UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

App Android para la programación de test

Estudiante: Luis Pérez Muñoz
Tutor: Eduardo Boemo Scalvinoni
Fecha: Julio 2020

App Android para la programación de test

AUTOR: Luis Pérez Muñoz
TUTOR: Eduardo Boemo Scalvinoni

Digital System Laboratory
Dpto. TEC
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2020

Resumen (castellano)

Este Trabajo de Fin de Grado consiste en la realización de una App de ejercicios de la guía de problemas de Design for Test de la asignatura DIE/DEPSE de la EPS UAM.

El objetivo de esta aplicación es favorecer el aprendizaje de la detección de fallos Stuck -At por parte de los alumnos del Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación.

La App desarrollada consta de las siguientes especificaciones:

- Tiene una serie de ejercicios incluidos en la App.
- Funciona tanto con conexión a Internet, como sin ella; ya que hace uso de una base de datos interna en el móvil. En dicha base de datos se guardan los ejercicios cargados en la aplicación, ya sean los que incluye por defecto la aplicación como los problemas de Stuck-At creados por el alumno dentro de la aplicación.
- Permite quitar problemas de la lista de ejercicios incluidos en la aplicación.
- Contiene una papelera de reciclaje donde se encuentran los problemas quitados de la lista de ejercicios incluidos en la aplicación.
- Permite la recuperación de los problemas encontrados en la papelera de reciclaje.
- Permite la eliminación de los problemas de Stuck-at encontrados en la papelera de reciclaje, para de esta forma evitar que la aplicación en el smartphone con Android del alumno ocupe demasiado espacio.
- Permite la creación de nuevos problemas por parte del alumno; dichos problemas deben estar formados por las puertas lógicas básicas nand, and, or, xor, inv; las puertas deben de tener como máximo 3 entradas.
- Incluye una ayuda para facilitar el manejo de la aplicación por parte del alumno. Dicha ayuda se ha dividido en (Abrir problema, Completar tabla, Comprobar la solución, Ver esquema, Resetear problema, Eliminar un problema, Inventar un problema personalizado, Resetear Estadísticas) para que de esta forma sea fácil de usar y rápida para el alumno.
- Hace uso de los recursos de Android para facilitar la traducción de esta a otros idiomas en un futuro; aunque en un principio está diseñada en Inglés para que sea accesible a más personas.
- Está estructurada internamente para facilitar en un futuro la incorporación de nuevos problemas en nuevas actualizaciones publicadas en Google Play.

Abstract (English)

This Bachelor Thesis consists in the realization of an App of exercises of the Design for Test problem guide of the subject DIE / DEPSE of the EPS UAM.

The objective of this application is to favor the learning of the Stuck-At fault detection by the students of the Telecommunication Technologies and Services Engineering Degree.

The developed App consists of the following specifications:

- Has a series of exercises included in the code of the App.
- Works with Internet connection, and without it, since it makes use of an internal database in the mobile. In this database, the exercises loaded in the application are saved, and also the Stuck-At problems created by the student in the application.
- Allows the elimination of Stuck-at problems loaded in the application.
- Contains a recycling bin where the problems removed from the list of exercises included in the application are found.
- Allows the recovery of problems found in the recycling bin.
- Allows the elimination of Stuck-at problems found in the recycling bin, to prevent the application on the student's An-droid smartphone consumes too much space.
- Allows the creation of new problems by the student; these problems must be formed by the basic logic gates nand, and, or, xor, inv; the gates must have a maximum of 3 inputs.
- Includes a help to facilitate the handling of the application by the student. This help has been divided into (Open problem, Complete table, Check solution, View scheme, Reset problem, Delete problem, Invent a custom problem, Reset statistics) so that it is easy to use and quick for the student.
- Uses the resources of Android to facilitate the translation of it to other languages in the future; although initially it is designed in English to make it accessible to more people.
- Is structured internally to facilitate in the future the incorporation of new problems in new updates published in Google Play.

Palabras clave (castellano)

Smartphone, Android, App, Stuck-at, DIE/DEPSE, puerta lógica, papelera de reciclaje.

Keywords (inglés)

Smartphone, Android, App, Stuck-at, DIE/DEPSE, logic gateway, recycle bin.

Agradecimientos

Primero quiero agradecer a mi tutor Eduardo Boemo Scalvinoni el permitirme realizar este trabajo y ayudarme en su realización resolviendo todas las preguntas que le he formulado.

Segundo darle las gracias a los profesores que he tenido durante la carrera, que me han permitido tener las herramientas necesarias para realizar este trabajo. En especial a Alejandro Sierra Urrecho por impartirme la asignatura de Desarrollo de Aplicaciones Para Dispositivos Móviles; gracias a los conocimientos adquiridos en esa asignatura me ha sido mucho más fácil realizar este TFG.

Tercero le doy las gracias a Pablo, Alberto, Jorge, José, Lucas y Alejandro, que desde que tuvimos la suerte de conocernos en la universidad no nos hemos separado, hemos ido superando juntos todos los retos que nos han ido apareciendo y hemos disfrutado de momentos inolvidables y los que quedan por disfrutar.

Por ultimo y no menos importante le doy las gracias también a mis padres, a mi hermano, a mi novia y a toda mi familia por el apoyo que me han dado, tratando de que no me desanimara y preocupándose por cómo iba en la universidad. Sin su apoyo no hubiera sido posible llegar hasta aquí.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1. Introducción

1.1 Motivación

Este TFG surge con la intención de aprovechar el amplio uso de los smartphones y las apps, para ofrecer a los estudiantes de ingeniería una manera alternativa de aprendizaje práctico de la asignatura DIE/DEPSE de la EPS UAM.

Para ello se desarrollará una aplicación que permita ofrecer a los alumnos ventajas y funcionalidades que no son posibles con el formato tradicional de guías de problemas. Entre ellas cabría destacar:

- Disponible gratis y sin publicidad en Google Play.
- Autocorrección de problemas stuck-at.
- Reducción de uso del papel.
- Solución de problemas planteados por el alumno.
- Control de tiempo empleado para resolver el problema.

1.2 Objetivos

El objetivo del TFG es diseñar una aplicación Android gratuita y sin anuncios, que permita ejercitar problemas de test de circuitos combinacionales sencillos, utilizando la técnica de stuck-at de la manera más fácil posible.

Los problemas que contendrá la aplicación pertenecen a la guía de problemas del 4º tema de la asignatura Dispositivos Integrados Especializados (DIE), del 3º curso del Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid (EPS-UAM).

Para hacer más fácil el uso de la aplicación se ofrecerá una ayuda para el manejo de la misma.

Desde mi punto de vista el objetivo es hacer uso de los recursos de Android para que la aplicación sea lo más eficiente y mantenible posible.

También aprender el concepto de stuck-at y sus implicaciones en el funcionamiento de los circuitos, llevando el concepto a su lado más práctico, permitiendo detectar problemas del tipo stuck-at de la manera más rápida posible.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1: Introducción:** Explica la motivación y objetivos del proyecto, así como la organización de la memoria.
- **Capítulo 2: Estado del arte:** Describe la situación actual del mercado de aplicaciones de test en Play Store.
- **Capítulo 3: Stuck-at**: Describe el problema que se puede dar en los circuitos electrónicos y en que consiste el problema de Stuck-at.
- **Capítulo 4: Desarrollo de la App y pruebas:** Precisa como se ha desarrollado el programa y analiza las pruebas realizadas.
- **Capítulo 5**: **Conclusiones y trabajo futuro:** Sintetiza las conclusiones que se han obtenido tras la realización del proyecto y propone posibles mejoras.

2. Estado del arte

En este capítulo se expondrá un breve estudio de las distintas aplicaciones de prueba disponibles en la Google Play, así como de las aplicaciones desarrolladas por DSLab UAM.

2.1 Aplicaciones similares

Antes de desarrollar cualquier aplicación, resulta conveniente buscar aplicaciones similares ya existentes en la Google Play Store. Afortunadamente se ha encontrado una aplicación titulada Stuck-at que se centra en la comprobación de errores en circuitos lógicos mediante la técnica de stuck-at; dicha aplicación ha servido de base para el desarrollo de la aplicación de este TFG.

A continuación, una breve descripción de donde encontrar Stuck-at y para que funciona.

URL: https://play.google.com/store/apps/details?Id=com.CIRDIG
Descripción: Es una aplicación educativa que ofrece una serie de ejercicios y un test sobre el modelo de detección de fallos stuck-at. Se incluye un tutorial sobre el modelo stuck-at y un menú de ayuda para el manejo de la aplicación.

La aplicación ha sido desarrollada por DSLab UAM como PFC de la EPS UAM [13]; por ello nos fijaremos en las aplicaciones desarrolladas por otros alumnos a través de DSLab UAM.

También analizaremos otras aplicaciones de tipo educativo existentes en la Google Play Store.

2.1.1 Aplicaciones DSLab UAM

Existen hasta 11 aplicaciones desarrolladas por DSLab UAM como proyecto de fin de carrera o trabajo de fin de grado realizado por otros compañeros de la UAM otros años.
De entre las 11, solamente 7 ofrecen una funcionalidad parecida a la que se desarrollará en este TFG, centrándose en otros temas de la universidad.

A continuación, una tabla con el nombre, el estado, la valoración media, un breve resumen y la URL que permite encontrar la memoria de cada una de las 7 aplicaciones.

N.º	Nombre	Estado	Valoración media	Breve Resumen	URL TFG/PFC
1	Combinational Circuits	Publicada y finalizada	4,2 ☆	Ofrece una serie de ejercicios de electrónica combinacional.	Https://reposito-rio.uam.es/handle/10486/660545
2	Sequential Circuits	Publicada y finalizada	4,3 ☆	Ofrece una serie de ejercicios de máquinas de estados.	Https://reposito-rio.uam.es/handle/10486/660543
3	MOS Circuits	Publicada y finalizada	4,7 ☆	Ofrece una serie de ejercicios de circuitos integrados MOS.	Https://reposito-rio.uam.es/handle/10486/663772
4	Arithmetic Circuits	Publicada y finalizada	Sin valoraciones	Ofrece una serie de ejercicios sobre circuitos aritméticos electrónicos.	Https://reposito-rio.uam.es/handle/10486/677847
5	Luts	Publicada y finalizada	Sin valoraciones	Ofrece una colección de ejercicios sobre Circuitos Electrónicos Digitales.	Https://reposito-rio.uam.es/handle/10486/679339
6	Stuck-at	Publicada y finalizada	Sin valoraciones	Ofrece una serie de ejercicios para practicar el modelo de detección de fallos de stuck-at.	Https://reposito-rio.uam.es/handle/10486/669462
7	Counters	Publicada y finalizada	3,8 ☆	Ofrece unos 14 ejercicios sobre contadores de diferentes modalidades.	Https://reposito-rio.uam.es/handle/10486/669567

Tabla 2.1.1-1: Aplicaciones anteriores de DSLab UAM.

2.1.2 Otras aplicaciones de circuitos digitales en Google Play

Existen muchas aplicaciones en la tienda de Google que se encuentran en el ámbito de la enseñanza, pero esta vez analizaremos aquellas que aparecen tras buscar circuitos digitales en la tienda. (No son de las Aplicaciones y aportan una utilidad.)

Se han seleccionado para su estudio aquellas que tienen una mejor valoración (del 4 al 5) a fecha 06/06/2020. De todas ellas se aportará la URL y su valoración.

2.1.2.1 Solución para mapas Karnaugh

Tiene una interfaz muy funcional pero no incluye una pequeña ayuda de cómo se usa, a diferencia de Electrônica Digital que si la incluye. Permite hallar la solución a los mapas de Karnaugh. No incluye una publicidad muy invasiva.

URL: https://play.google.com/store/apps/details?Id=karnagh.ammsoft.karnagh
Valoración: 4,6☆

2.1.2.2 Dicide: Circuitos digitales

Permite crear circuitos combinacionales a partir de su tabla de verdad, y crear circuitos secuenciales a través de máquinas de estados.

URL: https://play.google.com/store/apps/details?Id=com.avh.digitalcircuitdesign
Valoración: 4,5☆

2.1.2.3 Digital Circuits

Permite aprender la teoría de los circuitos digitales. Desde qué es un circuito digital hasta los dispositivos de memoria, pasando por las diferentes puertas lógicas.

