobre ella y el último punto que se detecta en la pantalla al levantar el dedo; los tipos ACTION_POINTER_DOWN y ACTION_POINTER_UP, se utilizan para cuando el usuario presiona la pantalla con más de un dedo, es decir, son el resto de puntos muestras se mantenga el primer dedo en la pantalla, de ahí que se defina únicamente para las pantallas multi-touch[9]. El evento restante, ACTION_MOVE, son todos los puntos por los que se desplaza el dedo una vez pulsado. Estos movimientos se han programado con una estructura de control tipo *switch* permitiendo definir distintas acciones para cada tipo de movimiento.

ACTION_POINTER_DOWN: Detecta todos los puntos que tocan la pantalla por primera vez, es decir, cada vez que el usuario toca el lienzo en nuestro caso. En esta aplicación, ha sido necesario inicializar la clase cada vez que sucedía esta acción ya que si el usuario se equivoca puede volver a dibujar y, de esta manera, se "limpia" el lienzo y se vuelve a empezar el ejercicio. Además, cada vez que se detecta que el usuario toca la pantalla, se asigna el color y grosor a la línea que debe dibujarse y se almacenan los puntos X e Y de la pantalla donde ha pulsado el usuario en el parámetro *Path* de la clase *FingerPaintPolyline* para pintarlos con ayuda de la clase *OnDraw* y calcular la longitud de la línea dibujada por el usuario en la clase *GetLineMeasure*.

ACTION_MOVE: Detecta cualquier cambio mientras se ha presionado la pantalla, en nuestro caso, el usuario debe estar moviendo el dedo presionado para pintar sobre el lienzo. Para poder pintar toda la línea con la clase *OnDraw*, cada vez que se detecta un movimiento, se almacenan los puntos X e Y en el parámetro *Path* de la clase *FingerPaintPolyline* y estos se colorean.

ACTION_POINTER_UP: Detecta el punto donde el usuario ha dejado de presionar la pantalla. Este punto indica el fin de la línea dibujada por lo que se almacena como último punto en el parámetro *Path* de la clase *FingerPaintPolyline* y además se almacenan para el cálculo total de la línea en la clase *GetLineMeasure*.

4.7.2.2. OnDraw

La clase *OnDraw* actúa como un evento que dibuja la línea que se define en los parámetros de la clase *FingerPaintPolyline* cada vez que se actualizan los datos y, para ello, se utilizan las clases *Canvas* y *Paint* de la librería *Android.Graphics*. *Canvas* define qué se va a dibujar, en nuestro caso un *Path*, pero existen muchas posibilidades más. Paint define cómo se va a dibujar, en nuestro caso el color y el grosor de la línea vienen definidos, pero pueden definirse más parámetros como la forma de la línea que se va a dibujar (*paint.StrokeJoin*) en nuestro caso redonda (*Paint.Join.Round*) o la forma del final de la línea dibujada (*paint.StrokeCap*) en nuestro caso un semicírculo cuyo centro es el final de la línea (*Paint.Cap.Round*)

Los parámetros *Color* y *StrokeWidth* definidos en la clase *FingerPaintPolyline* definen el color y el grosor de la línea que va a ser dibujada por lo que se almacenan como datos para la clase Paint. El parámetro *Path* de la clase *FingerPaintPolyline* se pasa como parámetro a la hora de dibujar con la clase *Canvas*. En la figura 18 se puede ver cómo se asignan estos parámetros.

[9] Las pantallas multitáctiles, o comúnmente multi-touch, son pantallas que disponen de una tecnología que permite detectar simultáneamente múltiples puntos de contacto e interpretarlos según las acciones que se realicen, como hacer zoom en imágenes o mapas separando y juntando dos dedos sobre la imagen.

```
paint.Color = fingerPaintPolyline.Color;
paint.StrokeWidth = fingerPaintPolyline.StrokeWidth;
canvas.DrawPath(fingerPaintPolyline.Path, paint);
```

Figura 18. Asignación de los parámetros definidos en la clase *FingerPaintPolyline*

4.7.2.3. GetLineMeasure

La clase *GetLineMeasure* calcula la longitud en centímetros de la línea dibujada por el usuario. Como se describió en el apartado 4.7.2.1. los puntos iniciales y finales de las líneas dibujadas son almacenadas, en esta aplicación se han guardado como las variables x1, y1, x2 e y2. Estos valores están definidos en píxeles por lo que para realizar el cálculo en centímetros deben dividirse entre el número exacto de píxeles por pulgada de la pantalla en la dimensión X o Y (según corresponda) y multiplicarlos por la densidad de la pantalla y por 2,54 para poder pasar el valor de pulgadas a centímetros como se muestra en la figura 19. Los datos píxeles reales y densidad real del dispositivo se obtienen de la clase *DisplayMetrics* de la librería *Android.Util*.

```
//Convertimos los valores x e y en cm:
double x1In = x1 / (Resources.DisplayMetrics.Xdpi);
double x2In = x2 / (Resources.DisplayMetrics.Xdpi);
double y1In = y1 / (Resources.DisplayMetrics.Ydpi);
double y2In = y2 / (Resources.DisplayMetrics.Ydpi);

double x1Cm = x1In * 2.54f * (Resources.DisplayMetrics.Density);
double x2Cm = x2In * 2.54f * (Resources.DisplayMetrics.Density);
double y1Cm = y1In * 2.54f * (Resources.DisplayMetrics.Density);
double y2Cm = y2In * 2.54f * (Resources.DisplayMetrics.Density);
```

Figura 19. Conversión de píxeles a centímetros reales

Una vez obtenidos los puntos en la pantalla en centímetros, con ayuda del teorema de Pitágoras, podemos calcular la distancia entre esos dos puntos. Según el teorema, el cuadrado de la hipotenusa de un triángulo rectángulo es igual a la suma de los catetos al cuadrado. En nuestro caso, los catetos serían las diferencias entre los puntos del eje X y del eje Y, es decir, su valor sería el que se muestra en las ecuaciones 1 y 2.

$$cateto_1 = |x_1 - x_2| \quad (1)$$

$$cateto_2 = |y_1 - y_2| \quad (2)$$

Y la hipotenusa sería la distancia que se quiere calcular, siendo la distancia lo que se muestra en la ecuación 3.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (3)$$

En la figura 20 se muestra cómo se realiza el cálculo de las ecuaciones 1, 2 y 3 en el lenguaje de programación gracias al método *Math* programado en JavaScript, resaltando que para calcular las potencias se utiliza el método *Math.Pow* y para calcular las raíces cuadradas se utiliza *Math.Sqrt*, y devuelve el valor de la distancia en centímetros y redondeado al entero más próximo.

```
//Calculamos la distancia
cateto1 = (x1Cm - x2Cm);
cateto2 = (y1Cm - y2Cm);

solucionMatematica = Math.Sqrt(Math.Pow(cateto1, 2) + Math.Pow(cateto2, 2));
redondeoSolucionMatematica = Math.Round(solucionMatematica);
distance = Convert.ToInt32(redondeoSolucionMatematica);

return distance;
```

Figura 20. Cálculo de distancia entre dos puntos

4.7.3. MostrarEnunciado

La clase *MostrarEnunciado* modifica el enunciado de tal forma que el alumno que realice la actividad deba dibujar una distancia aleatoria que quepa en el hueco definido para el lienzo. Ya que la aplicación se ha programado para que solo pueda utilizarse con la Tablet posicionada de forma horizontal, la longitud máxima va a estar definida por el ancho del lienzo, por lo que calculando esta longitud en centímetros (de la misma forma que en el apartado 4.7.2.3) definimos los valores aleatorios posibles siendo el valor mínimo el 1. Para poder asignar valores aleatorios entre un rango determinado se utiliza la clase *Random* () de la librería *Java.Util* y, una vez obtenido el valor, este puede asignarse como parámetro al texto definido como enunciado tal y como se muestra en la figura 21.

```
Random r = new Random();
distanceAsked = r.Next(1, maxCm);

String enunciado = ("Dibuja una línea de " + distanceAsked + " centímetros");
texto.SetText(enunciado, TextView.BufferType.Normal);
```

Figura 21. Obtención del enunciado para el juego DIBUJA

4.7.4. Comprobar resultado

Al pulsar el botón COMPROBAR, la aplicación compara la distancia real dibujada por el usuario (definida en el apartado 4.7.2.3) con la distancia mostrada por el enunciado (definida en el apartado 4.7.3). Dependiendo de si el resultado es correcto o no, aparecen, como se ha mencionado anteriormente, distintos mensajes por pantalla. En caso de acierto aparece el mensaje "¡¡Muy bien!!" como aparece en la figura 22 y, en caso de fallo, aparece el mensaje "Has dibujado una línea de X centímetros" con la distancia dibujada por el alumno para que pueda ayudarle con su aprendizaje como aparece en la figura 23.

Figura 22. Mensaje de acierto en el juego DIBUJA

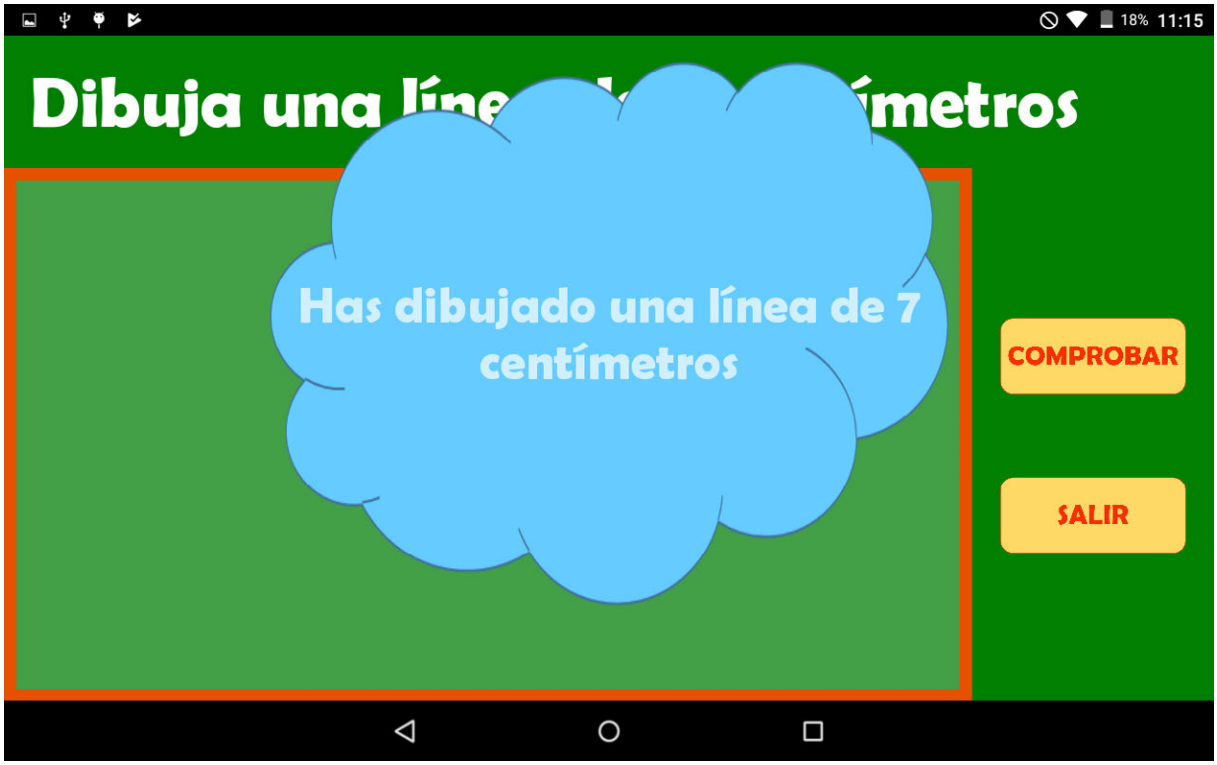

Figura 23. Mensaje de fallo en el juego DIBUJA

4.8. Juego ¿CUÁNTO MIDE?

El juego ¿CUÁNTO MIDE? consiste en que el alumno averigüe la longitud en centímetros de un segmento que aparece en la pantalla. En la siguiente figura 24, se muestra un ejemplo de lo que el alumno se va a encontrar al iniciar la actividad.

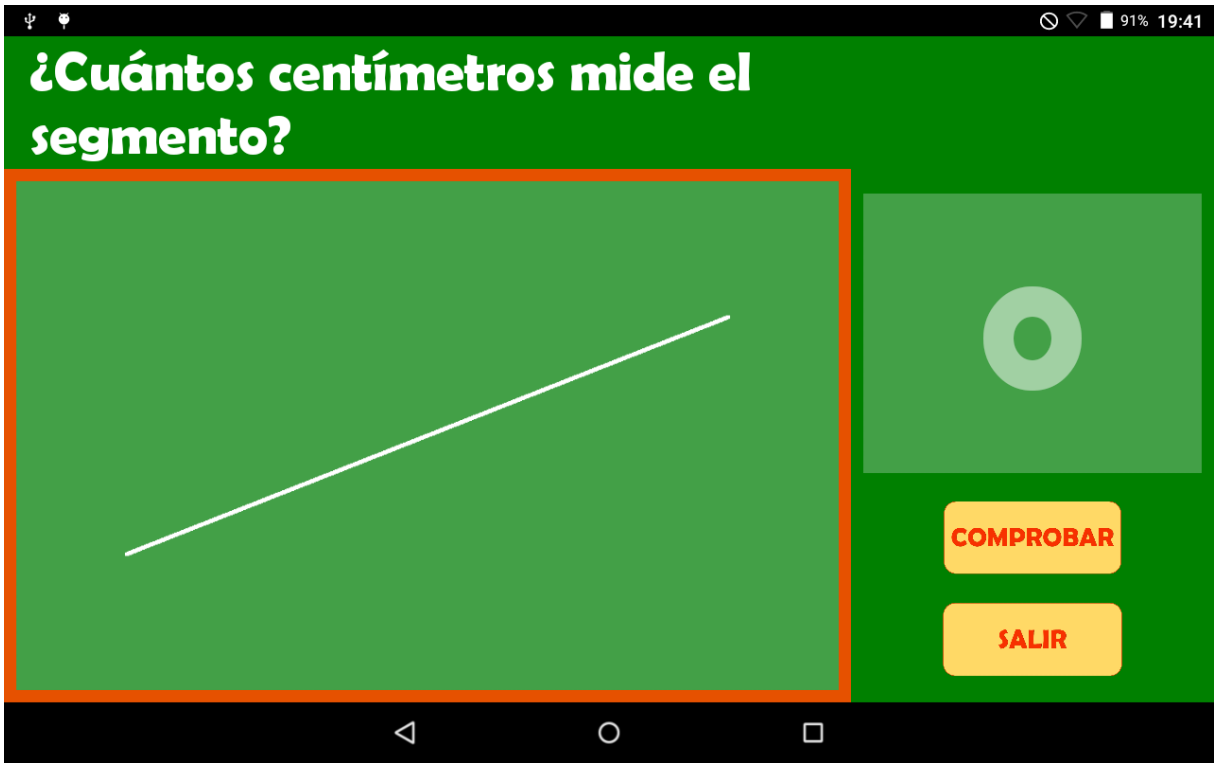

Figura 24. Pantalla inicial juego ¿CUÁNTO MIDE?