URL: https://play.google.com/store/apps/details?Id=in.softecks.digitalcircuits
Valoración: 4,6☆

2.1.2.4 Electrônica Digital

Está en portugués. Permite simplificar sentencias booleanas mediante mapas de Karnaugh de manera automática, averiguar cuál es la representación en hexadecimal y binario de un código ASCII y convertir números de una base (decimal, binario, hexadecimal, octal) a otra (decimal, binario, hexadecimal, octal).

URL: https://play.google.com/store/apps/details?Id=com.eletronica.android.ferramentas
Valoración: 4,5☆

2.1.2.5 Calculadoras de Ingeniería Electrónica

Tiene un diseño muy actual. Separa las herramientas disponibles de manera Offline de las disponibles a través de una conexión a internet. Permite realizar cálculos teóricos de una manera rápida y sencilla y además permite repasar la teoría relacionada con los cálculos que se están realizando.

URL: **https://play.google.com/store/apps/details?Id=com.schiller.herbert.calcparaeletronicafree**
Valoración: 4,7☆

3. Stuck-At

En esta sección se explicará que es Stuck-At y se incluirá un ejemplo práctico para que se entienda mejor.

3.1 Explicación de stuck-at

Stuck-at es uno de los problemas que se pueden dar en los circuitos digitales. Este problema se produce cuando una parte del circuito digital se encuentra siempre en el mismo voltaje y no varía sea cual sea la entrada del circuito.

El conocimiento de que existe este tipo de problema nos permite crear técnicas para tratar de detectarlo con la mayor rapidez y sencillez. Estas técnicas pueden ser usadas cuando se crean circuitos digitales y de esta forma asegurarse de que el circuito digital creado no tiene este tipo de problema.

Una de las técnicas existentes es el uso de tablas de verdad. Estas tablas se crean con tantas filas como posibles entradas tiene el circuito y con tantas columnas como posibles problemas se desean detectar.

A continuación, se utilizará la técnica de tablas de verdad en un circuito de ejemplo para que se entienda mejor el uso que se le puede dar a esta técnica.

3.1.1 Ejemplo práctico de Stuck-at

El circuito sobre el cual se utilizará la técnica de tablas de verdad está representado en la siguiente figura:

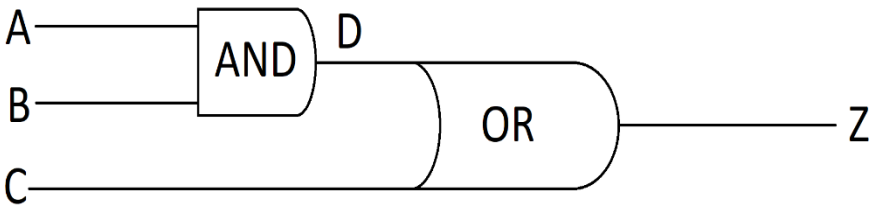

Figura 3.1.1-1: Esquema del circuito de ejemplo

Fuente: Propia

La tabla de verdad que debería cumplir este circuito [Figura 3.1.1-1] es la siguiente:

Entradas			Salida
C	B	A	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Tabla 3.1.1-1: Tabla de verdad ideal del circuito de ejemplo.

Pero obtenemos que la salida real del circuito está representada por la siguiente tabla de verdad:

Entradas			Salida
C	B	A	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Tabla 3.1.1-2: Tabla de verdad real del circuito de ejemplo.

La fila marcada en rojo es la salida en la que falla el circuito ya que no cumple con la salida ideal del mismo.

Si tratamos de detectar el fallo mediante tablas de verdad usando la técnica de Stuck-at obtendremos la siguiente tabla:

Entradas			**Salida**	**Posibles fallos**									
C	B	A	Z	A/0	A/1	B/0	B/1	C/0	C/1	D/0	D/1	Z/0	Z/1
0	0	0	0	–	–	–	–	–	X	–	X	–	X
0	0	1	0	–	–	–	X	–	X	–	X	–	X
0	1	0	0	–	X	–	–	–	X	–	X	–	X
0	1	1	1	X	–	X	–	–	–	X	–	X	–
1	0	0	1	–	–	–	–	X	–	–	–	X	–
1	0	1	1	–	–	–	–	X	–	–	–	X	–
1	1	0	1	–	–	–	–	X	–	–	–	X	–
1	1	1	1	–	–	–	–	–	–	–	–	X	–

Tabla 3.1.1-3: Tabla de verdad resultado de aplicar técnica Stuck-at sobre el circuito de ejemplo

Las columnas tituladas A/0; A/1; B/0; B/1; C/0; C/1; D/0; D/1; Z/0; Z/1 reflejan cada uno de los posibles fallos en los que se puede encontrar el circuito. Cada una de las anteriores columnas se han rellenado con (X) en caso de que el fallo sea detectado para esa entrada y con (–) en caso contrario, considerando que en cualquier columna Y/0 o Y/1, la parte Y del circuito se encuentra estancada en 0 o en 1.

La fila marcada en verde oscuro indica que el posible fallo del circuito real puede estar provocado por cualquiera de los posibles fallos indicados con una (X) en las distintas columnas.

Como podemos apreciar, mediante el uso de la técnica de Stuck-at podemos identificar aproximadamente en que parte del circuito se encuentra el fallo.

En este caso podemos descartar que las zonas C, D y Z estén conectadas a un voltaje constante, es decir, que sea donde se encuentre el fallo, porque más resultados reales que se muestran en la Tabla 3.1.1-2 también tendrían que ser erróneos si C, D o Z verdaderamente estuvieran conectadas a un voltaje constante.

Por lo tanto, el fallo debe estar provocado porque la zona A se encuentra conectada a un voltaje constante.

4. Desarrollo de la App y Pruebas

En esta sección se detallará cuáles son las herramientas utilizadas para la realización de la aplicación, como está organizada y que pruebas se han realizado para verificar su correcto funcionamiento.

4.1 Material Utilizado

Para la realización de este TFG se ha requerido de las siguientes herramientas físicas o de software:

- 2 Smartphones
- Ordenador Personal
- Programa Argouml
- Programa Eclipse
- Programa Android Studio
- Programa Word
- Programa Paint
- App Android Screen Recorder
- App Android Frame Capture

Los smartphones se han necesitado para verificar como se mostraría la aplicación en dispositivos móviles con distintas versiones del sistema operativo Android; asegurarse que el desempeño en ambos es el mismo y no se producen fallos inesperados. Uno de los smartphones usaba Android 5.0 Lollipop y el otro Android 8.1.0 Oreo.

El ordenador personal se ha utilizado para la creación de la aplicación con Android Studio y Eclipse, la terminación de esta memoria con Word, edición de algunas imágenes con Paint, obtención de imágenes con los toques realizados en la pantalla con las Apps Screen Recorder [6] y Frame Capture[7].

Argouml se ha usado para crear el diagrama de clases, diagrama de estados que cumplirá nuestra aplicación para que sea capaz de lograr su objetivo.

Eclipse se ha utilizado para crear un modelo básico que cumpliera con el objetivo de detectar fallos de stuck-at en puertas lógicas. Además, se ha usado para la creación del correcto funcionamiento de las puertas lógicas.

Android Studio se ha utilizado para adaptar el modelo básico creado en Eclipse, añadiéndole la interfaz gráfica y el almacenamiento de los problemas incrustados en la App en una base de datos SQLite.

4.2 Organización de la aplicación

Antes de diseñar y codificar la aplicación, hace falta definir bien cuales son los requisitos funcionales y no funcionales que tiene que cumplir, para en la etapa de pruebas verificar que la aplicación cumple con los requisitos.

Para definir el alcance y la funcionalidad de la aplicación se han realizado una serie de reuniones con el tutor. De dichas reuniones se han sacado los siguientes requisitos funcionales y no funcionales que la aplicación debe de cumplir. También se ha ido desarrollando la maqueta de la aplicación, que se podrá visualizar en el apartado Anexos B Maqueta final presentada al tutor.

4.2.1 Objetivos y Funcionalidad

4.2.1.1 Alcance

En el proyecto Stuck-at 2 se han incluido estos aspectos:

- Lista de ejercicios con problema de Stuck-at disponibles para su realización
- Posibilidad de eliminar los ejercicios para liberar espacio en el dispositivo móvil.
- Posibilidad de recuperar los ejercicios eliminados por accidente.
- Posibilidad de crear por parte del usuario un ejercicio con unas características muy específicas.
- Incorporación de un menú de ayuda que permita hacer un uso correcto de la aplicación accesible desde cualquier pantalla.
- Permitir rehacer un ejercicio tantas veces como el usuario quiera.
- Visualizar las estadísticas de uso del usuario, permitiendo restablecerlas.
- Informar de los creadores de la aplicación para permitir enviarles sugerencias.

El proyecto no contempla:

- Gestión interna de la comunicación entre usuario y desarrollador.

4.2.1.2 Funcionalidades de la Aplicación

La lógica interna de Stuck-at 2 se apoya en cinco subsistemas encargados de funciones diferentes (con la excepción de algunas tareas compartidas). Estos subsistemas son:

A. Subsistema de Gestión de Usuarios
B. Subsistema de Ejercicios
C. Subsistema de Puertas
D. Subsistema de Interfaz Gráfica
E. Subsistema de Almacenamiento

Subsistema de Gestión de Usuarios: Gestiona toda la información relativa al usuario de la aplicación, almacenando temporalmente los siguientes datos:

- N.º de ejercicios resueltos sin cometer ningún fallo
- N.º de ejercicios inventados
- N.º de ejercicios totales resueltos
- N.º de veces reiniciado algún ejercicio
- Tiempo total usado en la aplicación desde que se ha instalado
- Tiempo total gastado en resolver todos los ejercicios.

Todos los datos almacenados temporalmente son enviados al subsistema de Almacenamiento para que se guarden de manera permanente.

Subsistema de Ejercicios: Maneja los cambios realizados a la hora de resolver un ejercicio; notificando al subsistema de Gestión de Usuarios si el ejercicio se ha resuelto correctamente o no. También notifica al subsistema de Interfaz Gráfica para que se muestre por pantalla donde ha habido fallos y donde no.

Se encarga de distinguir entre el resultado esperado indicado por el usuario y el resultado correcto determinado por el sistema.

Subsistema de Puertas: Maneja el comportamiento de las puertas incluidas en el circuito ya sean puertas con fallos como puertas sin ningún fallo. También notifica al subsistema de Ejercicios de cuál es el resultado correcto del circuito según las características especificadas en el ejercicio.

Subsistema de Interfaz Gráfica: Gestiona como se ve la aplicación en la pantalla del smartphone. También gestiona las pulsaciones realizadas por el usuario de la aplicación.

Se encarga de permitir realizar las siguientes actividades por parte del usuario:

- Visualizar las estadísticas del usuario.
- Reiniciar las estadísticas del usuario.
- Ver la lista de ejercicios incluidos en la aplicación.
- Eliminar un ejercicio de la lista de ejercicios.
- Recuperar uno o más ejercicios de la lista de ejercicios disponibles para ser recuperados.
- Crear un ejercicio nuevo prefabricado.
- Resolver ejercicio de la lista de ejercicios incluidos en la aplicación.
- Ver esquema del circuito del ejercicio a resolver.
- Reiniciar resolución del ejercicio a resolver.
- Comprobar la solución propuesta del ejercicio.
- Visualizar la ayuda de la aplicación.
- Visualizar la información de contacto de los desarrolladores de la App.

En el caso de la *visualización de estadísticas del usuario* se encarga de solicitar los datos al subsistema de Gestión de Usuarios. Los datos recibidos del subsistema de Gestión de Usuarios son mostrados en la vista de estadísticas.

En el caso del *reinicio de las estadísticas del usuario* notifica la solicitud a los subsistemas de Gestión de Usuarios; de Ejercicios y de Almacenamiento para que realicen los cambios oportunos.

En el caso de la *visualización de los ejercicios incluidos en la aplicación* solicita los datos al subsistema de Almacenamiento y después muestra los datos recibidos.

En el caso de la *eliminación de un ejercicio de la lista de ejercicios* notifica al subsistema de Almacenamiento de cuál es el ejercicio a eliminar.

En el caso de la *recuperación de uno o más ejercicios* solicita al subsistema de Almacenamiento la lista de ejercicios disponibles para ser recuperados; notifica al subsistema de Almacenamiento cuales son los ejercicios que el usuario quiere recuperar para que el subsistema realice los cambios oportunos.

En el caso de la *creación de un ejercicio nuevo prefabricado* recoge los datos introducidos por el usuario y se los envía al subsistema de Ejercicios para que verifique que los datos son válidos. También le envía los datos validados al subsistema Almacenamiento para que se guarde de manera permanente.

En el caso de la *resolución de un ejercicio de la lista de ejercicios incluidos* en la aplicación se identifica el ejercicio solicitado por el usuario y se muestra una nueva interfaz para permitir realizar la resolución del ejercicio.

Una vez se muestra la nueva interfaz; el usuario podrá interactuar con la pantalla del dispositivo para poder resolver el ejercicio. Todos los datos introducidos relacionados con el ejercicio serán notificados al subsistema Ejercicios para que realice los cambios indicados.

En el caso de la *visualización del esquema del circuito del ejercicio a resolver* solicita los datos del circuito al subsistema de Ejercicios para representar correctamente el circuito.

Tanto para el caso de la *reiniciación de la resolución del ejercicio a resolver* como para el caso de la *comprobación de la solución propuesta del ejercicio* notifica la solicitud al subsistema de Ejercicios para que realice los cambios oportunos.

Tanto para el caso de la *visualización de la ayuda* como para el caso de la *visualización de la información de contacto de los desarrolladores* de la App la información está incorporada en el propio subsistema.

Subsistema de Almacenamiento: Guarda en el disco duro toda la información relativa a la aplicación para que está funcione correctamente. La información que almacena es la siguiente:

- Ejercicios incluidos en la aplicación ya sean inventados por el usuario, como los ejercicios que ya vienen incluidos en el código de la aplicación
- Datos relacionados con el usuario que aparecerán en las estadísticas.
- Ejercicios que han sido enviados a la papelera de reciclaje para ser eliminados.

4.2.2 Catálogo de Requisitos

4.2.2.1 Requisitos Funcionales

A continuación, se mostrará los RF (Requisito Funcional) de los que está compuesto la aplicación.

4.2.2.1.1 Subsistema de Gestión de Usuarios

RF 1. El sistema debe recolectar los datos de uso de la aplicación.

- 1.1. El sistema no solicitara al usuario ningún tipo de información personal.
- 1.2. El sistema solo podrá recolectar los siguientes tipos de datos de uso.
  - 1.2.1. El número de ejercicios resueltos sin cometer ningún fallo.
  - 1.2.2. El número de ejercicios inventados.
  - 1.2.3. El número de ejercicios totales resueltos.
  - 1.2.4. El número de veces que se ha reiniciado algún ejercicio.
  - 1.2.5. El tiempo total usado en la aplicación desde que se ha instalado.
  - 1.2.6. El tiempo total gastado en resolver todos los ejercicios resueltos hasta el momento.

RF 2. Toda información recolectada por el sistema debe ser accesible al usuario.

- 2.1. El usuario podrá reiniciar la información recolectada tantas veces como se desee. Excepto:
  - 2.1.1. El tiempo total usado en la aplicación no puede ser reiniciado.
  - 2.1.2. El número de ejercicios inventados no puede ser reiniciado.

4.2.2.1.2 Subsistema de Ejercicios

RF 3. El sistema debe distinguir entre la solución expresada por el usuario y la solución hallada por el propio sistema.

RF 4. El sistema debe de permitir internamente poder reiniciar la solución propuesta por el usuario.

- 4.1. El sistema debe notificar que la solución propuesta por el usuario ha sido reiniciada.

RF 5. El sistema debe de controlar el tiempo usado en resolver un ejercicio.

RF 6. El sistema debe de identificar el circuito que está relacionado con un ejercicio.

RF 7. El sistema debe de hallar la solución propia comparando la salida del circuito sin fallos con la salida del circuito con los fallos especificados en el ejercicio.

4.2.2.1.3 *Subsistema de Puertas*

RF 8. El sistema debe imitar el correcto funcionamiento de una puerta lógica.

- 8.1. Las puertas lógicas deben tener una sola salida.
- 8.2. Las puertas lógicas deben de tener una o más entradas.
- 8.3. Las puertas lógicas deben de comportarse teniendo en cuenta el tipo de fallo que contiene el circuito.

RF 9. El sistema debe permitir hallar la salida del circuito; teniendo en cuenta el fallo del ejercicio.

RF 10. El sistema debe crear circuitos electrónicos compuestos por las puertas lógicas especificadas en el RF 8.

4.2.2.1.4 *Subsistema de Interfaz Gráfica*

RF 11. La interfaz gráfica debe mostrar por pantalla la información solicitada por el usuario.

- 11.1. Para ello, la interfaz gráfica debe solicitar la información requerida a los otros subsistemas.
  - 11.1.1. Excepto en el caso de mostrar el menú de ayuda o la información de contacto de los desarrolladores que entonces la información debe ser proporcionada por el propio subsistema.

RF 12. La interfaz gráfica debe notificar las acciones indicadas por el usuario.

- 12.1. El reinicio de un ejercicio debe de ser notificado.
- 12.2. La comprobación de la solución de un ejercicio debe de ser notificado.
- 12.3. La visualización de las estadísticas del usuario debe de ser notificado.

RF 13. La interfaz gráfica debe de ser capaz de representar el diagrama eléctrico del circuito.

4.2.2.1.5 Subsistema de Almacenamiento

RF 14. El sistema debe de almacenar en memoria no volátil la información necesaria para funcionar correctamente.

14.1. El sistema debe guardar el estado actual de los Ejercicios.
14.2. El sistema debe de almacenar los datos de uso del usuario.
14.3. El sistema debe de guardar tanto los ejercicios que se encuentran en la papelera de reciclaje como los ejercicios que están listos para ser realizados.

RF 15. El sistema debe de recopilar la información importante solicitándola a otros subsistemas.

15.1. La información de los ejercicios debe ser proporcionada por el subsistema Ejercicios.
15.2. Los datos de uso del usuario deben de ser proporcionados por el subsistema de Gestión de Usuario.

4.2.2.2 Requisitos No Funcionales

RNF 1. La aplicación tendrá un tiempo de carga de los ejercicios optimizado.

RNF 2. La aplicación incluirá una interfaz específicamente diseñada para dispositivos móviles o Smartphones.

RNF 3. La aplicación no tendrá publicidad y se ofrecerá gratis al público.

RNF 5. La aplicación no requiere conexión a Internet para su correcto funcionamiento.

RNF 6. La aplicación será compatible en dispositivos Android con versiones 5.0 y superiores.

RNF 7. La aplicación solo puede hallar su propia solución cuando el usuario lo solicite.

RNF 8. La aplicación contara el tiempo de uso con un fallo de precisión de 1 segundo.

RNF 9. La aplicación incluirá una interfaz cuyos colores y diseño será parecidos a los diseños implementados en otras aplicaciones de DSLab UAM.

4.2.3 Diseño de Aplicación

Una vez tenemos los RF y RNF validados por nuestro tutor se empieza a diseñar la aplicación internamente mediante el uso de diagramas UML.

Se ha diseñado siguiendo la arquitectura software MVC(Modelo-Vista-Controlador), ya que se requiere el uso de interfaces gráficas y se desea que sea fácil de mantener.

1. Diagrama de clases

Como el diagrama de clases realizado para este proyecto es demasiado grande; se ha decidido incorporar en la memoria un diagrama de clases menos detallado realizado con la herramienta online (https://online.visual-paradigm.com/). En dicho diagrama sólo se ven las clases, interfaces, enumeraciones y dependencias de las que está formado el modelo del proyecto.

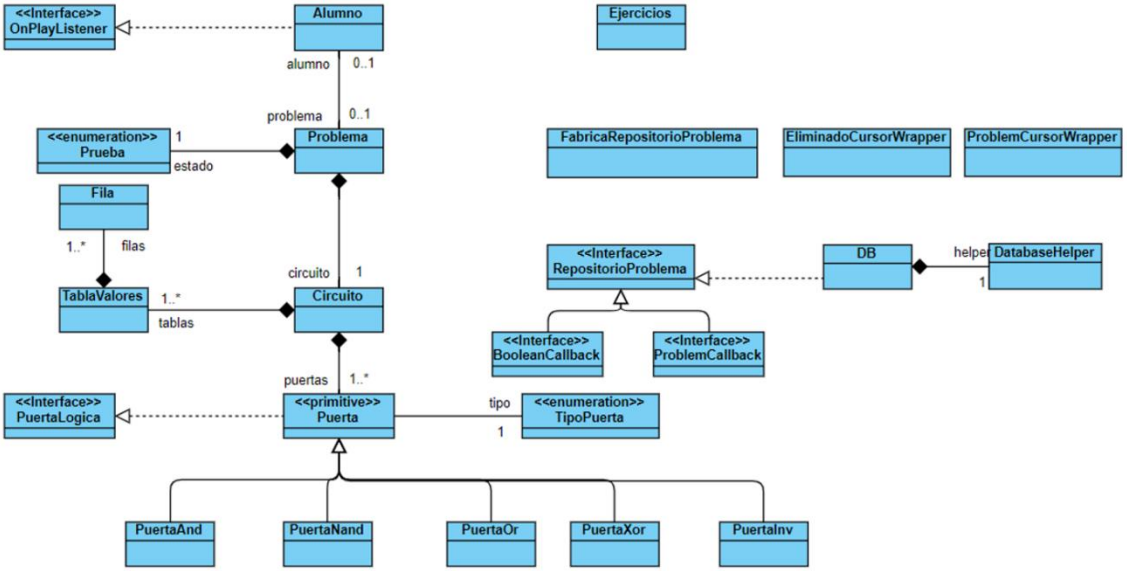

Figura 4.2.3-1: Diagrama de clases simplificado

Fuente: https://online.visual-paradigm.com/

Una vez hemos visualizado el Diagrama de clases simplificado se explicará en detalle cada una de las clases de las que está formado el modelo por el que está sustentado la aplicación en el Anexo Descripción de clases incluidas en el Diagrama de clases.

En la siguiente página, se mostrará un diagrama de estados que enseñará como se ha diseñado que funcione la aplicación para que se cumpla con los RF y RNF.

4.2.3.1 Diagrama de Estados

La siguiente imagen muestra el diagrama de estados desarrollado:

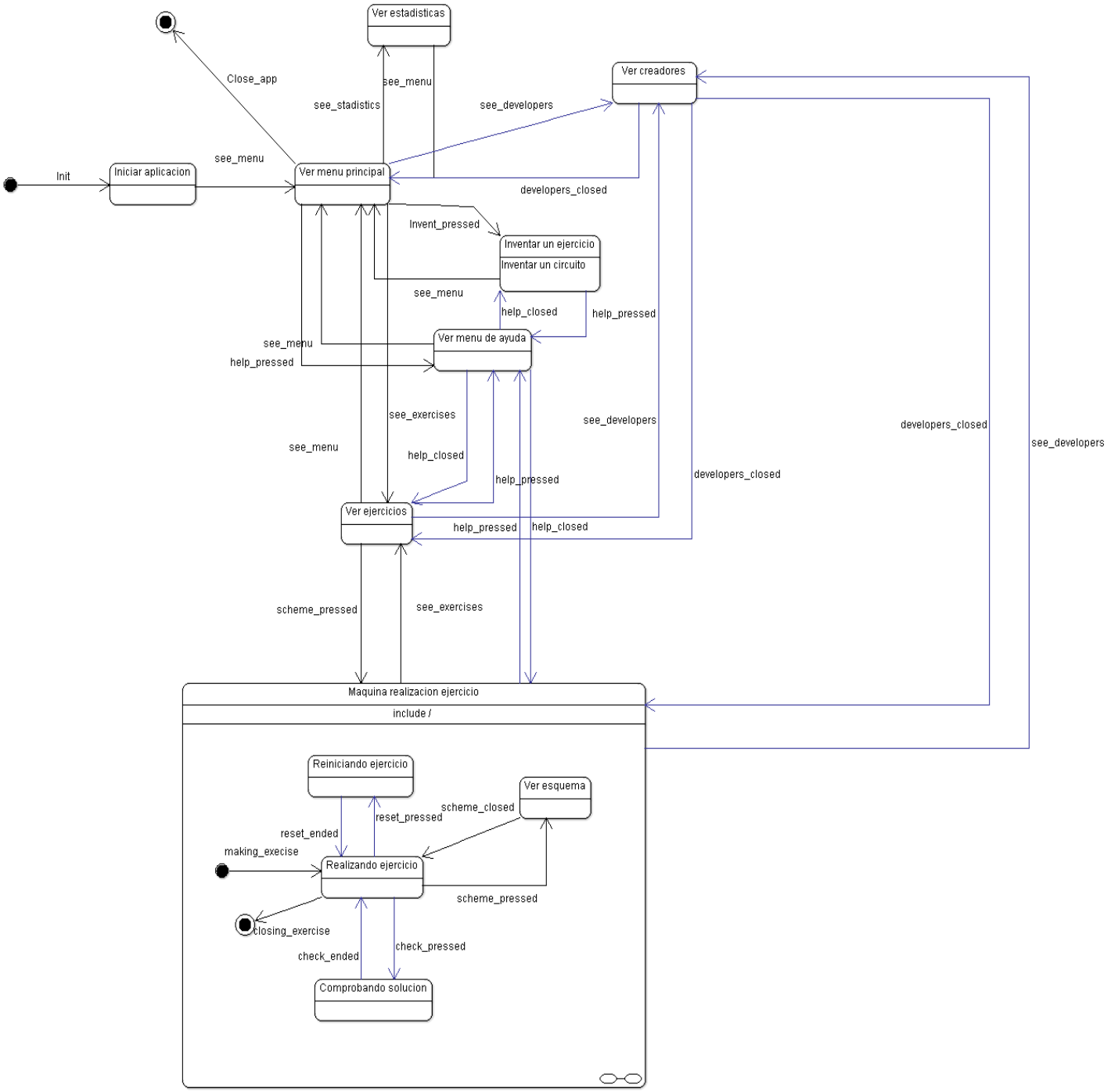

Figura 4.2.3-2: Diagrama de estados

Fuente: Propia

En la siguiente página describiremos el diagrama de estados.