En los siguientes apartados se explica en detalle las acciones principales que se han desarrollado a la hora de implementar esta actividad, así como las funciones necesarias para el correcto funcionamiento de la misma.

4.8.1. Descripción

La pantalla, como puede observarse en la figura 24, se divide en cuatro partes: el enunciado superior donde se pregunta cuánto mide el segmento, un recuadro donde aparece la línea cuya longitud hay que adivinar, un recuadro en la zona derecha donde introducir el resultado en números naturales y la zona inferior derecha de botones donde poder comprobar el resultado o salir del juego.

La línea puede estar orientada en varias direcciones, pero siempre aparece en el centro del recuadro y con una longitud no superior al recuadro que la contiene. El alumno a simple vista debe calcular la longitud en centímetros e introducir el valor en el recuadro de la derecha.

En este caso, para mostrar el segmento del juego, se ha decidido que aparezca en forma de imagen, es decir, en verdad no se dibujan líneas sobre un fondo verde continuamente, sino que cada vez que se inicia el ejercicio se genera una nueva imagen con una línea dibujada distinta, esta se almacena como una imagen en la misma ruta donde se encontraba la anterior (sustituyéndola) y esta se asigna como fondo al *layout* en cuestión. En la figura 25 se puede observar el código a introducir para poder asignar una imagen tipo *view* a un *layout*, donde el parámetro más importante es el *src*, la ruta de donde se va a obtener la imagen a mostrar.

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="0dp"
    android:layout_height="fill_parent"
    android:layout_weight="70"
    android:id="@+id/zonaDibujo"
    android:background="#ffe65100">
    <ImageView
        android:src="@android:drawable/ic_menu_gallery"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/imageZonaDibujo"
        android:background="#ff43a047"
        android:layout_margin="10dp" />
```

Figura 25. Código fuente para asignar una imagen tipo *view* a un *layout*

Una vez el alumno esté convencido de que la distancia introducida es la longitud del segmento, debe pulsar el botón COMPROBAR, saldrá por pantalla un mensaje con el resultado y se iniciará un nuevo ejercicio hasta que decida salir. Si decide salir y no realizar más ejercicios, puede pulsar el botón SALIR sin que cuente esa última actividad.

Al salir del juego se mostrará por pantalla una gráfica lineal con los resultados. Para más información puede consultar el apartado 4.12.1.

4.8.2. DibujarLinea

La clase *DibujarLinea* calcula de forma aleatoria una longitud acorde al tamaño del dispositivo y la pinta en el recuadro cada vez que se inicia el juego.

Tal y como se ha comentado anteriormente en el apartado 4.6.1, los tamaños de los *layout* dentro de una pantalla se han diseñado en forma porcentual, de tal forma que ocupen siempre el mismo porcentaje de tamaño del dispositivo. En este juego en concreto, el recuadro donde aparece dibujado el segmento cuya longitud hay que averiguar, ocupa un 70% del ancho de la

pantalla y un 80% del alto del mismo, por lo que calcular la longitud máxima que puede medir el segmento es tan sencillo como calcular el 70% de lo que mida realmente la pantalla. Para calcular el tamaño real de la pantalla se ha utilizado la clase *DisplayMetrics* de la librería *Android.Util* explicada en el apartado 4.7.2.3.

Para que el segmento aparezca centrado en el recuadro, se calcula de manera aleatoria un punto cualquiera de todo el recuadro (con ayuda de la clase *Random* () de la librería *Java.Util* explicada en el apartado 4.7.3.) y su punto opuesto al centro, forzando así que al mostrar la línea esta esté siempre centrada. Es decir, si se observa la figura 26, el punto A se calcula aleatoriamente entre los puntos mínimos y máximos de alto (*minHeight* y *maxHeight*) y ancho (*minWidth* y *maxWidth*) de la pantalla. Una vez obtenido este punto se calcula el punto opuesto con respecto al centro (en este caso punto B) y se dibuja la línea que va desde A hasta B, así la línea dibujada siempre estará centrada.

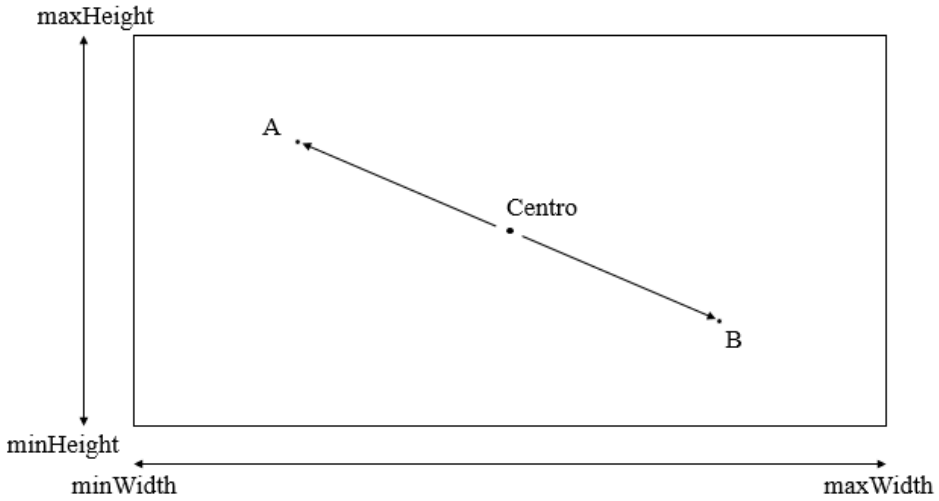

Figura 26. Cálculo del punto opuesto para que las líneas estén centradas

En la figura 27 se puede observar el código fuente de cómo se calculan los puntos mencionados anteriormente.

```
//Obtenemos los valores de las líneas a dibujar siempre centradas
x1 = r.Next(minWidth, maxWidth/2);
y1 = r.Next(minHeight, maxHeight);
x2 = maxWidth-x1;
y2 = maxHeight-y1;
```

Figura 27. Obtención de valores para el segmento del juego ¿CUÁNTO MIDE?

Una vez obtenidos los valores del segmento a mostrar, se genera un nuevo *bitmap*[10], cuyo fondo es el verde claro utilizado en todos los juegos, sobre el que se dibuja (con ayuda de las clases *Canvas* y *Paint* explicados en el apartado 4.7.2.2) el segmento calculado.

Una vez obtenido el bitmap completo, este se le asigna como contenido a la imagen del recuadro.

4.8.3. ReadResult y comprobar resultado

Al pulsar el botón COMPROBAR, la aplicación chequea si el valor introducido en el recuadro es un número válido (ver apartado 4.8.3.1) y, en caso afirmativo, procederá a chequear si el valor introducido por el usuario corresponde a la longitud de la línea.

[10] Un *bitmap* es un formato de imagen de mapa de bits (.BMP)

La clase *ReadResult*, es la encargada de calcular la distancia en centímetros de la línea mostrada redondeándola al entero más próximo. Para ello, se le pasan como parámetros los cuatro valores de los puntos iniciales y finales de la línea dibujada y, de igual forma que para el apartado 4.7.2.3., se calcula la longitud con ayuda del método *Math*.

Dependiendo de si el resultado es correcto o no, aparecen, como se ha mencionado anteriormente, distintos mensajes por pantalla. En caso de acierto aparece el mensaje "¡¡Has acertado!!" como en la figura 28 y, en caso de fallo, aparece el mensaje "Has fallado... La solución era X cm" siendo X el resultado correcto, para que pueda ayudar al alumno con su aprendizaje como se muestra en la figura 29.

Figura 28. Mensaje de acierto en el juego ¿CUÁNTO MIDE?

Figura 29. Mensaje de fallo en el juego ¿CUÁNTO MIDE?

4.8.3.1. Chequeo del valor introducido

Lo primero que realiza la aplicación una vez se pulsa el botón COMPROBAR, es verificar que el valor introducido por el usuario sea un valor válido. En este caso, como al desplegarse el teclado por la pantalla solo se permite introducir números, el único error que puede producirse es que el usuario pulse el botón COMPROBAR sin haber introducido un valor, en cuyo caso aparecería el mensaje por pantalla "Lo siento, para poder comprobar debes introducir un resultado" tal y como se muestra en la figura 30, y volvería a restaurarse la misma pantalla para que el alumno pueda introducir un valor.

Figura 30. Mensaje de valor introducido incorrecto en el juego ¿CUÁNTO MIDE?

4.9. Juego OPERACIONES

El juego OPERACIONES consiste en que el alumno vaya rellenando los recuadros vacíos de varias ecuaciones durante 6 niveles que se van superando independientemente de que el usuario responda correctamente o no. En la figura 31, se muestra un ejemplo de lo que el alumno se va a encontrar al iniciar la actividad.

Figura 31. Pantalla inicial juego OPERACIONES

En los siguientes apartados se explica en detalle las acciones principales que se han desarrollado a la hora de implementar esta actividad, así como las funciones necesarias para el correcto funcionamiento de la misma.

4.9.1. Descripción

El juego OPERACIONES dispone de 6 pantallas diferentes, una para cada nivel del juego, pero todas disponen la misma estructura genérica. La pantalla se divide en tres partes: el enunciado superior donde se indica el nivel en el que se encuentra el alumno, una zona intermedia donde se encuentran los operandos, el operador, el igual y el resultado, y la zona inferior de botones donde poder comprobar el resultado o salir del juego. Lo único que varía entre pantallas es la zona media, ya que, dependiendo del nivel, el usuario debe rellenar unas casillas u otras de la siguiente manera.

- Nivel 1: El recuadro a rellenar es el resultado
- Nivel 2: El recuadro a rellenar es el operando 1
- Nivel 3: Los recuadros a rellenar son el operando 2 y el resultado
- Nivel 4: El recuadro a rellenar es el operador
- Nivel 5: Los recuadros a rellenar son operando 1 y operando 2
- Nivel 6: Los recuadros a rellenar son operando 1 y operador

En la figura 32 se pueden apreciar las diferencias entre los distintos niveles y sus recuadros a rellenar.

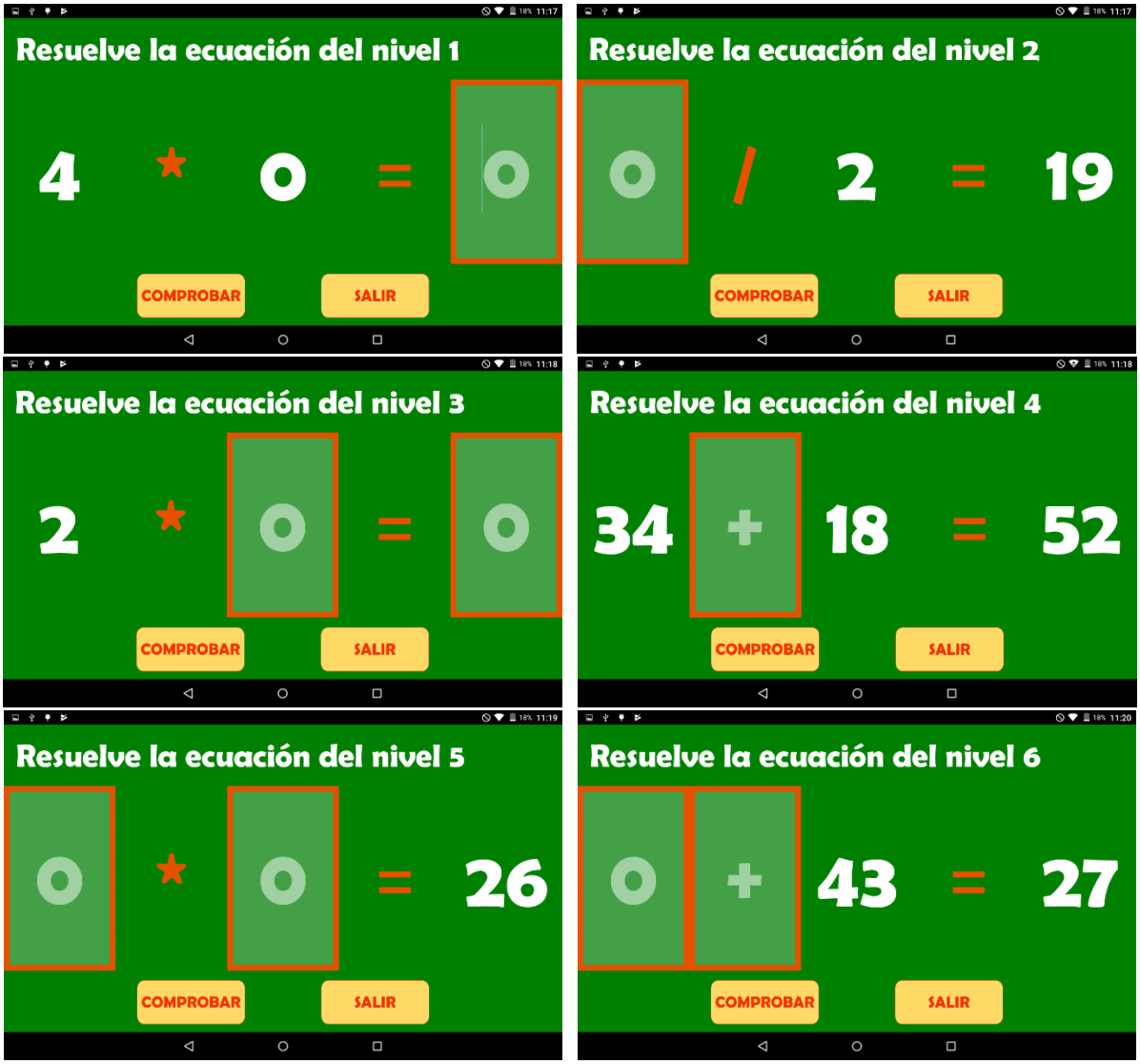

Figura 32. Todas las pantallas iniciales de cada nivel en el juego OPERACIONES

El diseño de este juego ha sido el más complejo de los tres ya que no sirve cualquier valor en los operadores y el resultado, es decir, debido a que la actividad ha sido diseñada para que se resuelva con valores naturales, se ha tenido que definir todos los valores posibles, para cada uno de los niveles, de los que puede disponer una ecuación y así evitar que el usuario deba introducir un número negativo o decimal o que aparezca una ecuación irracional como una división entre cero.

Una vez el alumno esté convencido de que el valor o los valores introducidos resuelven la ecuación, debe pulsar el botón COMPROBAR, saldrá por pantalla un mensaje con el resultado y pasará al ejercicio siguiente hasta que finalice los seis niveles o decida salir. Si decide salir y no realizar más ejercicios, puede pulsar el botón SALIR sin necesidad de terminarlos.

Al salir del juego se mostrará por pantalla una gráfica de barras con los resultados. Para más información puede consultar el apartado 4.12.2.

4.9.2. Parámetros

En este juego ha sido necesario crear dos variables que permiten el correcto flujo de los ejercicios por nivel y de los propios niveles. Estos son el parámetro *level* y el parámetro *comprobarVecesPorNivel*.

4.9.2.1. Level

El parámetro *level* indica el nivel en el que se debe comenzar un ejercicio de tal forma que comienza siempre en 1 al iniciar el juego. Una vez se finaliza el último ejercicio de un nivel, el parámetro aumenta en una unidad salvo en el nivel 6 ya que se finaliza el juego.

Además, es el parámetro utilizado para mostrar el enunciado en cada una de las pantallas (clase definida en el apartado 4.9.4)

4.9.2.2. comprobarVecesPorNivel

El parámetro *comprobarVecesPorNivel* se utiliza cada vez que se comprueba un ejercicio para saber si se han realizado todos los ejercicios del nivel definidos por el parámetro global *VecesPorNivel* definido en el apartado 4.10.2.

En este momento está definido a tres por lo que una vez llega a tres, el parámetro *level* aumenta en uno y este se inicializa a 1 para el nivel siguiente

4.9.3. MostrarLayout

La clase *MostrarLayout* selecciona, dependiendo del valor del parámetro *level*, el *layout* correspondiente cada vez que se inicia el juego o uno de los niveles. Al inicio de la actividad, al parámetro se le asigna el valor 1, es decir, debe mostrarse el *layout* correspondiente al nivel 1. Ya que la aplicación está diseñada para que se realicen tres actividades por nivel, hasta que no se realicen tres ejercicios, el parámetro *level* no aumenta en 1 y no se modifica el *layout*. Además, para cada uno de los ejercicios, es necesario que la aplicación vuelva a detectar los parámetros que el usuario ha de rellenar, es decir, como se muestra en la figura 33, cuando el nivel (el parámetro *level*) aumente al 2, la aplicación tiene que volver a buscar el parámetro *op1* (el operando 1) ya que el usuario va a introducir un valor en él. Para el caso del nivel 3, es necesario volver a buscar los parámetros *op2* (operando 2) y *res* (el resultado), todos los que deba rellenar en la ecuación el alumno.