4.2.3.1.1 Descripción de estados del diagrama de estados

El diagrama de estados está compuesto por los siguientes estados:

- *Iniciar Aplicación:* En este estado la aplicación carga los ejercicios incrustados en el código. Después se pasa al estado *Ver Menú Principal*.
- *Ver Menú Principal:* En este estado se muestra por pantalla algunas de las acciones que puede realizar el usuario. Las acciones que puede realizar son las siguientes:
    - Ver estadísticas: Para realizar esta acción se pasa al estado *Ver Estadísticas*.
    - Inventar un Ejercicio: Para realizar esta acción se pasa al estado *Inventar un Ejercicio*.
    - Ver Menú de Ayuda: Para realizar esta acción se pasa al estado *Ver Menú de Ayuda*.
    - Ver Ejercicios: Para realizar esta acción se pasa al estado *Ver Ejercicio.*
    - Ver Información de creadores: Para realizar esta acción se pasa al estado *Ver Creadores.*
- *Ver Creadores:* En este estado se muestra la información relacionada con los creadores y su dirección de contacto. Una vez el usuario ve la información de los creadores; la aplicación vuelve al estado del que vino.
- *Ver Estadísticas:* En este estado se muestra las estadísticas de uso del usuario de la aplicación. Una vez se ha visto las estadísticas o se han reiniciado las mismas se pasa al estado *Ver Menú Principal*.
- *Inventar un Ejercicio:* En este estado se permite al usuario crear un nuevo ejercicio con las especificaciones indicadas por el usuario. Una vez se ha creado el ejercicio se vuelve al estado *Ver Menú Principal*.
- *Ver Menú de Ayuda:* En este estado se puede ver el menú de ayuda para que el usuario haga un mejor uso de la aplicación. Una vez el usuario ve la ayuda que necesitaba; la aplicación vuelve al estado del que vino.
- *Ver Ejercicios:* En este estado se puede ver los ejercicios que se pueden resolver. Si el usuario quiere realizar un ejercicio de los ejercicios disponibles; se pasa al estado *Realizando Ejercicio*. Si por el contrario el usuario quiere volver al menú principal se pasará al estado *Ver Menú Principal*.
- *Realizando Ejercicio:* En este estado el usuario se encuentra resolviendo un ejercicio en específico. Si el usuario quiere reiniciar la solución del ejercicio entonces se pasará al estado *Reiniciando Ejercicio*. Si el usuario quiere ver el esquema del ejercicio entonces se pasará al estado *Ver Esquema*. Si el usuario quiere comprobar la solución que ha propuesto entonces se pasará al estado *Comprobando Solución*. Si el usuario no quiere seguir resolviendo el ejercicio entonces se pasará al estado *Ver Ejercicios.*
- *Reiniciando Ejercicio:* En este estado el usuario se encuentra reiniciando la solución propuesta en un ejercicio determinado. Una vez se ha reiniciado la solución se pasa al estado *Realizando Ejercicio*.
- *Ver Esquema:* En este estado el usuario se encuentra visualizando el esquema de un ejercicio determinado. Una vez se ha visto el esquema se pasa al estado *Realizando Ejercicio*.
- *Comprobando Solución:* En este estado el usuario se encuentra comprobando la solución propuesta en un ejercicio determinado. Una vez se ha terminado de comprobar la solución se pasa al estado *Realizando Ejercicio*.

4.2.4 Codificación de la aplicación

El desarrollo del código del modelo se ha realizado cumpliendo con la funcionalidad de las clases indicadas en el Diagrama de clases.

A continuación, se explicará cómo funcionan las actividades y fragmentos en Android Studio y se mostrará cuáles son las actividades y los fragmentos que se han desarrollado para la realización de este TFG.

4.2.4.1 Actividades y Fragmentos desarrollados en Android Studio

En Android las aplicaciones están formadas por Actividades y Fragmentos.
Tanto las actividades como los fragmentos tienen su propio ciclo de vida.

Las vistas con las que están relacionadas los fragmentos y las actividades están definidas en un archivo XML.

Durante el ciclo de vida de una actividad o fragmento se muestra la vista con la que están relacionados.

Los diferentes eventos que pueden ocurrir sobre una vista son tratados por los fragmentos o actividades. Los fragmentos o actividades modifican el modelo en función del evento recibido. Los cambios realizados en el modelo pueden o no cambiar la vista en función de si los datos que son modificados en el modelo están relacionados con la vista; entonces la vista cambiará. Y no en caso contrario.

Una vez se ha desarrollado el modelo de la aplicación podemos desarrollar las actividades y fragmentos de la aplicación en Android Studio.

Las actividades desarrolladas son las siguientes:

- AboutActivity
- AddCircuitActivity
- AddProblemaActivity
- HelpActivity
- HelpDetailActivity
- MainActivity
- ProblemaActivity
- Repositorio_Problemas_Activity
- ResetActivity
- SchemeActivity
- StatisticsActivity

Los fragmentos desarrollados son los siguientes:

- RepositorioFragment
- FragmentoProblema

Para evitar que la memoria sea demasiado larga de las anteriores actividades desarrolladas descartamos:

- HelpDetailActivity
- ProblemaActivity
- Repositorio_Problemas_Activity

HelpDetailActivity es descartada por ser una actividad demasiado simple; ya que muestra una determinada vista con la ayuda; en función del parámetro indicado en su creación. Además, cuando se pulsa el botón Close en la actividad HelpDetailActivity deja de mostrar la vista y termina.

ProblemaActivity y Repositorio_Problemas_Activity son descartados por ser actividades que ceden el manejo de la aplicación a los fragmentos FragmentoProblema y Repositorio-Fragment respectivamente. Aunque incluyen dichos fragmentos y son las responsables de implementar las interfaces de los fragmentos para que estos funcionen correctamente.

Tanto las vistas de las actividades como las vistas de los fragmentos se encuentran ubicados en el anexo Vistas de las actividades y de los fragmentos.

4.2.4.2 Actividades

A continuación, se explicará con mayor detalle cómo funciona las actividades no descartadas anteriormente de la aplicación.

4.2.4.2.1 AboutActivity

Muestra quienes han realizado la aplicación, cuáles son sus correos para contactar en caso de tener problemas o para ayudar a mejorar la aplicación.

Dichas acciones se realizan a través de la vista que tiene asociada.

Aparte de mostrar la vista con la información; permite que cuando se pulse en el botón Close se termine la actividad; dejando de mostrar la vista.

4.2.4.2.2 AddCircuitActivity

Pide especificar el tipo de puerta lógica; las entradas y las salidas por las que está formado cada una de las puertas lógicas del circuito.

Permite añadir puertas a un circuito inventado. Las puertas que se pueden añadir son OR, AND, NAND, XOR e INV. En caso de que se quiera añadir una OR, AND, NAND o XOR con más de una salida muestra un mensaje de error. En caso de que se quiera añadir una puerta INV con más de una entrada o más de una salida te muestra un mensaje de error.

4.2.4.2.3 StatisticsActivity

Muestra las estadísticas de uso del usuario.

Permite reiniciar las estadísticas de uso de la aplicación. El tiempo de uso y el número de ejercicios inventados no son reiniciados, cumpliendo con el RF 2.1.1 y RF 2.1.2.

4.2.4.2.4 AddProblemaActivity

Permite añadir un problema diseñado por el alumno a la aplicación. Se pide introducir el título del problema, las entradas del circuito del problema, la salida del circuito, los puntos intermedios del circuito y las puertas o nodos que componen el circuito a través de la actividad AddCircuitActivity.

El controlador evita añadir un problema sin título. En caso de tratar crear un problema cuyo circuito no tenga entradas ni salidas se muestra un mensaje de error. El controlador evita tratar de introducir a la aplicación problemas mal definidos. Para ello inicia la actividad AddCircuitActivity para obtener la definición del circuito; una vez tiene la definición comprueba que sea válida. Si no es válida la definición muestra un mensaje de que el circuito ha sido mal declarado. Se evita crear problemas con más de 3 entradas porque si no la tabla asociada al problema se vería muy pequeña.

4.2.4.2.5 ResetActivity

Permite recuperar un problema de la papelera de reciclaje o eliminar definitivamente los problemas encontrados en la papelera de reciclaje de la aplicación. En el caso de querer recuperar un problema se pide seleccionar el título del problema que se quiere recuperar y después pulsar en el botón Recover it! Pero en el caso de querer eliminar definitivamente se pide seleccionar el título del problema y después pulsar en el botón Remove it!

El controlador se asegura que sabemos que se eliminaran o se recuperaran los ejercicios seleccionados.

4.2.4.2.6 SchemeActivity

Muestra el esquema de un ejercicio en concreto.

El controlador de la actividad SchemeActivity además de mostrar la vista con el título del problema, el esquema y un botón; utiliza el botón para finalizar con la actividad.

4.2.4.2.7 HelpActivity

Muestra el menú de ayuda de la aplicación, el cual divide la ayuda en diferentes temáticas para facilitar encontrar la ayuda más rápidamente.

Permite visualizar la ayuda para abrir un problema, completar tablas, comprobar la solución propuesta, ver el esquema de un circuito, reiniciar un problema, eliminar un problema, inventar un problema mediante el uso de los respectivos botones que aparecen en la vista. Por último, se permite que cuando se pulse en el botón Close se termine la actividad; dejando de mostrar la vista.

4.2.4.2.8 MainActivity

Muestra el menú principal que está formado por 4 botones.

El 1º botón permite ver los ejercicios cargados en la aplicación como se ve en el fragmento RepositorioFragment.
El 2º botón permite empezar a inventar un problema de Stuck-at por parte del alumno como se ve en la actividad AddProblemaActivity.
El 3º botón permite visualizar las estadísticas de uso como se ve en la actividad StatisticsActivity.
El 4º botón permite visualizar la ayuda para hacer un uso correcto de la aplicación como se ve en la actividad HelpActivity.

4.2.4.3 Fragmentos

A continuación, se explicará con mayor detalle cómo funciona los fragmentos del programa.

4.2.4.3.2 FragmentoProblema

Permite realizar el ejercicio tipo examen en la aplicación.

Para realizar correctamente el ejercicio muestra el título del ejercicio o problema a resolver, una tabla, 4 botones y un cronometro.

La tabla está formada por tantas filas como posibles entradas puede tener el circuito del problema. Por tantas columnas como posibles errores tiene el problema a resolver. Cada celda indica si un determinado error stuck-at es detectado o no para una determinada entrada.

El 1º botón permite reiniciar la tabla del problema y el cronometro.
El 2º botón permite visualizar el esquema del problema como se ve en la actividad SchemeActivity.
El 3º botón permite comprobar la solución propuesta por el Alumno.
El 4º botón permite cerrar el problema y pasar al fragmento RepositorioFragment.
El cronometro cuenta cuanto tiempo ha pasado desde que has empezado a resolver el problema.

4.2.4.3.3 RepositorioFragment

Muestra los ejercicios cargados en la base de datos de la aplicación, de cada ejercicio te muestra el título; la fecha de su creación; el tiempo que llevas o has llevado realizando el ejercicio; el estado del ejercicio, es decir, en curso o terminado. Si se ha terminado muestra si has cometido errores o no.