```
else if (level == 2)
{
    SetContentView(Resource.Layout.OperacionesLayout_L2);
    op1 = FindViewById<EditText>(Resource.Id.editOp1);
    op1.SetTypeface(tf, TypefaceStyle.Normal);
}
else if (level == 3)
{
    SetContentView(Resource.Layout.OperacionesLayout_L3);
    op2 = FindViewById<EditText>(Resource.Id.editOp2);
    res = FindViewById<EditText>(Resource.Id.editRes);
    op2.SetTypeface(tf, TypefaceStyle.Normal);
    res.SetTypeface(tf, TypefaceStyle.Normal);
}
```

Figura 33. Asignación de parámetros al inicio de los niveles 2 y 3 en el juego OPERACIONES

Esto es necesario porque cada nivel tiene su propio *layout* y en muchos se tienen que rellenar los mismos parámetros que en niveles anteriores. Por ejemplo, en el nivel 5 el alumno debe completar los dos operandos, pero el operando 1 también se introdujo en el nivel 2 previamente y, al haber cambiado el *layout*, puede producirse errores en la detección del valor introducido por el alumno si este no se vuelve a detectar como en la figura 33.

4.9.4. MostrarEnunciado

La clase *MostrarEnunciado* es llamada cada vez que se inicia uno de los niveles ya que el enunciado debe modificarse. Con ayuda del parámetro *level* modifica el texto y lo asigna al enunciado del nuevo *layout*.

4.9.5. MostrarValores

La clase *MostrarValores* es la clase donde se definen, dependiendo del nivel, todos los parámetros posibles en las ecuaciones y se asigna, a cada variable que se muestra, uno de ellos de forma aleatoria.

4.9.5.1. Valores posibles del nivel 1

Como se ha mencionado anteriormente, en el nivel 1 el usuario debe rellenar el resultado de una ecuación exacta. Para ello, se ha generado primeramente la variable operador de forma aleatoria entre los caracteres "+", "-", "*" y "/" y, dependiendo del operador, se determinan todos los valores posibles a los operandos 1 y 2.

En caso de que el operador sea "+", tal y como se muestra en la figura 34, se asigna de forma aleatoria entre 1 y 99 un valor entero al operando 1 y un valor entre el 1 y 99 menos el valor aleatorio del operando 1 para el operando 2, de tal forma que la suma máxima sea 99.

```
op1 = r.Next(0, 99);
op2 = r.Next(0, 99 - op1);
```

Figura 34. Operandos posibles para el operador "+" en el nivel 1

En caso de que el operador sea "-", tal y como se muestra en la figura 35, se asigna de forma aleatoria entre 1 y 99 un valor entero al operando 1 y un valor entre el 1 y el valor aleatorio del operando 1 para el operando 2, de tal forma que la resta nunca sea con solución negativa.

```
op1 = r.Next(0, 99);
op2 = r.Next(0, op1);
```

Figura 35. Operandos posibles para el operador "-" en el nivel 1

En caso de que el operador sea "*", tal y como se muestra en la figura 36, se asignan valores aleatorios entre el 1 y el 10 a ambos operandos siendo el valor máximo de la ecuación 100.

```
op1 = r.Next(0, 10);
op2 = r.Next(0, 10);
```

Figura 36. Operandos posibles para el operador "*" en el nivel 1

En caso de que el operador sea "/", tal y como se muestra en la figura 37, se asigna un valor aleatorio entre 1 y 10 al operando 2 y se genera una lista con todos los valores posibles para el operando 1 dependiendo del valor del operando 2 de tal forma que la división sea exacta. De entre esos valores almacenados en la lista se selecciona uno aleatorio para el operando 1.

```
op2 = r.Next(1, 10);

for (int i=1; i < 11; i++)
{
    valor = op2 * i;
    list.Add(valor);
}

op1 = list[r.Next(0, 9)];
```

Figura 37. Operandos posibles para el operador "/" en el nivel 1

4.9.5.2. Valores posibles del nivel 2

Como se ha mencionado anteriormente, en el nivel 2 el usuario debe rellenar el operando 1 de una ecuación exacta. Para ello, se ha generado primeramente la variable operador de forma aleatoria entre los caracteres "+", "-", "*" y "/" y, dependiendo del operador, se determinan todos los valores posibles del operando 2 y el resultado.

En caso de que el operador sea "+", tal y como se muestra en la figura 38, se asigna de forma aleatoria entre 0 y 98 un valor entero al operando 2 y un valor entre el operando 2 y 99 para el resultado, de tal forma que el resultado siempre sea mayor que el operando 2 y el operando 1 pueda ser positivo.

```
op2 = r.Next(0, 98);
res = r.Next(op2, 99);
```

Figura 38. Operandos posibles para el operador "+" en el nivel 2

En caso de que el operador sea "-", tal y como se muestra en la figura 39, se asigna de forma aleatoria entre 0 y 99 un valor entero al operando 2 y un valor entre el 0 y 99 menos el valor aleatorio del operando 2 para el resultado, de tal forma que el valor posible para el operando 1 este entre 0 y 99.

```
op2 = r.Next(0, 99);
res = r.Next(0, 99-op2);
```

Figura 39. Operandos posibles para el operador "-" en el nivel 2

En caso de que el operador sea "*", tal y como se muestra en la figura 40, se asigna un valor aleatorio entre 0 y 10 al operando 2 y se genera una lista con todos los valores posibles para el resultado dependiendo del valor del operando 2. De entre esos valores almacenados en la lista se selecciona uno para el resultado de tal forma que el operando 1 sea un natural del 0 al 10.

```
op2 = r.Next(0, 10);
for (int i = 0; i < 11; i++)
{
    valor = op2 * i;
    list.Add(valor);
}

res = list[r.Next(0, 10)];
```

Figura 40. Operandos posibles para el operador "*" en el nivel 2

En caso de que el operador sea "/", tal y como se muestra en la figura 41, se asigna un valor aleatorio entre 1 y 10 al operando 2 y, al resultado, un valor entre 1 y el entero más próximo de la división de 100 entre el valor aleatorio del operando 2, de tal forma que los valores posibles para el operando 1 no superen 100.

```
op2 = r.Next(1, 10);
res = r.Next(1, (int)Math.Round(100/(double)op2));
```

Figura 41. Operandos posibles para el operador "/" en el nivel 2

Por ejemplo, suponiendo que el valor aleatorio del operando 2 es el 3, los valores posibles para el resultado serían los mostrados en la ecuación 4.

$$res = \left[1, Math.Round\left(\frac{100}{3}\right)\right] = [1, 33] \quad (4)$$

Los valores mínimo y máximo que se muestran en la ecuación 5 para el operando 1 dependerían del valor mínimo y máximo del resultado.

$$\begin{aligned} op1_{min}/3 = 1 \rightarrow\ op1_{min} = 3 \\ op1_{max}/3 = 33 \rightarrow\ op1_{max} = 99 \end{aligned} \quad (5)$$

4.9.5.3. Valores posibles del nivel 3

Como se ha mencionado anteriormente, en el nivel 3 el usuario debe rellenar el operando 2 y el resultado de una ecuación. En este caso se ha generado primeramente la variable operador de forma aleatoria entre los caracteres "+", "-", "*" y "/" y, dependiendo del que se obtenga, se eligen de forma aleatoria los del operando 1.

En caso de que el operador sea "+", "-" o "/" tal y como se muestra en la figura 42, se asigna de forma aleatoria entre 0 y 99 un valor entero al operando 1 ya que, al tener el usuario que introducir un valor para el operando 2 y el resultado, puede utilizarse cualquiera de ellos.

```
op1 = r.Next(0, 99);
```

Figura 42. Operandos posibles para los operadores "+", "-" y "/" en el nivel 3

En caso de que el operador sea "*", tal y como se muestra en la figura 43, se asigna un valor aleatorio entre 0 y 10 al operando 1 ya que la aplicación está diseñada para las tablas de multiplicar del 1 al 10.

```
op1 = r.Next(0, 10);
```

Figura 43. Operandos posibles para el operador "*" en el nivel 3

4.9.5.4.Valores posibles del nivel 4

Como se ha mencionado anteriormente, en el nivel 4 el usuario debe seleccionar un operador de entre los caracteres "+", "-", "*" y "/" para que la ecuación se cumpla.

De cara a que aleatoriamente puedan colocarse todos los caracteres y no siempre sea el mismo, los valores de los operandos y el resultado dependen de un operador imaginario aleatorio. Es decir, como puede observarse en la figura 44, primero se elige aleatoriamente un operador, y, dependiendo del operador, se asignan, de entre todos los valores posibles, los valores de los operandos y del resultado.

```
String OperandoString = "+-*/";
char[] randomOperando = new char[1];
randomOperando[0] = OperandoString[r.Next(OperandoString.Length)];
operando = new string(randomOperando);
```

Figura 44. Selección del operando aleatorio en el nivel 4

En caso de que el operador sea "+", tal y como se muestra en la figura 45, se asigna de forma aleatoria entre 1 y 99 un valor entero al operando 1, un valor entre el 1 y 99 menos el valor aleatorio del operando 1 para el operando 2, de tal forma que la suma máxima sea 99 y el resultado de la operación a la variable resultado.

```
op1 = r.Next(0, 99);
op2 = r.Next(0, 99 - op1);
res = op1 + op2;
```

Figura 45. Operandos posibles y resultado para el operador "+" en el nivel 4

En caso de que el operador sea “-”, tal y como se muestra en la figura 35, se asigna de forma aleatoria entre 1 y 99 un valor entero al operando 1 y un valor entre el 1 y el valor aleatorio del operando 1 para el operando 2, de tal forma que la resta nunca sea con solución negativa.

```
op1 = r.Next(0, 99);
op2 = r.Next(0, op1);
res = op1 - op2;
```

Figura 46. Operandos posibles y resultado para el operador "-" en el nivel 4

En caso de que el operador sea “*”, tal y como se muestra en la figura 36, se asignan valores aleatorios entre el 1 y el 10 a ambos operandos siendo el valor máximo de la ecuación 100.