Permite gestionar la vista asociada a la hora de eliminar problemas eliminando de la vista el problema señalado, mostrando un mensaje de que el problema ha sido eliminado y recargando la vista mostrando solamente el resto.

Permite recuperar los ejercicios anteriormente eliminados lanzando la actividad ResetActivity.

4.2.5 Pruebas de la Aplicación

Durante la realización de la aplicación se ha ido realizando pruebas para comprobar que lo implementado cumple con lo deseado.

4.2.5.1 Pruebas Unitarias

Son un tipo de prueba de caja negra; en dichas pruebas se compara el resultado esperado de una función aplicándole una entrada determinada con el resultado real. Si los resultados coinciden la función ha pasado la prueba. En caso contrario se considera que la implementación de la función no es correcta y habría que cambiarla.

En la siguiente página se verá un ejemplo de prueba unitaria desarrollada en este TFG.

```java
@Test
public void obtenerSalida() {
    List<Integer> entradas = new ArrayList<>();
    entradas.add(0);
    entradas.add(0);
    assertNull(p1.ObtenerSalida(entradas));
    assertNull(p1.ObtenerSalida(null));
    entradas.clear();
    entradas.add(0);
    entradas.add(0);
    entradas.add(0);
    assertEquals(0, (int) p1.ObtenerSalida(entradas).get(0));
    entradas.clear();
    entradas.add(0);
    entradas.add(0);
    entradas.add(1);
    assertEquals(0, (int) p1.ObtenerSalida(entradas).get(0));
    entradas.clear();
    entradas.add(0);
    entradas.add(1);
    entradas.add(0);
    assertEquals(0, (int) p1.ObtenerSalida(entradas).get(0));
    entradas.clear();
    entradas.add(0);
    entradas.add(1);
    entradas.add(1);
    assertEquals(0, (int) p1.ObtenerSalida(entradas).get(0));
    entradas.clear();
    entradas.add(1);
    entradas.add(0);
    entradas.add(0);
    assertEquals(0, (int) p1.ObtenerSalida(entradas).get(0));
    entradas.clear();
    entradas.add(1);
    entradas.add(0);
    entradas.add(1);
    assertEquals(0, (int) p1.ObtenerSalida(entradas).get(0));
    entradas.clear();
    entradas.add(1);
    entradas.add(1);
    entradas.add(0);
    assertEquals(0, (int) p1.ObtenerSalida(entradas).get(0));
    entradas.clear();
    entradas.add(1);
    entradas.add(1);
    entradas.add(1);
    assertEquals(1, (int) p1.ObtenerSalida(entradas).get(0));
}
```

Figura 4.2.5-1: Prueba unitaria de correcto funcionamiento de Puerta And

Fuente: Propia

En la anterior imagen se muestra una de las pruebas unitarias realizadas a la clase PuertaAnd. El test comprueba que las diferentes entradas en una puerta And producen la salida esperada coincidiendo con su tabla de verdad. También se verifica que no funcionan en caso de no pasarle todas las entradas necesarias.

4.2.5.2 Pruebas de interfaz grafica

Son un tipo de prueba que verifica que las interfaces de usuario diseñadas en los archivos XML correspondientes se muestran correctamente. Para la realización de este tipo de pruebas se dispone de un framework lanzado por Google llamado **Espresso**.

A continuación, un ejemplo de prueba unitaria usando el framework Espresso.

```
@Test
public void ProblemaActivity2Showed() {
    activityScenarioRule.launchActivity(ProblemaActivity.newIntent(getApplicationContext(), problem.guardarProblema()));
    onView(withText(problem.getTitle())).check(matches(isDisplayed()));
    onView(withText("1/1")).check(matches(isDisplayed()));
    onView(withId(R.id.tabla)).check(matches(isDisplayed()));
    onView(withText("Reset")).check(matches(isDisplayed()));
    onView(withText("Scheme")).check(matches(isDisplayed()));
    onView(withText("Check")).check(matches(isDisplayed()));
    onView(withId(R.id.tabla)).perform(click());
    onView(withId(R.id.tabla)).perform(swipeRight());
    onView(withText("No more content to the left")).check(matches(isDisplayed()));
    onView(withText("Close")).check(matches(isDisplayed()));
    activityScenarioRule.finishActivity();
}
```

Figura 4.2.5-2: Ejemplo de prueba de interfaz usando framework Espresso

Fuente: Propia

En el anterior ejemplo se comprueba que la interfaz gráfica del fragmento

FragmentoProblema se muestra correctamente en el dispositivo. También se comprueba que el ejercicio que se está tratando de resolver contiene solamente una tabla, y por lo tanto se muestra el mensaje “No more content to the left” cuando se realiza un SwipeRight sobre la tabla.

4.2.5.3 Pruebas de aceptación

Son el ultimo tipo de prueba realizado en este TFG; los usuarios finales de la Aplicación verifican que la aplicación satisface sus necesidades.

El resultado de la valoración de los usuarios finales se ve reflejado en la encuesta realizada mediante el uso de los formularios de Google. El enlace donde se encuentra la encuesta es https://forms.gle/obW5Jk7ZiHk3AuCz6.

A continuación, unos gráficos sacados de la valoración en la encuesta de los usuarios finales.

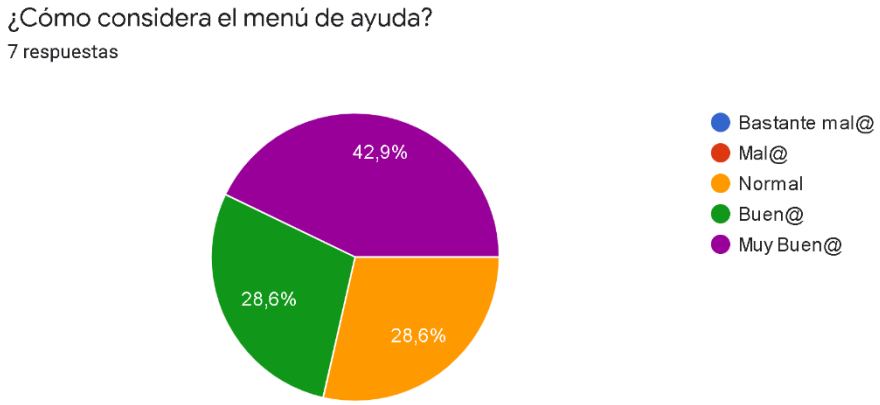

Figura 4.2.5-3: Respuestas de ¿Cómo … menú de ayuda?

Fuente: https://docs.google.com/forms

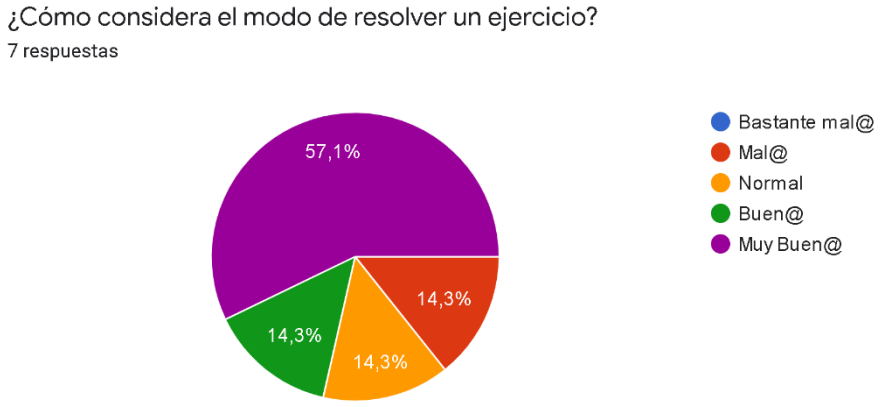

Figura 4.2.5-4: Respuestas de ¿Cómo considera el modo de resolver un ejercicio?

Fuente: https://docs.google.com/forms

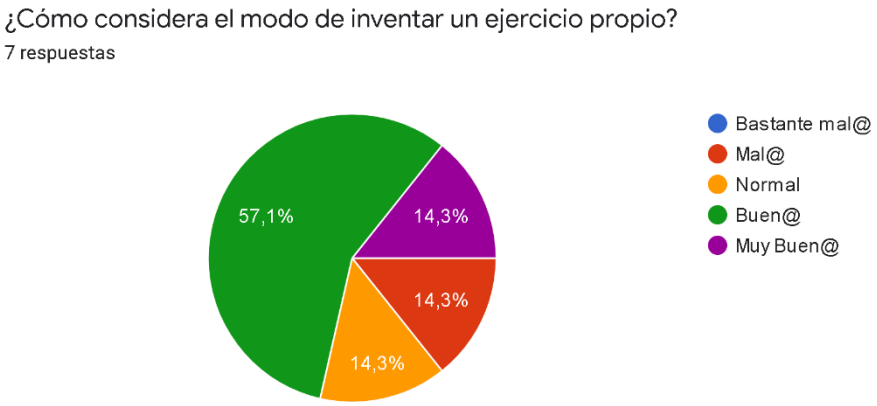

Figura 4.2.5-5: Respuestas de ¿Cómo considera el modo de inventar un ejercicio propio?

Fuente: **https://docs.google.com/forms**

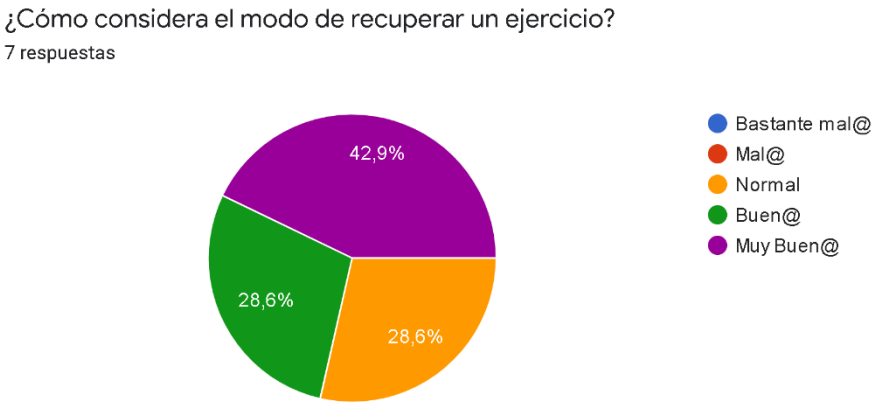

Figura 4.2.5-6: Respuestas de ¿Cómo … un ejercicio?

Fuente: **https://docs.google.com/forms**

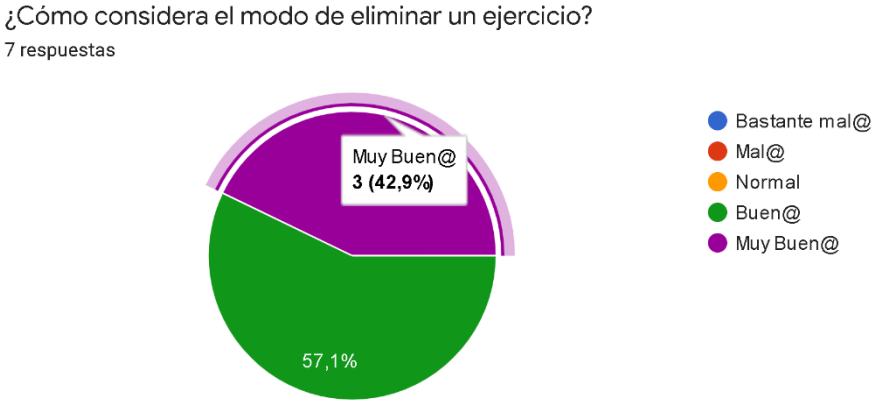

Figura 4.2.5-7: Respuestas de ¿Cómo considera el modo de eliminar un ejercicio?