```
op1 = r.Next(0, 10);
op2 = r.Next(0, 10);
res = op2 * op1;
```

Figura 47. Operandos posibles y resultado para el operador "*" en el nivel 4

En caso de que el operador sea “/”, tal y como se muestra en la figura 37, se asigna un valor aleatorio entre 1 y 10 al operando 2 y se genera una lista con todos los valores posibles para el operando 1 dependiendo del valor del operando 2 de tal forma que la división sea exacta. De entre esos valores almacenados en la lista se selecciona uno aleatorio para el operando 1.

```
op2 = r.Next(1, 10);

for (int i = 1; i < 11; i++)
{
    valor = op2 * i;
    list.Add(valor);
}

op1 = list[r.Next(0, 9)];
res = op1 / op2;
```

Figura 48. Operandos posibles y resultado para el operador "/" en el nivel 4

4.9.5.5.Valores posibles del nivel 5

Como se ha mencionado anteriormente, en el nivel 5 el usuario debe rellenar los operandos 1 y 2 de una ecuación para que esta sea certera. En este caso, como puede observarse en la figura 49, se ha generado primeramente la variable operador de forma aleatoria entre los caracteres “+”, “-”, “*” y “/”.

```
String OperandoString = "+-*/";
char[] randomOperando = new char[1];
randomOperando[0] = OperandoString[r.Next(OperandoString.Length)];
operando = new string(randomOperando);
```

Figura 49. Selección del operando aleatorio en el nivel 5

Al resultado, como se aprecia en la figura 50, se le asigna cualquier valor entre el 0 y el 99 ya que en este caso no hay ninguna limitación.

```
res = r.Next(0, 99);
```

Figura 50. Resultados posibles para cualquier operador en el nivel 5

4.9.5.6.Valores posibles del nivel 6

Como se ha mencionado anteriormente, en el nivel 6 el usuario debe rellenar el operando 1 y el operador de una ecuación para que esta sea certera. De cara a que aleatoriamente puedan colocarse todos los caracteres y no siempre sea el mismo, los valores del operando 2 y el resultado dependen de un operador imaginario aleatorio. Es decir, como puede observarse en la figura 51, primero se elige aleatoriamente un operador imaginario y, dependiendo del operador, se asignan, de entre todos los valores posibles, los valores al operando 2 y al resultado.

```
String OperandoString = "+-*/";
char[] randomOperando = new char[1];
randomOperando[0] = OperandoString[r.Next(OperandoString.Length)];
operando = new string(randomOperando);
```

Figura 51. Selección del operando aleatorio en el nivel 6

En caso de que el operador sea "+", tal y como se muestra en la figura 52, se asigna de forma aleatoria entre 0 y 98 un valor entero al operando 2 y un valor entre el operando 2 y 99 para el resultado, de tal forma que el resultado siempre sea mayor que el operando 2 y el operando 1 pueda ser positivo.

```
op2 = r.Next(0, 98);
res = r.Next(op2, 99);
```

Figura 52. Operandos 2 posibles y resultado para el operador "+" en el nivel 6

En caso de que el operador sea "-", tal y como se muestra en la figura 53, se asigna de forma aleatoria entre 0 y 99 un valor entero al operando 2 y un valor entre el 0 y 99 menos el valor aleatorio del operando 2 para el resultado, de tal forma que el valor posible para el operando 1 este entre 0 y 99.

```
op2 = r.Next(0, 99);
res = r.Next(0, 99-op2);
```

Figura 53. Operandos 2 posibles y resultado para el operador "-" en el nivel 6

En caso de que el operador sea "*", tal y como se muestra en la figura 54, se asigna un valor aleatorio entre 0 y 10 al operando 2 y se genera una lista con todos los valores posibles para el resultado dependiendo del valor del operando 2. De entre esos valores almacenados en la lista se selecciona uno para el resultado de tal forma que el operando 1 sea un natural del 0 al 10.

```
op2 = r.Next(0, 10);
for (int i = 0; i < 11; i++)
{
    valor = op2 * i;
    list.Add(valor);
}

res = list[r.Next(0, 10)];
```

Figura 54. Operandos 2 posibles y resultado para el operador "*" en el nivel 6

En caso de que el operador sea "/", tal y como se muestra en la figura 55, se asigna un valor aleatorio entre 1 y 10 al operando 2 y, al resultado, un valor entre 1 y el entero más próximo de la división de 100 entre el valor aleatorio del operando 2, de tal forma que los valores posibles para el operando 1 no superen 100.

```
op2 = r.Next(1, 10);
res = r.Next(1, (int)Math.Round(100/(double)op2));
```

Figura 55. Operandos 2 posibles y resultado para el operador "/" en el nivel 6

4.9.6. Comprobar resultado

Una vez pasados los chequeos que se explicarán en el apartado 4.9.6.1., se verifica la solución propuesta por el alumno. Para ello se han programado seis clases (ComprobarResultado1 al 6) en las que se calculan las soluciones de los ejercicios y después se comparan con las propuestas del alumno.

Por ejemplo, para el caso del nivel 1 en el que hay que introducir una respuesta a la ecuación, se calcula la respuesta correcta con los valores que se han mostrado por pantalla y se compara con la indicada por el alumno. En caso de que sean iguales se devuelve un valor true y en caso de fallo *false*.

Existen ciertas excepciones que se han tenido que tener en cuenta a la hora de comprobar los resultados para los casos en los que se tiene que introducir un operador (nivel 4) o se tienen que rellenar dos recuadros (niveles 3, 5 y 6) ya que existe la posibilidad de que no solo haya una posible respuesta. En el caso de los operadores puede ser que más de uno sea válido como por ejemplo el mostrado en la ecuación 6, donde el operando puede ser "+" o "*".

$$2 \cdots 2 = 4 \quad // \quad 2 + 2 = 4 \quad // \quad 2 * 2 = 4 \tag{6}$$

En el caso de tener que rellenar dos recuadros existen muchas más posibilidades ya que prácticamente puede usarse cualquier valor para uno de los dos huecos y el otro depender del introducido.

En la figura 56 se muestra cómo para el caso del nivel 4 se ha calculado el resultado verdadero dependiendo del valor introducido por el alumno, es decir, el resultado (*resultadoVerdadero*) que daría la ecuación con los operandos mostrados (*operador1* y *operador2*) y el operador del alumno (*oper*). Una vez obtenido se compara con el que se ha mostrado en el ejercicio (*resultado*), devolviendo un valor *true* si son iguales o *false* si son distintos.

```
private bool ComprobarResultado4(string oper)
{
    int resultadoVerdadero = 0;

    switch (oper)
    {
        case "+":
            resultadoVerdadero = Convert.ToInt32(operador1) + Convert.ToInt32(operador2);
            break;
        case "-":
            resultadoVerdadero = Convert.ToInt32(operador1) - Convert.ToInt32(operador2);
            break;
        case "*":
            resultadoVerdadero = Convert.ToInt32(operador1) * Convert.ToInt32(operador2);
            break;
        case "/":
            if (Convert.ToInt32(operador2) == 0)
            {
                return false;
            }
            else
            {
                resultadoVerdadero = Convert.ToInt32(operador1) / Convert.ToInt32(operador2);
            }
            break;
    }
    if (resultadoVerdadero == Convert.ToInt32(resultado))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Figura 56. Comprobación operador en el nivel 4 del juego OPERACIONES

Una vez se sabe si la solución es correcta o incorrecta, la aplicación muestra distintos mensajes dependiendo del nivel y si el usuario va a realizar un nuevo ejercicio en el mismo nivel o pasa al siguiente nivel. Los mensajes por pantalla dependiendo de los casos mencionados vienen descritos en la tabla 4.

Tabla 4. Mensajes por pantalla por niveles al comprobar en el juego OPERACIONES

Nivel	Pasa al siguiente nivel	Acierta o falla	Mensaje
1	No	Acierta	¡¡ Correcto !!
1	No	Falla	Lo siento, no has acertado... Inténtalo de nuevo
1	Sí	Acierta	¡¡ Correcto !! Veamos el nivel 2...
1	Sí	Falla	Lo siento, no has acertado... Veamos el nivel 2
2	No	Acierta	¡¡ Exacto !!
2	No	Falla	Lo siento, no has acertado... Inténtalo de nuevo
2	Sí	Acierta	¡¡ Exacto !! Veamos el nivel 3...
2	Sí	Falla	Lo siento, no has acertado... Veamos el nivel 3
3	No	Acierta	¡¡Has respondido correctamente!!
3	No	Falla	Lo siento, no has acertado... Inténtalo de nuevo
3	Sí	Acierta	¡¡Has respondido correctamente!! Veamos el nivel 4...
3	Sí	Falla	Lo siento, no has acertado... Inténtalo de nuevo en el nivel 4
4	No	Acierta	¡¡Genial!!
4	No	Falla	Lo siento, no has acertado... Inténtalo de nuevo
4	Sí	Acierta	¡¡Genial!! Veamos el nivel 5...
4	Sí	Falla	Lo siento, no has acertado... Veamos el nivel 5
5	No	Acierta	¡¡ Muy bien !!
5	No	Falla	Lo siento, no has acertado... Inténtalo de nuevo
5	Sí	Acierta	¡¡Muy bien!! Veamos el nivel 6...
5	Sí	Falla	Lo siento, no has acertado... Veamos el nivel 6
6	No	Acierta	¡¡ Correcto !!
6	No	Falla	Lo siento, no has acertado... Inténtalo de nuevo
6	Sí	Acierta	¡¡ Correcto !! Veamos tus resultados...
6	Sí	Falla	Lo siento, no has acertado... Veamos tus resultados

4.9.6.1. Chequeo del valor introducido

Dependiendo del nivel en el que se encuentre un alumno, al pulsar el botón COMPROBAR se realizan ciertos chequeos para verificar si el valor introducido por el alumno es válido o no.

En cualquier caso, se chequea que el usuario haya introducido algún valor en el recuadro, devolviendo, como se puede ver en la figura 57, un valor *false* en caso de que no lo haya introducido para que se mantenga en el mismo ejercicio.

```
case 1:

    res = FindViewById<EditText>(Resource.Id.editRes);

    if (res.Text == "")
    {
        cadena = ("Debes introducir un resultado en el recuadro naranja");
        text.Text = cadena;
        toast.Show();
        return false;
    }
    else
    {
        return true;
    }
```

Figura 57. Chequeo del resultado en el nivel 1 del juego OPERACIONES

Además, si el usuario no ha introducido ningún valor y ha dado al botón COMPROBAR por error, aparece un mensaje por pantalla:

Para el caso del nivel 1 en el que falte el resultado el texto será “Debes introducir un resultado en el recuadro naranja” tal y como se muestra en la figura 58.

Figura 58. Mensaje por pantalla de error en el chequeo del resultado en el nivel 1 del juego OPERACIONES

Para el caso del nivel 2 en el que falte el operando aparece el mensaje de texto “Debes introducir un número en el recuadro naranja” tal y como se muestra en la figura 59.

Figura 59. Mensaje por pantalla de error en el chequeo del operando 1 en el nivel 2 del juego OPERACIONES

Para el caso del nivel 4 en el que no se añada ningún operador aparece el mensaje de texto "Debes introducir un operador en el recuadro naranja" tal y como se muestra en la figura 60.

Figura 60. Mensaje por pantalla de error en el chequeo del operador en el nivel 4 del juego OPERACIONES

En los niveles 3 y 5 en el que hay que rellenar dos recuadros, aparece el mensaje de texto "Debes introducir un número en los recuadros naranjas" tal y como se muestra en la figura 61.

Figura 61. Mensaje por pantalla de error en el chequeo del operando 2 y el resultado en el nivel 3 del juego OPERACIONES

Para el nivel 6, se comprueba primeramente que se haya introducido un operador y en caso de que no se haya introducido, se muestra un mensaje parecido al de la figura 60 especificando que es el segundo recuadro naranja. En caso de que haya introducido algún carácter, se chequea que se haya introducido algún número para el operando 1, mostrando en caso de que esté vacío un mensaje parecido al de la figura 58 especificando que es el primer recuadro naranja.

Debido a que para introducir el valor del operador aparece un teclado completo, se ha diseñado un chequeo adicional para los niveles 4 y 6 donde si el valor introducido para el operador es distinto a cualquiera de los valores permitidos ("+", "-", "*" y "/") aparece un mensaje por pantalla como en la figura 62 indicando los caracteres válidos y repitiendo el mismo ejercicio.

Figura 62. Mensaje por pantalla de error en el chequeo del carácter del operador en el nivel 4 del juego OPERACIONES

Una vez pasados los filtros, se comprueba el resultado tal y como se explica en el apartado 4.9.6.

4.10. Variables globales

Se han definido tres variables globales para el correcto funcionamiento de la aplicación.

4.10.1. UserName

La variable global *UserName* se utiliza para almacenar el nombre del alumno que va a realizar los juegos introducido en la pantalla "MainActivity". Este valor se almacena junto con los resultados de cada juego que realizan los alumnos en la base de datos y se utiliza como parte de los filtros cuando se realizan búsquedas en la misma. Además, aparece en un mensaje por pantalla al pulsar el botón JUGAR como se muestra en la figura 63 al inicio del juego.

En caso de que el alumno preste la Tablet a otro compañero, este puede modificar su nombre durante el juego retrocediendo en las pantallas con los botones de control de la Tablet hasta la pantalla de "MainActivity" y modificar el recuadro de nombre sin afectar a los datos ya almacenados en la BBDD.

Figura 63. Mensaje por pantalla de bienvenida

4.10.2. VecesPorNivel

La variable global *VecesPorNivel* almacena el valor del número de ejercicios por nivel que debe realizar el alumno en el juego de operaciones. Actualmente está definido a 3 pero puede modificarse en el fichero "Variables".