Fuente: **https://docs.google.com/forms**

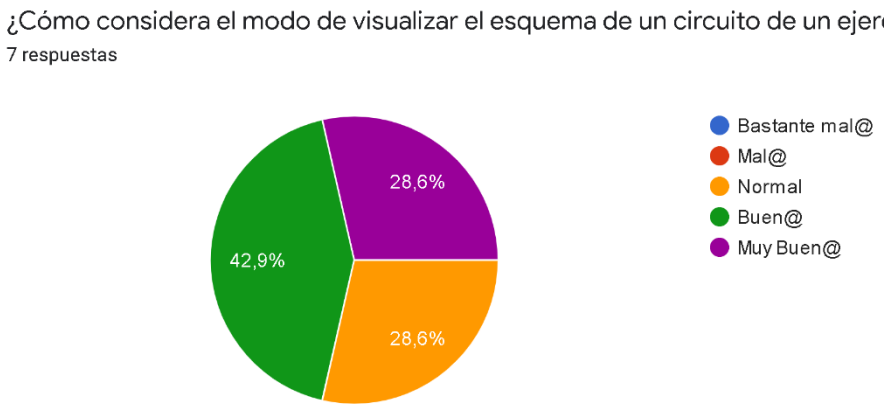

Figura 4.2.5-8: Respuestas de ¿Cómo considera el modo de visualizar el esquema de un circuito de un ejercicio?

Fuente: https://docs.google.com/forms

Figura 4.2.5-9: Respuestas de ¿Cómo considera el modo de visualizar los errores a la hora de resolver un ejercicio?

Fuente: https://docs.google.com/forms

5. Conclusiones y trabajo futuro

A continuación, se indicarán las conclusiones a las que se ha llegado mientras se estaba realizando el TFG y los posibles trabajos futuros que se podrían haber realizado durante el TFG si se le dedicara más tiempo.

5.1 Conclusiones

Las conclusiones que se han sacado en el desarrollo de este TFG son de 2 tipos:

- Como programador
- Como estudiante

5.1.1 Conclusiones como estudiante

Un TFG es un proyecto largo; requiere de muchas horas para llevarlo a cabo. Unas horas dedicadas para organizar la realización del TFG y otras horas para investigar posibles soluciones a problemas ocurridos durante el desarrollo del TFG.

Android es un sistema operativo para móviles muy versátil; ya que permite la realización de una misma tarea de maneras muy diferentes. Si se quiere realizar un programa móvil enfocado a un tema muy específico de la enseñanza; este sistema operativo es el adecuado. Ya que se puede encontrar en móviles de diferentes rangos de precios.

La realización de un programa para testear problemas de Stuck-at ha sido relativamente sencillo ya que Java contiene operadores binarios que realizan el funcionamiento de dichas puertas. Por ejemplo; el resultado de la puerta and para la entrada 0 0 0 se obtiene al usar el operador & sobre los 2 primeros ceros y otra vez sobre el otro cero y el resultado de la operación intermedia. La expresión que resume el ejemplo es la siguiente 0 & (0 &0).

Lo más complicado de realizar ha sido tratar de realizar la correcta representación del esquema de un circuito; ya que se tiene que calcular primero el espacio que ocupara el circuito a representar y después dibujar las puertas que lo componen juntos con sus determinadas conexiones.

Se ha decidido añadir un Manual de gestión de problemas incrustados en la APP para facilitar la gestión a posteriori de los problemas incluidos en el programa. En caso de necesitar incluir nuevos problemas a la aplicación sólo habría que realizar los pasos indicados en el manual.

5.1.2 Conclusiones como programador

El desarrollo de un TFG necesita de mucha depuración y de muchas pruebas. Los errores que se pueden producir en un programa muchas veces son difíciles de solucionar porque se suelen detectar los problemas en puntos del programa muy avanzados. Para solucionar dichos errores es mejor depurar el problema realizando los pasos previos a que se produjera error. Una vez se conoce las condiciones que produce el fallo; se crea una prueba que detecta ese fallo. Después de haber creado la prueba que detecta el fallo se corrige el problema.

Es recomendable crear pruebas que detecten los fallos inesperados para aumentar nuestro banco de pruebas. Un banco de pruebas grande permite garantizar que el producto final entregado al cliente contiene el menor número de fallos ya que la mayoría han sido detectados y reparados cuando se realizaban las pruebas.

Un código correctamente estructurado facilita su depuración y sus pruebas. También facilita el desarrollo de nuevas funcionalidades porque se sabría que nuevas funcionalidades son desarrolladas por determinadas clases. Por ejemplo; en nuestro código la implementación de una nueva puerta lógica heredaría de la clase Puerta y sería añadida al enum TipoPuerta; también implementaría aquellos métodos de la interfaz PuertaLogica que no han sido implementados por la clase Puerta.

5.2 Trabajo futuro

La aplicación desarrollada en el TFG cumple con el objetivo establecido. Pero este programa ha sido desarrollado para en un futuro tener las siguientes características:

- Servidor para profesores.
- Ranking de puntuaciones.
- Registro vinculado con la Universidad.
- Chat Alumno Profesor

5.2.1 Servidor para profesores

El Servidor para profesores tendrá varias funcionalidades incluidas. Entre ellas destaca que es el sitio donde los profesores de la asignatura de DIE podrán cargar circuitos con problemas de Stuck-at.

5.2.2 Ranking de puntuaciones

El ranking de puntuaciones se desarrollará para incentivar el aprendizaje del problema de Stuck-at de la manera más entretenida y divertida. Dicho ranking creará las puntuaciones de los usuarios en función de las siguientes características:

- El tiempo que tarda en resolver el problema.
- El número de errores cometidos.
- La dificultad considerada por el profesor cuando creó el problema

5.2.3 Registro vinculado con la Universidad

El registro vinculado con la Universidad permitirá a los profesores usar la aplicación para evaluar a los alumnos durante la impartición de la asignatura. Evitará el acceso a aquellos posibles usuarios que no formen parte de la Universidad.

5.2.4 Chat Alumno Profesor

El Chat Alumno Profesor permitirá solucionar de una manera más rápida posibles dudas o problemas que le puedan surgir al alumno mientras está resolviendo un ejercicio de la Aplicación; siempre y cuando el ejercicio sobre el que tenga dudas esté disponible en el Servidor de ejercicios.

Referencias

[1] Anonym, Automatic test pattern generation. Https://en.wikipedia.org/wiki/Automatic_test_pattern_generation

[2] Anonym, Design for testing. Https://en.wikipedia.org/wiki/Design_for_testing

[3] Anonym, Stuck-at fault. Https://en.wikipedia.org/wiki/Stuck-at_fault

[4] Anónimo, Espresso. https://es.wikipedia.org/wiki/Espresso_(framework)

[5] Roman Nurik, Launcher Icon Generator. https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html

[6] Kimcy929, Screen Recorder. https://play.google.com/store/apps/details?id=com.kimcy929.screenrecorder

[7] Frame Capture, Frame Capture. https://play.google.com/store/apps/details?id=com.framecapture.framecapture

[8] Pedro Madrigal Marina. Aplicación de problemas resueltos de circuitos digitales combinacionales bajo Android. https://repositorio.uam.es/handle/10486/660545

[9] Pablo Molinero Merino. Aplicación de problemas resueltos de circuitos digitales secuenciales bajo Android. https://repositorio.uam.es/handle/10486/660543

[10] Ana González Fernández. Aplicación Android para la enseñanza de temas de circuitos integrados básicos MOS. https://repositorio.uam.es/handle/10486/663772

[11] Mariano José Casado Abril. Tutorial interactivo en Android sobre circuitos aritméticos. https://repositorio.uam.es/handle/10486/677847

[12] Daniel Gómez García. Aplicación Android sobre mapeo de funciones lógicas en Look-Up Tables. https://repositorio.uam.es/handle/10486/679339

[13] Juan Burgos Abadie. Aplicación de problemas resueltos de circuitos digitales combinacionales bajo Android. https://repositorio.uam.es/handle/10486/669462

[14] Cristina María Sobrino Hipólito. Aplicación Android para resolver problemas de circuitos digitales secuenciales. https://repositorio.uam.es/handle/10486/669567

[15] Google, Espresso.https://developer.android.com/training/testing/espresso/basics

[16] Adriano Moutinho. Solución para mapas Karnaugh. https://play.google.com/store/apps/details?id=karnagh.ammsoft.karnagh

[17] Adrián Vaca Humanes. DiCiDe: Circuitos digitales. https://play.google.com/store/apps/details?id=com.avh.digitalcircuitdesign

[18] Intelitech. Digital Circuits. https://play.google.com/store/apps/details?id=in.softecks.digitalcircuits

[19] Leonardo Mesquita / Bruno Silva. Eletrônica Digital. https://play.google.com/store/apps/developer?id=Leonardo+Mesquita+/+Bruno+Silva

[20] SCHILLER App. Calculadoras de Ingeniería Electrónica. https://play.google.com/store/apps/details?id=com.schiller.herbert.calcparaeletronicafree

[21] Google. Android. https://www.android.com/intl/es_es/

Glosario

API	Application Programming Interface
MVC	Modelo Vista Controlador
PFC	Proyecto Fin de Carrera
UAM	Universidad Autónoma de Madrid
EPS	Escuela Politécnica de Madrid
DSLab UAM	Digital System Laboratory Universidad Autónoma de Madrid
TFG	Trabajo de Fin de Grado
URL	Uniform Resource Locator
UML	Lenguaje Unificado de Modelado
Framework	Marco de trabajo

Anexos

A. Manual de gestión de problemas incrustados en la APP

Los problemas incrustados en la App son cargados en segundo plano por una función. Dicha función se llama cargarEjercicios y se encuentra disponible en la clase Ejercicios.

La clase Ejercicios se encarga de la gestión de carga de los Ejercicios realizando las siguientes acciones:

- Comprueba que los ejercicios no han sido cargados anteriormente.
- Indica que los ejercicios se han cargado.
- Carga los ejercicios incrustados en la función cargar Ejercicios siempre y cuando esos ejercicios no estén ya cargados en la aplicación.

La función solamente carga los ejercicios que tiene incrustados en la función siempre y cuando no se encuentren ya cargados en la App.

Si se quiere añadir nuevos problemas a la aplicación o editar los problemas incorporados en la aplicación se debe realizar los siguientes pasos:

1. Cambiar la versión de la base de datos por una superior o inferior.
2. Editar la función cargarEjercicios para que contenga los ejercicios incrustados.
3. Verificar que los ejercicios indicados en la función cargarEjercicios contienen como máximo 3 entradas; porque si contiene más de 3 entradas la tabla relacionada con el problema se vería de manera errónea.
4. Verificar que se cargan solamente los ejercicios que no se encuentran ya cargados en la aplicación.

La aplicación no distingue entre usuarios nuevos y usuarios viejos.

Por lo tanto, los problemas incorporados en versiones anteriores de la aplicación solamente los tendrán los usuarios viejos de la aplicación. Si se desea que esos problemas lo tengan los usuarios de las nuevas versiones de la aplicación no deben ser eliminados de la función de la aplicación.

B. Maqueta final presentada al tutor

Según se iban haciendo reuniones con el tutor para conocer el objetivo final de la aplicación solicitada se ha ido desarrollando una maqueta que satisfacía las necesidades expuestas.

La maqueta se ha usado para:

- Corregir aspectos visuales finales de la aplicación.
- Verificar los RF y RNF demandados por el tutor.

La maqueta final tiene la siguiente apariencia:

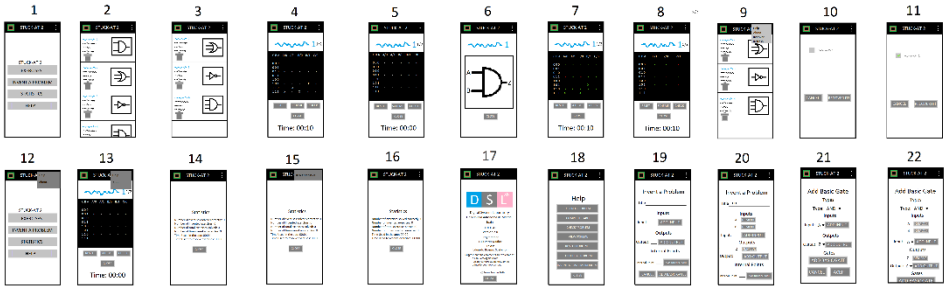

Figura B-1: Maqueta final de la App

Fuente: Propia

A continuación, se agruparán las interfaces de la aplicación por funcionalidad de la aplicación.

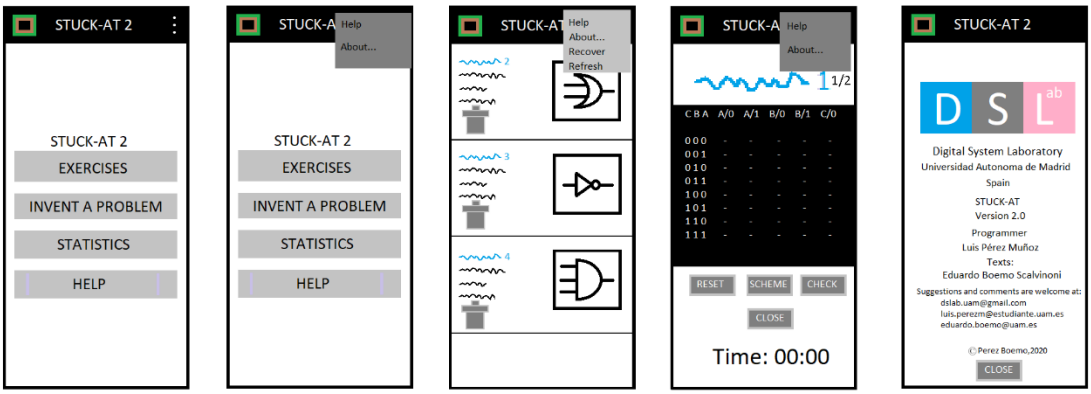

Figura B-2: Visualizar creadores de la App

Fuente: Propia

La Figura B-2 muestra las diferentes maneras posibles para visualizar la información de los creadores de la App.

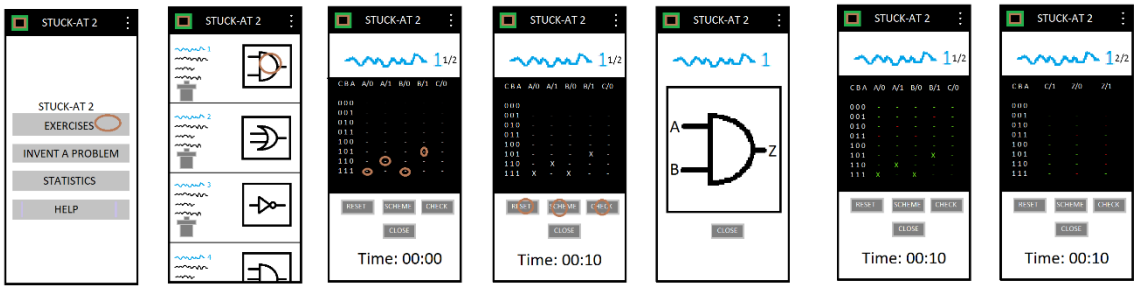

Figura B-3: Realización de ejercicio en la App

Fuente: Propia

La Figura B-3 muestra los pasos necesarios para realizar un ejercicio en la App.

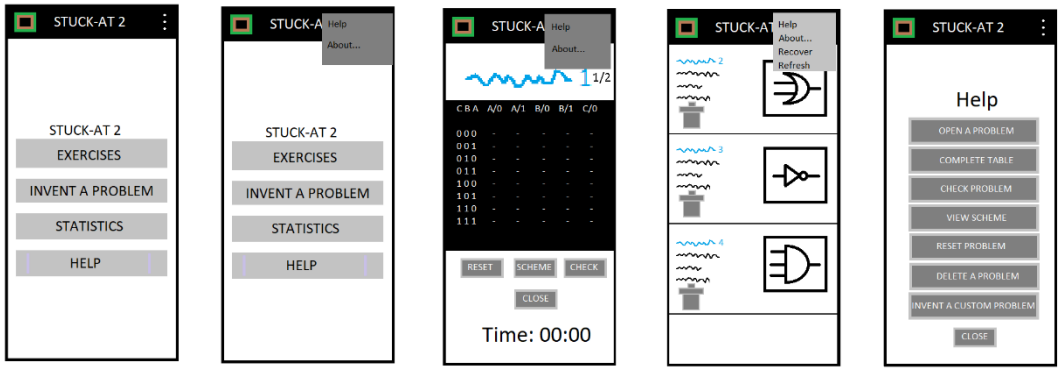

Figura B-4: Visualización de menú de Ayuda

Fuente: Propia

La Figura B-4 muestra las distintas maneras de acceder al menú de ayuda pulsando el botón Help.

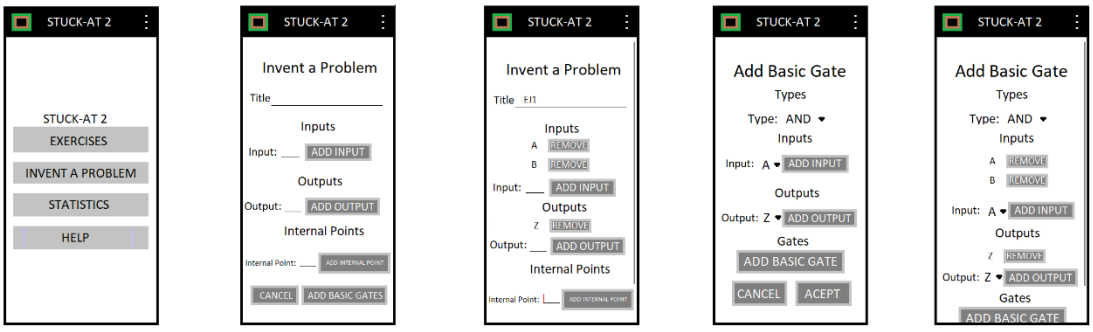

Figura B-5: Invención de ejercicio en la App

Fuente: Propia

La Figura B-5 muestra los pasos necesarios para inventar un ejercicio en la App.

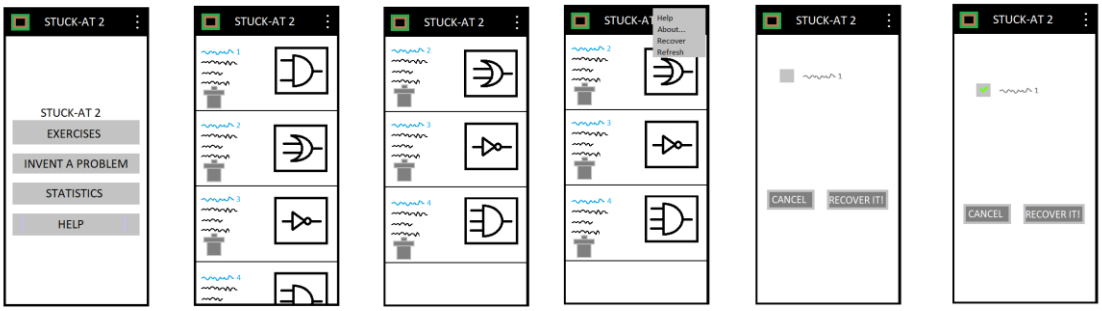

Figura B-6: Recuperación de ejercicio en la App

Fuente: Propia

La Figura B-6 muestra el proceso de recuperación de un ejercicio eliminado por error.

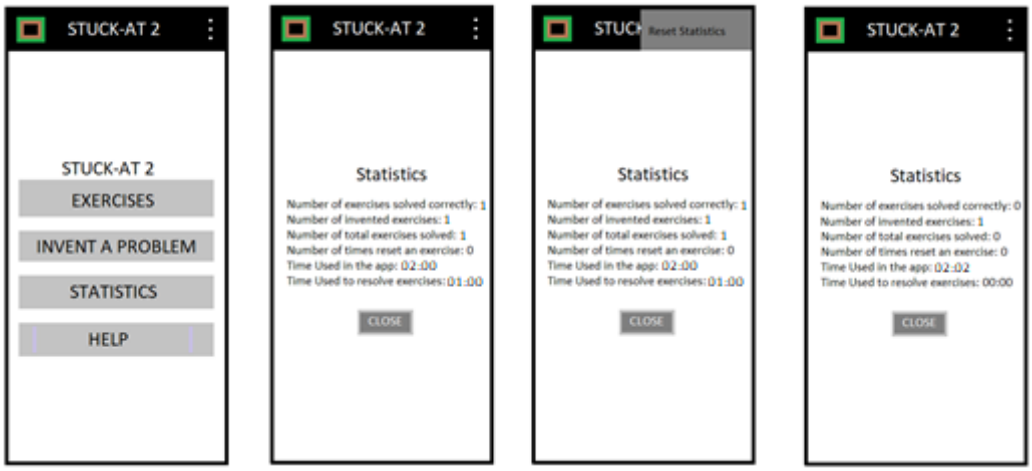

Figura B-7: Visualización de estadísticas en la App

Fuente: Propia

La Figura B-7 muestra los datos que aparecerán tras pulsar en Statistics y lo que ocurrirá al reiniciar las estadísticas.

C. Descripción de clases incluidas en el Diagrama de clases

Las interfaces, enumeraciones y clases que se describirán en este anexo forman parte del Diagrama de clases simplificado de la Figura 4.2.3-1.

C.1Interfaces

Las interfaces definidas permiten ajustar el comportamiento de la aplicación mediante los métodos que incluyen.

Si se desea incrementar las funcionalidades de ciertos componentes de la aplicación, sería recomendable añadir esa nueva funcionalidad a la interfaz correspondiente.

Las interfaces definidas en el diagrama de clases para que la aplicación funcione correctamente son las siguientes:

C.1.1 Interfaz OnPlayListener

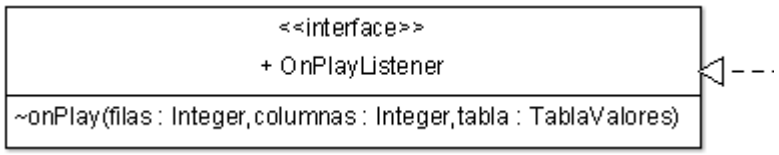

Figura C.1.1-1: Interfaz OnPlayListener en Diagrama de clases

Fuente: Propia

La interfaz OnPlayListener se ha diseñado para definir el método a implementar para poder resolver los ejercicios existentes en la aplicación.

El método requerirá de la fila y la columna para determinar la celda de la tabla que se ha pulsado. También requerirá de la tabla para saber el valor actual de dicha celda y poder cambiarlo.

C.1.2 Interfaz BooleanCallback

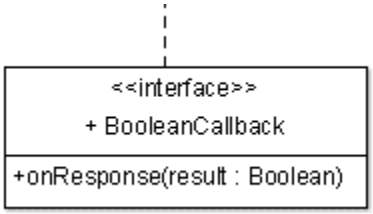

Figura C.1.2-1: Interfaz BooleanCallback en Diagrama de clases

Fuente: Propia

La interfaz BooleanCallback se ha definido para obtener el resultado de una transacción en la base de datos.
El método onResponse permitirá al usuario de la aplicación saber si su acción ha sido guardada correctamente o si por el contrario se ha producido un error.

C.1.3 Interfaz ProblemCallback

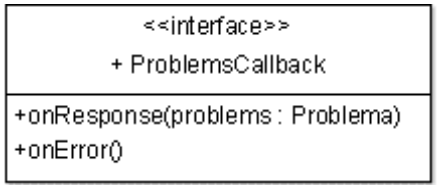

Figura C.1.3-1: Interfaz ProblemCallback en Diagrama de clases

Fuente: Propia

La interfaz ProblemCallback se ha definido para indicar el resultado de tratar de obtener los ejercicios de la base de datos.
El método onResponse indica cuales son los ejercicios que se han podido obtener.
El método onError indica que el tratar de obtener los ejercicios de la base de datos ha sido erróneo.

C.1.4 Interfaz PuertaLogica

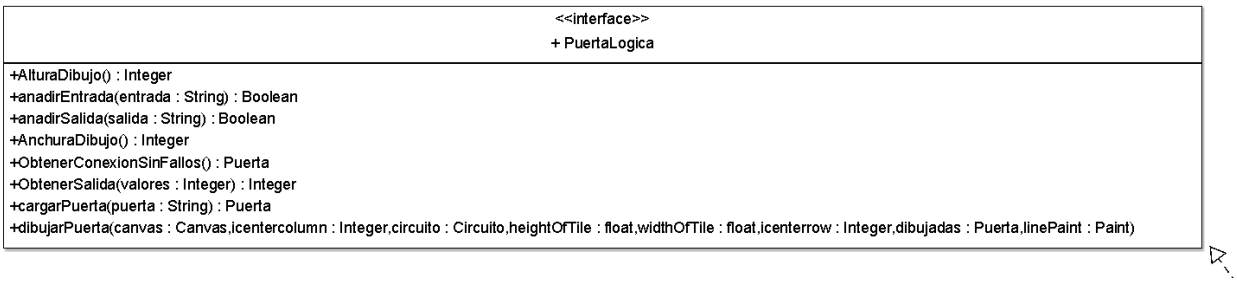

Figura C.1.4-1: Interfaz PuertaLogica en Diagrama de clases

Fuente: Propia

La interfaz PuertaLogica se ha diseñado para definir los métodos que deben implementar los diferentes tipos de puertas incluidos en la aplicación. Dichos métodos son:

- AlturaDibujo: Indica la altura que ocuparía un tipo de puerta determinado.
- AnchuraDibujo: Indica la anchura que ocuparía un tipo de puerta determinado.
- anadirEntrada: Permite controlar el número de entradas incluidas en un tipo de puerta determinado.
- anadirSalida: Permite controlar el número de salidas incluidas en un tipo de puerta determinado.
- ObtenerConexionSinFallos: Permite obtener una puerta idéntica a la existente, pero sin ningún fallo de Stuck-at.