4.10.3. Salir

La variable global *Salir* se utiliza para detectar si un alumno ha realizado algún ejercicio anteriormente en la misma sesión de tal forma que la aplicación no aumente el contador de sesiones cada vez que se realice un ejercicio. Es decir, como se inicializan los juegos para variar los enunciados, es necesaria una variable que indique a la APP si el usuario es la primera vez que accede al juego o no. Para ello se ha definido el código de la figura 64 al inicio de cada ejercicio, de tal forma que al pulsar el botón COMPROBAR se modifique este parámetro a un valor *false* y así a partir del segundo ejercicio el valor de sesión se obtiene de la última línea guardada y no aumenta en uno. Cuando el jugador decide finalizar y pulsa el botón SALIR, si ha realizado algún ejercicio, el valor de la variable será *false*, mostrará todos los resultados para la sesión y modificará l variable *Salir* a true. Si el usuario no ha realizado ningún ejercicio, el valor de la variable *Salir* seguirá siendo *true* por lo que saldrá sin mostrar ningún usuario.

```
if (Variables.Salir == true)
{
    sesion = baseDatosFunciones.ObtenerSesion("JM");
    sesion++;
}
else
{
    sesion = baseDatosFunciones.ObtenerSesion("JM");
}
```

Figura 64. Comprobación de sesión con la variable *Salir*

4.11. Base de datos

En esta aplicación se ha diseñado una base de datos que almacene cada uno de los resultados comprobados en cada actividad y así poder mostrar los datos de la última sesión al finalizar un juego.

Para ordenar los datos de cada juego de una forma más sencilla y debido a que es necesario guardar distintos parámetros dependiendo del juego, se han creado tres tablas de datos distintas, una para cada uno de los juegos:

- Juego DIBUJA: ResultadosJD
- Juego ¿CUÁNTO MIDE?: ResultadosJM
- Juego OPERACIONES: ResultadosJO

En los siguientes apartados se explican los principales detalles de cada una de las funciones creadas para el correcto funcionamiento de la base de datos.

4.11.1. Cargar y cerrar la base de datos

Cada vez que se ejecuta una función en la que haya que escribir o leer datos de la base de datos, es necesario cargar la base de datos con todas sus tablas y cerrarla una vez finalizado su uso.

En la figura 65 se muestran las clases *CargarBaseDatos* y *CerrarBaseDatos* diseñadas para cargar y cerrar una base de datos. Primero, los parámetros *folder* y *dbPath* obtienen la carpeta donde se almacena la BBDD y, gracias a ella, la ruta donde está la BBDD con nombre "resultados.db". Una vez obtenida la ruta, se crea la conexión con la BBDD y se crean o cargan las tres tablas mencionadas anteriormente.

```
public SQLiteConnection CargarBaseDatos()
{
    string folder = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);
    string dbPath = System.IO.Path.Combine(folder, "resultados.db");

    SQLiteConnection db = new SQLiteConnection(dbPath);

    db.CreateTable<ResultadosJO>();
    db.CreateTable<ResultadosJD>();
    db.CreateTable<ResultadosJM>();

    return db;
}

private void CerrarBaseDatos(SQLiteConnection db)
{
    db.Close();
}
```

Figura 65. Cargar y cerrar la base de datos

Cuando se finaliza la carga u obtención de datos de la BBDD siempre es necesario cerrarla con la función *Close* (). En este caso, se ha decidido crear la clase *CerrarBaseDatos* por si es necesario en un futuro realizar alguna tarea adicional al cerrar la BBDD.

4.11.2. ObtenerSesion

La función *ObtenerSesion* detecta el valor de la sesión actual del alumno para un juego en concreto. Para ello, cada vez que se solicita una sesión, es necesario pasar como variable las iniciales del juego del que se va a obtener la sesión:

- Juego DIBUJA: "JD"
- Juego ¿CUÁNTO MIDE?: "JM"
- Juego OPERACIONES: "JO"

Gracias al parámetro, la función sabe en qué tabla debe buscar la sesión y recorre todos los valores de sesión que posee el nombre del usuario actual (almacenado en la variable global *UserName*) quedándose con el último de ellos (la última sesión) y lo devuelve como parámetro entero.

4.11.3. Agregar datos a la base de datos

Las funciones *AgregarResultadosJD*, *AgregarResultadosJM* y *AgregarResultadosJO* almacenan los valores que ha obtenido el alumno para cada uno de los ejercicios en la base de datos. Para los juegos de medir y dibujar, se almacenan los siguientes valores.

- *Id*: Identificador que autoincrementa para cada entrada
- *UserName*: Nombre
- *Session*: Sesión
- *Date*: Fecha y hora
- *RealResult*: Resultado correcto
- *UserResult*: Resultado del usuario
- *FinalResult*: Resultado final, *true* en caso de acierto y *false* en caso de fallo

Para el juego de OPERACIONES se almacenan los siguientes valores. Se almacenan todos estos valores a pesar de que no todos se utilicen en esta aplicación de cara a que, si se realiza una nueva versión de la misma, puedan crearse gráficas más interesantes para el profesor a la hora de estudiar los fallos y aciertos de sus alumnos.

- *Id*: Identificador que autoincrementa para cada entrada
- *UserName*: Nombre
- *Session*: Sesión
- *Date*: Fecha y hora
- *Level*: Nivel
- *RealOpr*: Operador mostrado (si no procede: a)
- *RealOp1*: Operando 1 mostrado (si no procede: -1)
- *RealOp2*: Operando 2 mostrado (si no procede: -1)
- *RealRes*: Resultado mostrado (si no procede: -1)
- *UserOpr*: Operador introducido por el usuario (si no procede: a)
- *UserOp1*: Operando 1 introducido por el usuario (si no procede: -1)
- *UserOp2*: Operando 2 introducido por el usuario (si no procede: -1)
- *UserRes*: Resultado introducido por el usuario (si no procede: -1)
- *FinalResult*: Resultado final, *true* en caso de acierto y *false* en caso de fallo

Cada vez que inicia una de las funciones, es necesario cargar la base de datos en el parámetro *db*, generar un parámetro tipo *ResultadoJM*, *ResultadoJD* o *ResultadoJO* dependiendo del juego del que se van a guardar los datos con todos los valores mencionados anteriormente, insertar el parámetro en la BBDD y, finalmente, cerrarla tal y como se muestra en la figura 66.

```
public void AgregarResultadosJM(int sesion, int realResult, int userResult, bool result)
{
    SQLiteConnection db = CargarBaseDatos();

    ResultadosJM resultadoJM = new ResultadosJM { UserName = usuario, Session = sesion, ... };

    db.Insert(resultadoJM);
    CerrarBaseDatos(db);
}
```

Figura 66. Agregar datos en la base de datos

4.11.4. Mostrar resultados

Las funciones *MostrarResultadosJM*, *MostrarResultadosJD* y *MostrarResultadosJO* se utilizan para obtener los valores de la última sesión de un alumno para cada uno de los juegos de manera que puedan mostrarse después en una gráfica.

En esta aplicación, se obtienen distintos valores para cada uno de los juegos al finalizar una sesión. Para los juegos ¿CUÁNTO MIDE? y DIBUJA se obtienen los mismos:

- Resultados correctos (*RealResult*)
- Resultados introducidos por el usuario (*UserResult*)

Cada una es una línea de datos de tal forma que se puedan mostrar ambas en la gráfica y compararlas.

Para el juego OPERACIONES, no se muestran datos recogidos como tal, sino que se tratan para mostrarse de una forma más entendible por el alumno. En este caso, se ha decidido contar los aciertos y fallos por niveles para poder mostrarlos en una gráfica de barras por colores. Para ello se han generado dos parámetros por nivel, uno que recuente todas las entradas que se reciben en un nivel y el otro que cuente solamente si el parámetro de resultado final para esa fila es true.

Es importante señalar que en estas funciones se definen bastantes características sobre el diseño de las gráficas que se comentan en el apartado 4.12 de gráficas y resultados.

4.11.5. Obtener aciertos

Cada vez que se muestran los resultados al salir de un juego, debajo de la gráfica con los resultados se muestra en porcentaje la efectividad de la sesión en cuanto a aciertos y el número de aciertos en comparación con el número de ejercicios realizados. Para ello, es necesario contar el número de respuestas correctas que se han obtenido en la sesión y esto lo realiza la función *ObtenerAciertos*. Lo que hace esta función es obtener todos los resultados guardados en la última sesión del alumno y, con la ayuda de un contador, verificar si el campo *FinalResult* es *true* o *false*.

4.11.6. Obtener total de resultados

Para obtener el número total de ejercicios realizados por el alumno se utiliza la función *ObtenerTotal*. Lo que hace esta función es obtener todos los resultados guardados en la última sesión del alumno y contar todas las filas que se obtengan.

4.12. Gráficas y resultados

Esta aplicación se ha diseñado para que al salir o finalizar un juego se muestre por pantalla los resultados de la última sesión y así el alumno pueda ver su progreso en cada actividad realizada. Para ello, se ha utilizado la librería *MPAndroidChart* que ofrece una amplia variedad de gráficas y posibilidades a la hora de mostrar los datos, puede obtenerse más información sobre todas sus posibilidades desde su página en la plataforma *GitHub*[11].

Para una mejor visualización de los resultados, se ha optado por mostrar los resultados de los juegos ¿CUÁNTO MIDE? y DIBUJA de forma lineal como se muestra en la figura 67 donde se aprecian los resultados de cuatro intentos habiendo acertado los ejercicios 2 y 3.

[11] GitHub es una plataforma donde alojar proyectos para el desarrollo colaborativo. La página de *MPAndroidChart* es https://github.com/PhilJay/MPAndroidChart/tree/master/MPChartExample

Figura 67. Resultados en gráfica lineal

Los resultados del juego OPERACIONES se representan mediante una gráfica de barras como se muestra en la figura 68. En este caso no se representan directamente datos almacenados, si no que se recuentan los aciertos y niveles para que el usuario aprecie los niveles que se le han dado mejor o peor y visualmente entienda sus progresos.

Figura 68. Resultados en gráfica de barras

Todas las gráficas se muestran en *layouts* superpuestos al del juego que se eliminan pulsando el botón OK de la parte inferior de las figuras anteriores.

A pesar de utilizar distintos tipos de gráficas, existen ciertos parámetros comunes para cualquier tipo de gráfica de la librería *MPAndroidChart* por lo que el capítulo se ha dividido en un apartado común para todas las gráficas y otros capítulos más específicos de las gráficas lineales y de barras.

4.12.1. Parámetros genéricos

Para el caso de la muestra de los resultados, se ha querido realizar de tal forma que sea lo más entendible y provechoso para los alumnos. En este apartado se van a explicar los parámetros genéricos más importantes y comunes a las tres gráficas, especificando las diferencias que se han tenido que considerar para cada una de ellas.

4.12.1.1. Layout

Los *layout* que definen el fondo de las gráficas mostradas, se dividen en cuatro partes:

- El título RESULTADOS
- La parte más notable de la pantalla donde se muestran los datos
- Una línea informativa de los aciertos respecto al total de ejercicios realizados
- Una línea informativa del porcentaje de efectividad de la sesión.

Los tres *layout* (*MostrarResultadosJD*, *MostrarResultadosJM* y *MostrarResultadosJO*) tienen la misma estructura, incluso disponen del mismo código, el de la figura 69, para la parte donde se muestran los datos ya que está definido como una gráfica combinada (*CombinedChart*) independientemente del tipo de gráfica que se desee mostrar.

```
<com.github.mikephil.charting.charts.CombinedChart
    android:layout_width="0dp"
    android:layout_height="fill_parent"
    android:layout_weight="100"
    android:id="@+id/chart"
    android:background="#ffa5d6a7"
    android:animateLayoutChanges="false" />
```

Figura 69. Parte del *layout* para mostrar gráficas

4.12.1.2. Diferencias en el diseño de ejes

En el caso del diseño de los ejes, se debe configurar por separado los ejes izquierdo y derecho en los ejes Y y el eje X por lo que cualquier cambio que quiera realizarse a los tres ejes es necesario especificarlo para los tres. Finalmente, se ha decidido que todos los ejes y sus valores estén definidos por el color RGB: 127,96,0 de un tono amarillo oscuro, de un tamaño fijo y el mismo tipo de letra que el que se ha definido para la aplicación. Además, se ha fijado como color principal para el fondo de la gráfica, un color RGB: 153, 255, 153 de un tono verde muy claro.

Aun así, para una mejor visualización de los datos, existen pequeñas diferencias entre las gráficas lineales y la de barras. Para la gráfica lineal, se ha decidido que los ejes estén definidos en un valor por encima y por debajo a los valores máximo y mínimo que se vayan a mostrar, de tal forma que los valores no coincidan con el final de la gráfica. Para el caso de la gráfica de barras, el valor más bajo es 0 y el valor más alto es 3 ya que se ha configurado para que el alumno realice tres ejercicios por nivel. En el caso de que se modifique este parámetro global, sería necesario modificarlo en los comandos de la figura 70 dentro de la actividad *OperacionesActivity* ya que, si no, no se mostrarían los valores a partir del tercer ejercicio.

```
chart.AxisLeft.AxisMaximum = 3;
chart.AxisLeft.AxisMinimum = 0;
chart.AxisRight.AxisMaximum = 3;
chart.AxisRight.AxisMinimum = 0;
```

Figura 70. Valores definidos para los ejes Y en el juego OPERACIONES

Otro detalle que resaltar es que para el eje X de la gráfica de barras del juego OPERACIONES se ha generado una lista de valores desde "Nivel 1" hasta "Nivel 6" en vez de valores numéricos. Además, se indica para cada uno de los niveles el parámetro que ha tenido que completar el usuario, siendo "Res" el resultado, "Oper" el operador, "Op1" el operando 1 y "Op2" el operando 2. Para ello, tal y como se muestra en la figura 71, es necesario crear una lista de variables (en este caso *labels*), asignar el parámetro *SetDrawLabels* a true y asignar con la clase *ValueFormatter* un *IndexAxisValueFormatter* pasando como parámetro la lista de niveles. En este caso no se ha insertado para el valor 0 ningún parámetro ya que los valores comienzan a asignarse desde el valor 1.

```
chart.XAxis.SetDrawLabels(true);
String[] labels = new String[] { "", "Nivel 1: "+"Res", "Nivel 2: "+"Op 1", "Nivel 3: "+"Op2 y Res",
    "Nivel 4: "+"Oper", "Nivel 5: "+"Op1 y Op2", "Nivel 6: "+"Op1 y Oper" };
chart.XAxis.ValueFormatter=new IndexAxisValueFormatter(labels);
```

Figura 71. Asignar valores a los ejes en *MPAndroidChart*

4.12.1.3. Título, aciertos totales y efectividad

Según como se ha descrito en el apartado 4.12.1.1. existen cuatro partes de la pantalla de las que tres muestran exactamente lo mismo para los tres juegos: el título, los aciertos totales respecto del total de ejercicios realizados en la sesión y la efectividad.

En los apartados 4.11.5 y 4.11.6, se explica como la base de datos calcula los valores que se muestran por pantalla. En la figura 72, se ve cómo se coge el valor de los textos, se les asigna el tipo de letra elegido para la APP, se obtienen los aciertos totales (*aciertos*) del total de ejercicios (*totales*) indicando el juego en cuestión, se calcula el porcentaje de efectividad y se pasa a los textos el valor del nuevo mensaje a mostrar.

```
TextView titulo = FindViewById<TextView>(Resource.Id.tituloResultados);
TextView textAciertosTotales = FindViewById<TextView>(Resource.Id.textAciertosTotales);
TextView porcentajeAciertosTotales = FindViewById<TextView>(Resource.Id.porcentajeAciertosTotales);

titulo.SetTypeface(tf, TypefaceStyle.Normal);
textAciertosTotales.SetTypeface(tf, TypefaceStyle.Normal);
porcentajeAciertosTotales.SetTypeface(tf, TypefaceStyle.Normal);

aciertos = baseDatosFunciones.ObtenerAciertos("JO");
totales = baseDatosFunciones.ObtenerTotal("JO");
double porcentajeSinRedondeo = aciertos * 100 / totales;
double porcentaje = Math.Round(porcentajeSinRedondeo);
int porcentajeEntero = Convert.ToInt32(porcentaje);

String aciertosTotales = ("Aciertos totales: " + aciertos + " de " + totales);
textAciertosTotales.SetText(aciertosTotales, TextView.BufferType.Normal);
String porcAciertosTotales = ("Efectividad: " + porcentajeEntero + "%");
porcentajeAciertosTotales.SetText(porcAciertosTotales, TextView.BufferType.Normal);
```

Figura 72. Cálculo y asignación de aciertos, totales y efectividad

4.12.2. Mostrar resultados en líneas

Para los juegos dibujar y medir se han utilizado gráficas lineales para poder comparar de una forma más visual los resultados introducidos por los alumnos y los resultados correctos.

Tal y como se explica en el apartado 4.11.4, para los juegos ¿CUÁNTO MIDE? y DIBUJA se obtienen los datos introducidos por los alumnos y los datos correctos, de tal forma que se dibujen dos líneas, una para cada uno de los datos recogidos, de tal forma que se crucen ambas

líneas en todos los ejercicios donde el usuario haya acertado. Estas dos filas de datos se almacenan en sendos parámetros tipo *LineDataSet* (*dataset1* y *dataset2*), se les asignan colores distintos para diferenciarlas en la gráfica, se define un círculo para resaltar cada resultado y se asigna el grosor de la línea. Después, estos se añaden a un parámetro tipo *ILineDataSet* que se pasa como parámetro a un nuevo *LineData*, el cual recoge toda la información sobre los datos, lo que devuelven las clases *MostrarResultadosJM* y *MostrarResultadosJD* a sus respectivas *activities*, identificado como data en la figura 73.

```
for (int i = 0; i < allUserEntries.Count(); i++)
{
    userEntries.Add(new Entry(i + 1, allUserEntries[i].UserResult));
    realEntries.Add(new Entry(i + 1, allUserEntries[i].RealResult));
}
LineDataSet dataset1 = new LineDataSet(userEntries, "Tus resultados");
LineDataSet dataset2 = new LineDataSet(realEntries, "Resultados correctos");
dataset1.SetColor(Color.Rgb(255, 102, 0), 100);
dataset1.SetCircleColor(Color.Rgb(255, 102, 0));
dataset1.SetDrawCircleHole(false);
dataset1.LineWidth = 4;
dataset2.SetColor(Color.Rgb(204, 0, 153), 100);
dataset2.SetCircleColor(Color.Rgb(204, 0, 153));
dataset2.SetDrawCircleHole(false);
dataset2.LineWidth = 6;

dataSets.Add(dataset2);
dataSets.Add(dataset1);

data = new LineData(dataSets);
```

Figura 73. Creación del parámetro *data* para las gráficas lineales en *BaseDatosFunciones*

Como se ha mencionado anteriormente, las gráficas se han definido del tipo combinadas para que pueda mostrarse más de una gráfica al mismo tiempo. Es por eso que en la actividad donde se solicitan los datos a mostrar, estos se reciben como un parámetro tipo *LineData,* ya que son gráficas lineales, y deben almacenarse como una variable de datos tipo *CombinedData*, a partir de ahora se denominará como *data*. También es necesario crear una variable gráfica tipo *CombinedChart* al que pasarle los datos mencionados anteriormente y al que se le indican todos los parámetros mencionados en el apartado 4.12.1.2 sobre los ejes, leyendas y diseño de la gráfica, esta se denomina *chart* en esta aplicación.

Después de asignar a las variables *chart* y *data* todos sus parámetros, se pasa como parámetro el conjunto de datos al *chart* y se carga con la función *Invalidate* ().

Una vez que el alumno ya ha revisado sus datos, puede pulsar el botón OK para volver a la pestaña anterior donde seleccionó el juego que realizó.

4.12.3. Mostrar resultados en barras

Para mostrar los resultados del juego OPERACIONES, se ha optado por una gráfica de barras, de tal forma que se muestren en cada una de las barras los aciertos y los fallos de cada uno de los niveles realizados.

Tal y como se explica en el apartado 4.11.4, para el juego OPERACIONES, no se muestran datos recogidos como tal, sino que se generan dos parámetros por nivel, uno que recuente todas las entradas que se reciben en un nivel (*levelXtot*) y el otro que cuente solamente si el parámetro de "resultado final" para esa fila es true (*levelXOk*). Para ello se filtra en todas las líneas guardadas para el usuario y dependiendo del nivel se guarda en un parámetro o en otro. En la figura 74 se muestra cómo se calculan cuando el nivel es igual a 1, es decir, la columna *Level* en la base de datos es igual al valor 1.

```
if (allUserEntries[i].Level == 1)
{
    level1tot++;
    if (allUserEntries[i].FinalResult == true)
    {
        level1Ok++;
    }
}
```

Figura 74. Cálculo de los parámetros *levelXtot* y *levelXOk*

Para devolver un parámetro tipo *BarData* (*data*) cuando se solicitan los resultados, es necesario generar una lista tipo *List<BarEntry>* (denominado *bothResults*) donde ir guardando los resultados de cada nivel y un parámetro tipo *BarDataSet* (denominado *barSet2*) al que asignarle los datos y configurar las etiquetas de "Aciertos" y "Fallos" para poder identificar los colores al mostrar los datos. En la figura 75, se muestra cómo se añaden las entradas a la lista de entradas tipo *Bar* indicando como primer campo los aciertos y como segundo los fallos (el total de ejercicios calculados menos los aciertos), asignar toda la lista de entradas almacenadas a *barSet2* y este a su vez a la variable data que se va a devolver.

```
bothResults.Add(new BarEntry(1, new float[] { level1Ok, level1tot - level1Ok }));
bothResults.Add(new BarEntry(2, new float[] { level2Ok, level2tot - level2Ok }));
bothResults.Add(new BarEntry(3, new float[] { level3Ok, level3tot - level3Ok }));
bothResults.Add(new BarEntry(4, new float[] { level4Ok, level4tot - level4Ok }));
bothResults.Add(new BarEntry(5, new float[] { level5Ok, level5tot - level5Ok }));
bothResults.Add(new BarEntry(6, new float[] { level6Ok, level6tot - level6Ok }));

BarDataSet barSet2 = new BarDataSet(bothResults, "");
barSet2.SetStackLabels(new String[] { "Aciertos", "Fallos" });
barSet2.SetColors(new int[] { Color.Rgb(255, 102, 0), Color.Rgb(204, 0, 153) });
barSet2.SetDrawValues(false);
data = new BarData(barSet2);
```

Figura 75. Creación del parámetro *data* para las gráficas de barras en *BaseDatosFunciones*

Como se ha mencionado también en el juego anterior, las gráficas se han definido del tipo combinadas para que pueda mostrarse más de una gráfica al mismo tiempo. Es por eso por lo que en la actividad donde se solicitan los datos a mostrar, estos se reciben como un parámetro tipo *BarData,* ya que son gráficas de barras, y deben almacenarse como una variable de datos tipo *CombinedData*, a partir de ahora se denominará como *data*. También es necesario crear una variable gráfica tipo *CombinedChart* al que pasarle los datos mencionados anteriormente y al que se le indican todos los parámetros mencionados en el apartado 4.12.1.2 sobre los ejes, leyendas y diseño de la gráfica, esta se denomina *chart* en esta aplicación.

Después de asignar a las variables todos sus parámetros, se pasa como parámetro el conjunto de datos al parámetro tipo *CombinedChart* y se carga con la función *Invalidate* ().

Una vez que el alumno ya ha revisado sus datos, puede pulsar el botón OK para volver a la pestaña anterior donde seleccionó el juego que realizó.

4.13. Notificaciones por pantalla

A lo largo de toda la aplicación, aparecen ciertas notificaciones por pantalla indicando información al alumno. Estas aparecen dentro de una nube dibujada en el centro de la pantalla que está almacenada en la carpeta *Drawable* dentro de *Resources* con el nombre de "NUBE".

Para ello se ha generado un *layout* exclusivo para todos los mensajes por pantalla que va a superponerse gracias a la función *Toast* cada vez que se solicite. La función *Toast* permite

mostrar mensajes por pantalla sin congelar la aplicación y sin que el usuario pueda interactuar con él, se utiliza para mensajes informativos.

Para poder asignar al *layout* diseñado con la nube, *Custom_Toast*, al parámetro *toast*, es necesario crear un parámetro tipo *LayoutInfleter* (denominado *inflater*) y asignarlo tal y como se muestra en la figura 76.

```
View layout = inflater.Inflate(Resource.Layout.Custom_Toast,
                               (ViewGroup)FindViewById(Resource.Id.custom_toast_container));
```

Figura 76. Asignar a un parámetro tipo *view* un *layout*

Todos los mensajes por pantalla utilizados en esta aplicación son de duración larga (en torno a 3,5 segundos) ya que la duración corta disponible (2 segundos) es demasiado corta para que un niño pueda leerlo. En cada una de las funciones donde se deban mostrar mensajes por pantalla, es necesario generar el parámetro *toast* por lo que puede modificarse cualquiera de las notificaciones a una duración más corta si así se deseara. Además, como se muestra en la figura 77, también sería necesario pasar como parámetro el *layout* generado en la figura 76.

```
Toast toast = new Toast(ApplicationContext)
{
    Duration = ToastLength.Long,
    View = layout
};
```

Figura 77. Generación de un parámetro tipo *toast* y sus principales características

Y dentro de la función, cada vez que se quiera modificar el mensaje, basta con modificar un parámetro tipo *string* (en nuestro caso cadena), asignarlo al texto del layout (en nuestro caso *text*) y mostrarlo con el método *Show* () como puede apreciarse en el ejemplo de la figura 78. Para ver ejemplos de mensajes por pantalla, pueden observarse las figuras del apartado 4.6.9.1.

```
cadena = ("Lo siento, para poder comprobar debes introducir un resultado");
text.Text = cadena;
toast.Show();
```

Figura 78. Modificar y mostrar el texto de un *toast*

5. CONCLUSIONES Y LÍNEAS FUTURAS

5.1. Conclusiones

Que las nuevas TIC son utilizadas como una herramienta más en el ámbito escolar, no es nada nuevo. Es por eso, que en este proyecto se ha pretendido crear una aplicación con tres juegos que puedan servir de apoyo para los alumnos y alumnas de primaria a la hora de adquirir nuevos conceptos matemáticos y, además, sirvan de ejemplo para los profesores que se encuentran frente a estas tecnologías.

Durante todo el proyecto se han querido tener en cuenta todos los puntos mencionados en la introducción sobre la técnica denominada *Gamification*: Porcentajes de efectividad y recuento de resultados correctos por sesión, mensajes de apoyo y ánimo en cada fallo y mostrar las soluciones al equivocarse para adquirir más fácilmente los conceptos, influyen en el incremento de la motivación. A pesar de que se han utilizado estas técnicas, existen otras que pueden potenciar la motivación como son el conseguir puntos con cada acierto, premios por acumular un número elevado de respuestas correctas seguidas o desbloquear niveles con los puntos conseguidos. También, si el juego se va a desarrollar en la misma Tablet para todos los alumnos, pueden diseñarse avatares personalizados por cada usuario y mostrar tablas con los resultados puntuando mejor a los que realicen más ejercicios o al que mejor haya progresado de la semana. Existen infinidad de posibilidades para aumentar la motivación en *Gamification*.

En cuanto a los juegos que se han diseñado relacionados con segmentos, no se ha encontrado ninguna aplicación disponible similar en las tiendas de aplicaciones por lo que ofrecen una solución al estudio del concepto de longitud en alumnos y alumnas de primaria que no se había implementado hasta ahora. Bien es cierto que queda limitada por el tamaño del dispositivo, es decir, cuanto mayor sea la Tablet, más variedad de distancias puede adivinar un usuario, pero si es de tamaño inferior al recomendado, las posibilidades de adquirir el concepto de forma correcta son pequeñas.

En cuanto al juego relacionado con operaciones numéricas, sí existen varias APP similares disponibles en las tiendas de aplicaciones, pero en ninguno se superan los niveles si no se responde correctamente en los ejercicios y en la mayoría solo hay que rellenar únicamente el resultado. Además, muy pocos muestran un resumen de los aciertos y fallos obtenidos. Como mejora, podría realizarse una opción de ajustes donde el profesor pudiera seleccionar el número máximo de valores para los operandos, seleccionar la cantidad de niveles que quisiera mostrar o elegir los operadores de tal forma que se adapte a las necesidades de los alumnos y alumnas.

Sin duda, este proyecto no sería totalmente útil si no se diseñaran más juegos y estos, a su vez, se probasen con niños. Por lo que es imprescindible que los juegos diseñados sean utilizados por algún profesor de alguna escuela y así saber si se adaptan a las necesidades de los alumnos o si cumplen con la función por la que se han diseñado. Además, tras utilizarlos, pueden aportar nuevas ideas para poder adaptar los juegos a ellos o indicar necesidades para implementar otros ejercicios que les ayuden durante su educación en la etapa escolar, principal motivo por el que se ha elaborado este proyecto.

Para finalizar, indicar que la realización de este proyecto me ha ayudado a mejorar mis habilidades de búsqueda de información y conocimientos de programación ya que desconocía la programación C# y Android, creatividad a la hora de pensar en los diseños de la aplicación, superación cuando uno de los juegos no funcionaba como deseaba, precisión a la hora de encontrar los fallos en el diseño y, sobre todo, motivación de saber que el proyecto va a aportar un granito de arena a un proyecto mayor dedicado a la mejora en la educación de las nuevas y futuras generaciones.

5.2. Líneas futuras

A continuación, se muestra un listado con las posibles líneas futuras, algunas comentadas en las conclusiones anteriores:

- La primera línea futura y principal es testear la aplicación con niños y niñas de educación primaria para detectar mejoras y que la aplicación se adapte a ellos.
- La segunda línea futura y también principal es verificar que el uso de esta aplicación ayude a los niños a adquirir los conceptos por los que se han diseñado. Podrían realizarse pruebas antes, durante y después de su utilización o revisar mediante la base de datos si los alumnos van obteniendo mejores resultados a medida que utilizan la aplicación.
- Para el juego OPERACIONES, implementar un botón de ajustes con el que poder definir los operadores, valores mínimos y máximos y en número de ejercicios en el juego de operaciones para que pueda ajustarse a las necesidades de cada usuario.
- Para el juego DIBUJA, modificar la configuración para que la línea dibujada por el jugador se transforme en una línea recta de inicio a fin y así se aprecie mejor la línea dibujada y sea más sencillo para el usuario.
- De cara a la motivación de los alumnos, diseñar avatares para cada usuario y que aparezcan en una tabla tipo “mejores de la semana”, conseguir puntos con cada acierto, premios por acumular un número seguido de respuestas correctas o por realizar más ejercicios que los demás, y la posibilidad de desbloquear niveles con los puntos obtenidos.
- Un posible ejercicio nuevo sería que el alumno relacione objetos de distintos tamaños (con valores coherentes en centímetros, metros o incluso kilómetros) y adivine el tamaño de uno de ellos disponiendo otro como referencia, por ejemplo, la longitud entre dos frutas en centímetros, entre dos personas en decímetros o entre dos edificios en metros sin necesidad de que sea a tamaño real en el dispositivo como el juego diseñado en esta APP.
- Otra posible actividad sería mostrar por pantalla distintas imágenes donde apareciera un objeto en particular con una cifra de distancia y que tuviera que introducir las unidades. Por ejemplo, aparece el dibujo de una persona y a su lado 1.6, pues el alumno debería indicar que la medida está en metros.
- Buscar nuevas necesidades para los alumnos y alumnas y diseñar más juegos que les ayuden a adquirir otros conceptos como son porcentajes, fracciones, figuras geométricas o relaciones trigonométricas.

6. REFERENCIAS BIBLIOGRÁFICAS

[1] Tintoré, R. "Implementación de las TICs en la Educación: Cómo y Porqué", *GoConqr* , 27 septiembre 2017. Página Web: https://www.goconqr.com/es/blog/implementacion-de-las-tics-en-la-educacion-como-y-porque/

[2] De Miguel García, ML. "Las TICs aplicadas a las Necesidades Educativas Especiales", *PFG*, *Escuela de Educación de Soria*, 2 Julio 2014.

[3] Fernández Álvarez, A. "Ventajas y riesgos de las TIC en educación", *Nubecina*. Página Web: https://www.nubemia.com/ventajas-y-riesgos-de-las-tic-en-educacion/.

[4] Aula Planeta. "Redes wifi, proyectores y pizarras electrónicas se habrán generalizado en las aulas españolas en los próximos dos años", 2015. Enlace de descarga: http://www.aulaplaneta.com/wp-content/uploads/2015/04/NdP_14abr_Redes-wifi-proyectores-y-pizarras-electr%C3%B3nicas-se-habr%C3%A1n-generalizado-en-las-aulas-espa%C3%B1olas-en-los-pr%C3%B3ximos-dos-a%C3%B1os.pdf

[5] Jahnke, I.; Norqvist, L.; Olsson, A. "Digital Didactical Designs in iPad-Classrooms", *EC-TEL*, 2013: 611-612

[6] Li, S.C.; Pow, J.; Wong, E. M. L.; Fung, A.C.W. "Empowering Students Learning through Tablet PCs: A Case Study." *Education and Information Technologies*, 2010, 15(3): 171-180.

[7] Geist, E.A. "A qualitative examination of two year-olds interaction with tablet based interactive technology", *Journal Instructional Psychology*, 2012, 39(1): 26-35

[8] Clark, R.E. "Media Will never influence learning", *Educational Technology Research & Development*, 1994, 42(2): 21-29.

[9] Sailer M., Hense J., Mandl H. and Klevers M. "Psychological perspectives on motivation through Gamification", *Interaction Design and Arquitecture(s) Journal,* vol. 19 pp. 28-37, 2013.

[10] Passolunghi, M.C.; Vercelloni, B.; Schadee, H. "The precursors of mathematics learning: working memory, phonological ability and numerical competence", *Cognitive Development*, 2007, 22(2): 165-184.

[11] Miyake, A.; Friedman, N.P.; Emerson, M.J.; Witzki, A.H.; Howerter, A.; Wager, T.D. "The unity and diversity of executive functions and their contributions to complex 'frontal lobe' tasks: a latent variable analysis" *Cognitive Psychology*, 2000, 41(1): 49-100.

[12] Jordan, N.C.; Kaplan, D.; Olah, L. N.; Locuniak, M. N. "Number sense growth in kindergarten: A longitudinal investigation of children at risk for mathematics difficulties", *Child Development,* 2006, 77: 153-175

[13] Geary, D.C. "Cognitive predictors of achievement growth in mathematics: a five year longitudinal study", *Developmental Psychology*, 2012, 47(6): 1539-1552.

[14] Carraher, T.N.; Carraher, D.W.; Schliemann, A.D. "Mathematics in the streets and in schools", *British Journal of Developmental Psychology*, 1985, 3(1): 21-29.

[15] Resnick, L.B. "From protoquantities to operators: building mathematical competence on a foundation of everyday knowledge", *Analysis of Arithmetic for Mathematics Teaching*, 1992, 19: 275-323.

[16] Visual Studio Code. "IntelliSense", 10 Abril 2018. Página web: https://code.visualstudio.com/docs/editor/intellisense

[17] Microsoft. Visual Studio. "Edición del código: Escriba código, navegue y corrija problemas". Página web: https://visualstudio.microsoft.com/es/vs/features/ide/

[18] Microsoft. Developer Network. "Utilizar IntelliSense", 2015. Página web: https://msdn.microsoft.com/es-es/library/hcw1s69b.aspx

[19] Microsoft. Documentos de Visual Studio. "Hacer que el código funcione en Visual Studio", 2 Mayo 2018. Página web: https://docs.microsoft.com/es-es/visualstudio/ide/find-and-fix-code-errors?view=vs-2017

[20] Microsoft. Docs. "Requisitos del sistema de la familia de productos de Visual Studio 2017", 7 Mayo 2018. Página web: https://docs.microsoft.com/es-es/visualstudio/productinfo/vs2017-system-requirements-vs

[21] Microsoft. "Introducción al lenguaje C# y .NET Framework", Visual Studio, 2013. Página web: https://msdn.microsoft.com/es-es/library/z1zx9t92(v=vs.120)

[22] System.OutOfMemoryException. "Un poquito de historia de C#", *Un blog sobre .NET y sus curiosidades*, 18 Mayo 2015. Página web: https://sparraguerra.wordpress.com/2015/05/18/un-poquito-de-historia-de-c/

[23] ECMA-334 Standard. "C# Language Specification", 5ª Edición, *ECMA International*, Diciembre 2017. Enlace de descarga: http://www.ecma-international.org/publications/files/ ECMA-ST/Ecma-334.pdf

[24] López, A. "Desarrollo de Apps Cross-Platform con Xamarin.Forms", *SG Software Guru*. Página web: https://sg.com.mx/revista/47/desarrollo-apps-cross-platform-xamarinforms

[25] Microsoft. "Visual Studio y Xamarin", Documentos de Visual Studio, 30 Marzo 2018. Página web: https://docs.microsoft.com/es-es/visualstudio/cross-platform/visual-studio-and-xamarin

ANEXO I: MANUAL DE INSTALACIÓN Y USO DE LA APLICACIÓN

El anexo I se compone de un manual de uso para que la persona que vaya a utilizarla entienda por completo las posibilidades de la aplicación y su correcta utilización. Incluye primeramente una guía de instalación, seguido de una explicación genérica y específica de cada uno de los juegos.

I.1 REQUISITOS DEL DISPOSITIVO

Para poder ejecutar la aplicación, tanto en un emulador como en un Smartphone físico, es necesario que se tengan como mínimo las siguientes características:

- Espacio de memoria de 35 Mb para la aplicación.
- Memoria RAM de al menos 2 Gb.
- Versión mínima del sistema operativo Android: 5.0 Lollipop.
- Es recomendable que la aplicación se ejecute en una Tablet de mínimo 11" para que los juegos de segmentos dispongan de un número razonable de valores posibles.

Como la aplicación de ejemplo no se encuentra en el Play Store, existen dos maneras de copiar la aplicación al teléfono:

- Compilándola mediante Visual Studio. Una vez se ha ejecutado este proceso, la aplicación se mantendrá en el dispositivo hasta que el usuario la borre o hasta que se compile una nueva versión de la misma.
- Se podría crear el fichero ejecutable (*apk*) a través de las herramientas que proporciona Visual Studio, pero es un proceso complejo que no merece la pena, ya que el usuario que reciba la aplicación tendría que activarse las opciones de desarrollador.

I.1.1 Compilar la aplicación mediante Visual Studio en un dispositivo físico

Para poder copiar la aplicación al dispositivo hay que realizar los siguientes pasos:

1. Abrir el proyecto completo en Visual Studio. Se puede descargar la versión gratuita Community desde su página web[12] si no se dispone del programa.
2. Habilitar las opciones de desarrollador en el dispositivo dónde se vaya a copiar la aplicación. Para ello, el usuario debe pulsar 5 veces seguidas encima del número de compilación que se encuentra en ajustes del teléfono/acerca del dispositivo. Una vez habilitado, activar las opciones del desarrollador y la depuración USB.
3. Descargar el plugin desde el Android SDK Manager para que Visual Studio permita probar la aplicación en un Smartphone físico. Este puede descargarse desde Herramientas, Android, Android SDK Manager. El paquete que se necesita instalar es el llamado Google USB Driver, que se encuentra debajo de la pestaña Extras.
4. Conectar el dispositivo a través del USB del ordenador y aparecerá, en Visual Studio, la opción para probar la aplicación en el mismo. Una vez se compile el proyecto y se ejecute, el Smartphone ejecutará la aplicación y esta quedará copiada en el mismo como una aplicación más.

[12] https://www.visualstudio.com/es/free-developer-offers/

I.2 USO DE LA APLICACIÓN

Una vez instalada la aplicación puede comenzar a usarla.

Nada más abrir la aplicación, la APP solicita el nombre del alumno que vaya a realizar las actividades tal y como se muestra en la figura 79.

Figura 79. Primera pantalla APP.

El nombre debe ser introducido donde indica “Escribe tu nombre”. Por ejemplo, si el usuario se llama “Raquel” quedaría como se muestra en la figura 80.

Figura 80. Primera pantalla APP con nombre.

Cualquier alumno puede volver a esta pantalla y modificar su nombre si así lo desea o necesita. Una vez introducido, el usuario debe pulsar el botón JUGAR para poder continuar a la elección de su juego.

La primera elección que el usuario debe escoger es la que se muestra en la figura 81: SEGMENTOS u OPERACIONES.

Figura 81. Pantalla de selección entre SEGMENTOS u OPERACIONES

Si el usuario pulsa encima de la opción OPERACIONES podrá jugar al juego que se explica en el apartado I.2.1 Juego OPERACIONES. Si, por el contrario, el usuario pulsa en la opción SEGMENTOS, tendrá que elegir entre uno de los dos juegos disponibles en esta sección tal y como se muestra en la figura 82: ¿CUÁNTO MIDE? o DIBUJA.

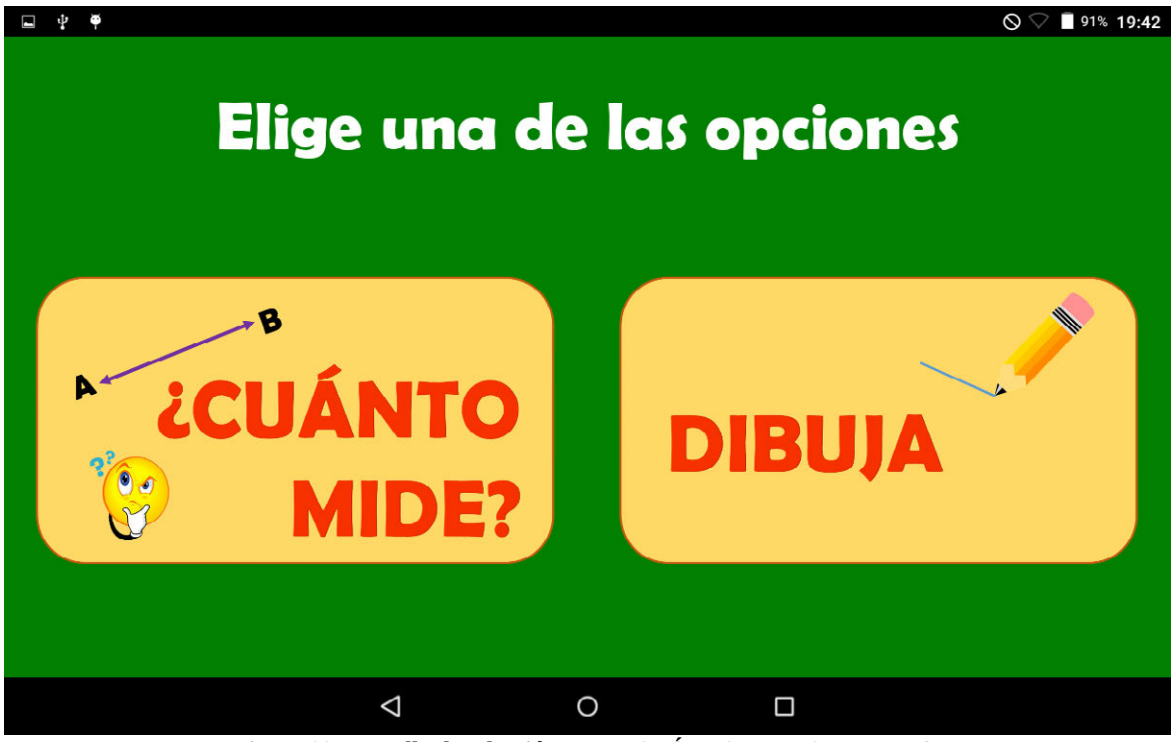

Figura 82. Pantalla de selección entre ¿CUÁNTO MIDE? o DIBUJA.

Si el usuario escoge la opción ¿CUÁNTO MIDE?, podrá jugar al juego que se explica en el apartado I.2.2 Juego ¿CUÁNTO MIDE? Si por el contrario elige la opción DIBUJA el usuario podrá jugar al juego que se explica en el apartado I.2.3 Juego DIBUJAR.

I.2.1 Juego OPERACIONES

Si el usuario elige realizar la actividad OPERACIONES, este debe ir rellenando los recuadros que aparezcan vacíos en los 6 niveles de los que dispone la aplicación. En cada nivel hay que completar uno o dos recuadros (operando 1, operando 2, operador o resultado), por ejemplo, en el nivel 1 el usuario debe introducir un resultado para la operación que aparezca (ver figura 83). Pulsando dentro del recuadro, aparecerá un teclado con todos los números o caracteres y el alumno debe seleccionar el número u operador correspondientes para completar la ecuación de forma correcta.

Figura 83. Pantalla juego OPERACIONES nivel 1.

Si el usuario debe introducir el operando, debe elegir una de las 4 opciones disponibles:

- "+" para sumar.
- "-" para restar.
- "*" para multiplicar.
- "/" para dividir.

Si por equivocación selecciona cualquier otro carácter, un mensaje aparecerá por pantalla indicándole las opciones posibles.

Una vez el usuario crea que su solución es correcta, debe pulsar en COMPROBAR donde, acto seguido, aparecerá por pantalla un mensaje indicando si ha acertado o si ha fallado y pasará a la siguiente ecuación. Así hasta que complete 3 ecuaciones por cada uno de los niveles.

Una vez finalizados todos los ejercicios, aparecerán los resultados de todos los niveles en forma de gráfica de barras como en la figura 84. Los aciertos se representan en color naranja y los fallos en color morado (se indica esta información en la parte inferior de la gráfica) y puede alejarse y acercarse pulsando con dos dedos encima de la gráfica, separándolos y juntándolos.

Además, justo debajo de la gráfica, aparecerá un recuento del total de aciertos y el porcentaje de efectividad durante esa sesión.

Es importante resaltar que los datos que aparecen de la gráfica son de la última sesión, de tal forma que, si el usuario decide salir a mitad del ejercicio y vuelve a intentarlo, solamente aparecerá la información de esta última sesión, no del total de ejercicios que haya realizado durante el uso de la APP.

Figura 84. Resultados juego OPERACIONES. Gráfica de barras.

Para finalizar la sesión, el usuario debe pulsar en el botón OK situado en la parte de abajo a la derecha y volverá a la pantalla representada en la figura 81.

Si el usuario decide salir antes de finalizar todos los ejercicios, debe pulsar en el botón SALIR y aparecerán los resultados que haya realizado hasta ese momento en esa sesión.

I.2.2 Juego ¿CUÁNTO MIDE?

Si el usuario elige realizar la actividad ¿CUÁNTO MIDE?, este debe introducir en el recuadro de la derecha los centímetros que crea que mide el segmento que se muestra en el recuadro de la izquierda, tal y como aparece en la figura 85.

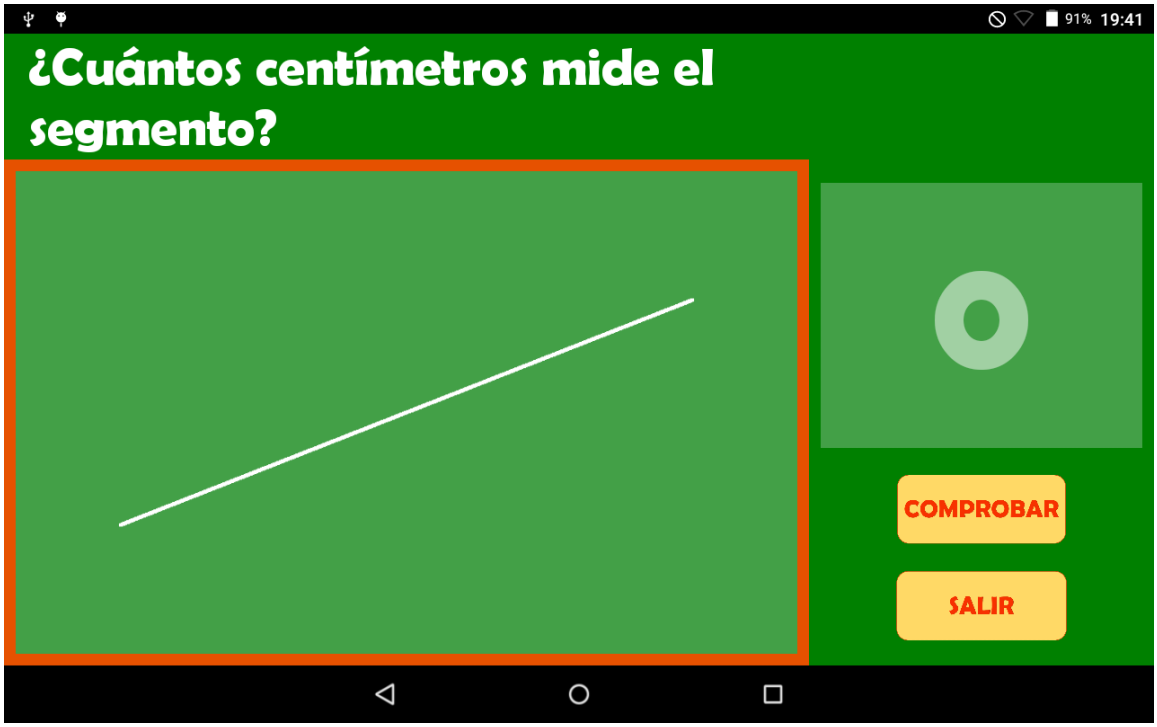

Figura 85. Pantalla juego ¿CUÁNTO MIDE?

Una vez el usuario crea que su solución es correcta, debe pulsar en COMPROBAR y aparecerá por pantalla un mensaje indicando si ha acertado o, en el caso de haber fallado, los centímetros que debería de haber introducido. Acto seguido, pasará a un nuevo ejercicio y así hasta que decida salir del juego donde deberá pulsar en el botón SALIR.

Una vez el usuario decida salir, aparecerán los resultados de todos los intentos realizados en esa sesión en forma de gráfica lineal como se muestra en la figura 86. Los resultados correctos que debería de haber introducido el usuario se representan en morado y los resultados introducidos por el alumno se representan con una línea naranja. El usuario puede alejarse y acercarse en la gráfica pulsando con dos dedos encima de la gráfica, separándolos y juntándolos. Además, debajo de la gráfica, aparecerá un recuento del total de aciertos y el porcentaje de efectividad de esa última sesión.

Para finalizar el juego, el usuario debe pulsar en el botón OK situado en la parte de abajo a la derecha y volverá a la pantalla representada en la figura 82.

Figura 86. Resultados juego SEGMENTOS. Gráfica lineal.

I.2.3 Juego DIBUJAR

Si el usuario elige realizar la actividad DIBUJAR, este debe dibujar en el recuadro de la izquierda una línea que mida los centímetros que le indique el enunciado. En la figura 87 aparece un ejemplo de lo que el usuario se encontrará al iniciar el juego donde, en este caso, debe dibujar una línea de 7 centímetros.

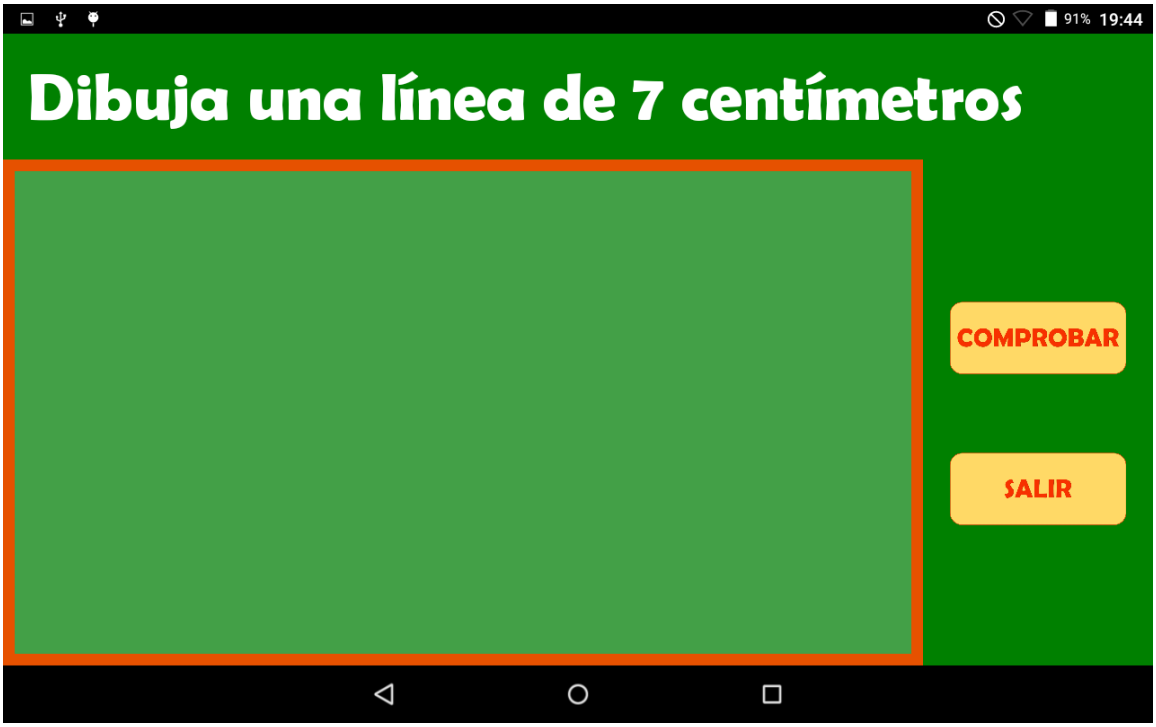

Figura 87. Pantalla juego DIBUJAR

Simplemente pulsando en el recuadro y arrastrando el dedo por la pantalla, el alumno puede dibujar la línea que decida. Si el usuario cree que la línea que ha dibujado no se corresponde con la distancia indicada, puede volver a dibujar en el lienzo y la anterior línea se borrará automáticamente.

Una vez el usuario crea que su solución es correcta, debe pulsar en COMPROBAR y aparecerá por pantalla un mensaje indicando si ha acertado o, en el caso de haber fallado, los centímetros de la línea que ha dibujado. Acto seguido, aparecerá un nuevo ejercicio, así hasta que decida salir del juego donde deberá pulsar en el botón SALIR.

Una vez el usuario decida salir, aparecerán los resultados de todos los intentos realizados en esa sesión en forma de gráfica lineal como se muestra en la figura 86 de la sección anterior. Los centímetros que debería de haber dibujado el usuario se representan en morado y los centímetros dibujados por el alumno se representan mediante una línea naranja. El usuario puede alejarse y acercarse en la gráfica pulsando con dos dedos encima de la gráfica, separándolos y juntándolos. Además, debajo de la gráfica, aparecerá un recuento del total de aciertos y el porcentaje de efectividad de la última sesión.

Para finalizar el juego, el usuario debe pulsar en el botón OK situado en la parte de abajo a la derecha y volverá a la pantalla representada en la figura 82.

ANEXO II: PRESUPUESTO

La tabla siguiente contiene los recursos que han sido necesarios para la realización de este proyecto, y en la tabla 6 el precio total del proyecto.

Tabla 5. Recursos necesarios para la realización del proyecto

	NOMBRE	PRECIO (€/mes)	UNIDADES	TOTAL (€)
MANO DE OBRA	Graduado en Ingeniería	1430,00 €	8	11440,00 €
SOFTWARE	Visual Studio Community	0,00 €	1	0,00 €
	Office X	69,00 €	1	69,00 €
HARDWARE	Ordenador del laboratorio	700,00 €	1	700,00 €
	Monitor del laboratorio	125,00 €	1	125,00 €
	Ratón y teclado del laboratorio	17,00 €	1	17,00 €
	Tablet	200,00 €	1	200,00 €
	Internet	45,00 €	8	360,00 €

Tabla 6. Presupuesto total del proyecto

CONCEPTO	COSTE (€)
MANO DE OBRA	11440,00 €
SOFTWARE	69,00 €
HARDWARE	1402,00€
Total sin IVA	12911,00€
IVA (21 %)	2711,31€
TOTAL	15622,31€