- ObtenerSalida: Permite determinar el valor de una determinada salida para una determinada entrada teniendo en cuenta los posibles fallos que pueda tener la puerta sobre la que se está aplicando el método.
- cargarPuerta: Permite obtener una puerta determinada a partir de sus características recibidas como cadena de caracteres.
- dibujarPuerta: Permite dibujar una puerta en un Canvas dado indicandole donde se dibuja y el pincel usado para pintarlo.

C.1.5 Interfaz RepositorioProblema

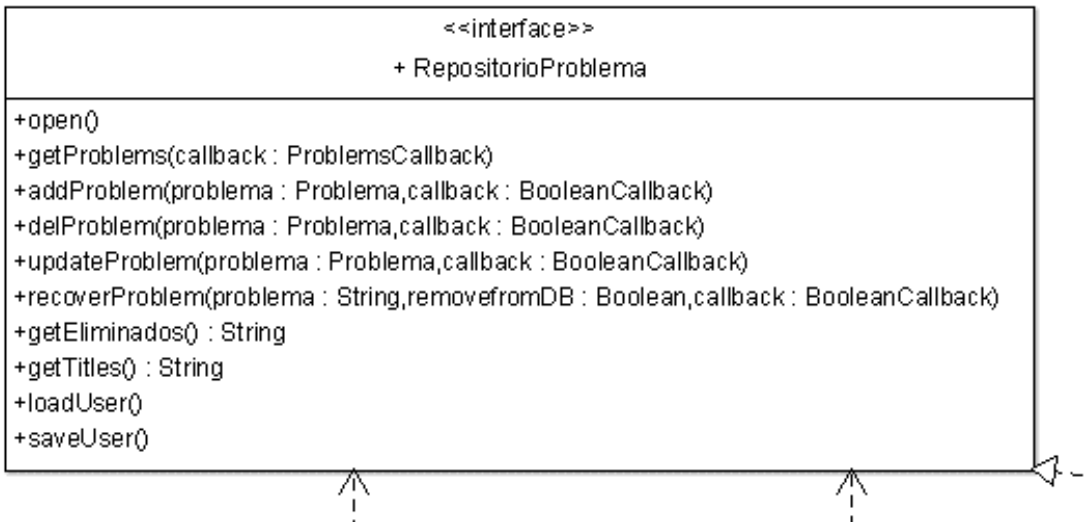

Figura C.1.5-1: Interfaz RepositorioProblema en Diagrama de clases

Fuente: Propia

La interfaz RepositorioProblema se ha diseñado para incluir todos los métodos necesarios a implementar por una base de datos para que la aplicación funcione correctamente. Los métodos que implementar son los siguientes:

- open: Permite abrir la base de datos para realizar el resto de las operaciones.
- getProblems: Permite obtener los problemas cargados en la base de datos.
- addProblems: Permite añadir un nuevo problema a la base de datos.
- delProblem: Permite enviar a la papelera de reciclaje un problema existente en la base de datos.
- updateProblem: Permite actualizar un problema existente de la base de datos.
- recoverProblem: Permite recuperar un problema de la papelera de reciclaje o eliminarlo definitivamente de la base de datos.
- getEliminados: Permite obtener los títulos de los problemas que se encuentran en la papelera de reciclaje de la base de datos.
- getTitles: Permite obtener los títulos de los problemas de la base de datos.
- loadUser: Permite cargar los datos estadísticos del usuario guardados en la base de datos.
- saveUser: Permite guardar los nuevos datos estadísticos del usuario en la base de datos.

C.2Enumeraciones

Las enumeraciones definidas en el diagrama de clases permiten identificar rápidamente las puertas que están implementadas de momento en la aplicación y los diferentes estados por los que se puede encontrar un ejercicio.

Si en futuras actualizaciones se desea añadir nuevas puertas lógicas o nuevos estados de un problema; sería recomendable modificar las respectivas enumeraciones.

Las enumeraciones definidas son las siguientes:

C.2.1 Enumeración TipoPuerta

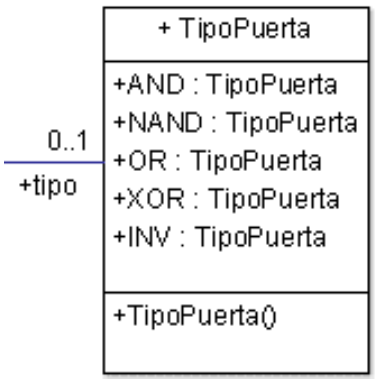

Figura C.2.1-1: Enumeración TipoPuerta en Diagrama de clases

Fuente: Propia

La enumeración TipoPuerta se ha diseñado para definir cuáles son los tipos de puertas que puede incluir un circuito.

Los tipos de puertas definidos son los siguientes:

- Puerta And
- Puerta Nand
- Puerta Or
- Puerta Xor
- Puerta Inv.

C.2.2 Enumeración Prueba

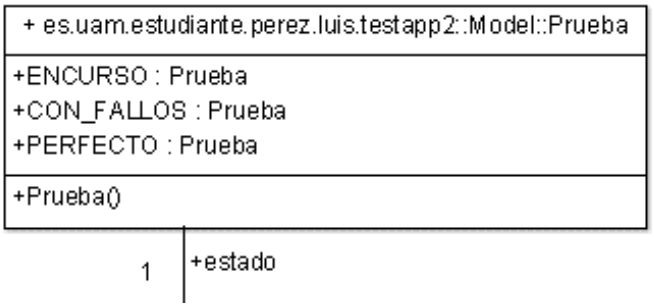

Figura C.2.2-1: Enumeración Prueba en Diagrama de clases

Fuente: Propia

La enumeración Prueba se ha diseñado para definir los distintos estados por los que se puede encontrar un ejercicio.

Los estados definidos son los siguientes:

- Ejercicio en curso.
- Ejercicio finalizado con fallos.
- Ejercicio finalizado sin fallos.

C.3Clases

Las clases definidas en el diagrama de clases permiten el correcto funcionamiento de la aplicación. Algunas clases implementando los métodos definidos en las interfaces con las que están relacionadas.

Las clases PuertaAnd, PuertaNand, PuertaOr, PuertaXor y PuertaInv no tienen una descripción de sus métodos ya que implementan la interfaz PuertaLogica; la cual contiene la descripción de los métodos. Y el único método que no implementan de la PuertaLogica es el constructor.

La descripción de las clases definidas en el diagrama empieza en la siguiente página.

C.3.1 Clase EliminadoCursorWrapper

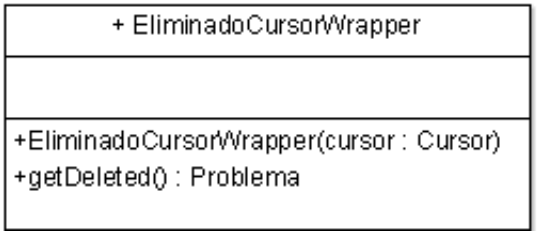

Figura C.3.1-1: Clase EliminadoCursorWrapper en Diagrama de clases

Fuente: Propia

La clase EliminadoCursorWrapper extiende de la clase CursorWrapper; se ha diseñado para obtener los ejercicios que han sido marcados para eliminar por parte del Alumno.

Los métodos implementados son los siguientes:

- EliminadoCursorWrapper: permite instanciar un objeto de tipo EliminadoCursor-Wrapper a partir de una instancia Cursor.
- getDeleted: permite obtener un problema marcado como eliminado en la base de datos.

C.3.2 Clase ProblemCursorWrapper

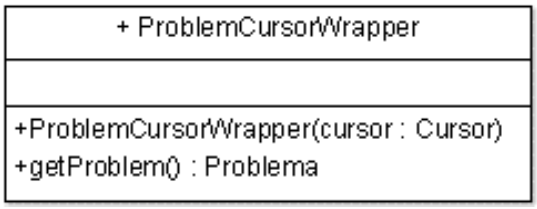

Figura C.3.2-1: Clase ProblemCursorWrapper en Diagrama de clases

Fuente: Propia

La clase ProblemCursorWrapper extiende de la clase CursorWrapper; se ha diseñado para obtener los ejercicios que se pueden resolver de la base de datos por parte del Alumno.

Los métodos implementados son los siguientes:

- ProblemCursorWrapper: permite instanciar un objeto de tipo ProblemCursorWrap-per a partir de una instancia Cursor.
- getProblem: permite obtener un problema de la base de datos.

C.3.3 Clase Alumno

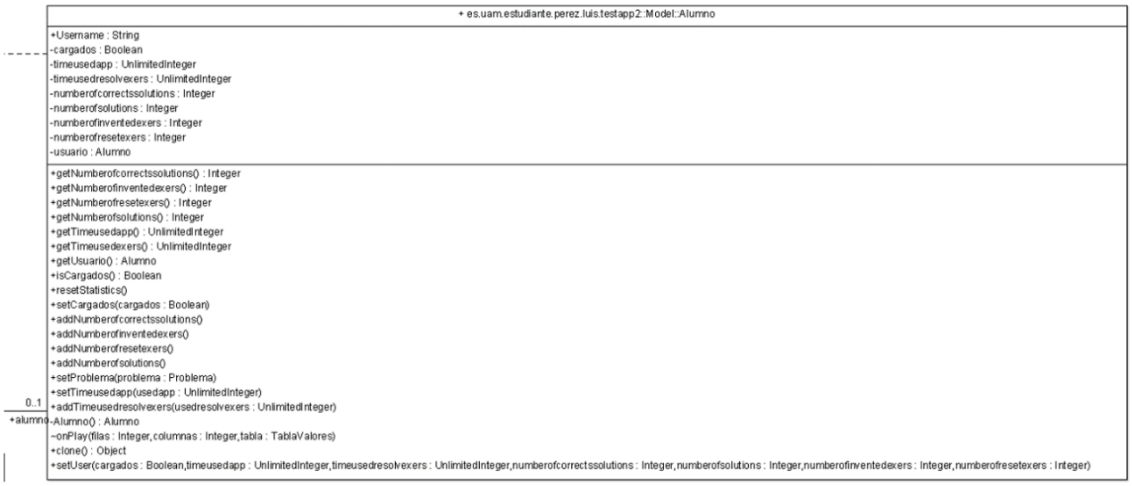

Figura C.3.3-1: Clase Alumno en Diagrama de clases

Fuente: Propia

La clase Alumno implementa la interfaz OnPlayListener; se ha diseñado para almacenar temporalmente toda la información relativa al uso de la aplicación por parte del usuario.

Los métodos implementados son los siguientes:

- getNumberofcorrectssolutions: permite obtener el número total de ejercicios resueltos correctamente.
- getNumberofinventedexers: permite obtener el número total de ejercicios inventados en la aplicación.
- getNumberofresetexers: permite obtener el número total de veces reseteados todos los ejercicios de la aplicación.
- getNumberofsolutions: permite obtener el número total de ejercicios resueltos.
- getTimeuseapp: permite obtener el tiempo total usado en la aplicación.
- getTimeuseexers: permite obtener el tiempo usado en resolver los ejercicios.
- getUsuario: permite obtener el usuario de la aplicación.
- isCargados: permite identificar si se han cargado todos los ejercicios o no.
- resetStatistics: permite reiniciar las estadísticas de uso del usuario.
- setCargados: permite señalar si se han cargado o no los ejercicios.
- addNumberofcorrectsolutions: permite incrementar en 1 el número de ejercicios resueltos correctamente.
- addNumberofinventexers: permite incrementar en 1 el número de ejercicios inventados en la aplicación.
- addNumberofresetexers: permite incrementar en 1 el número de ejercicios reiniciados en la aplicación.
- addNumberofsolutions: permite incrementar en 1 el número de ejercicios resueltos.
- setProblema: permite asignar el problema que se va a resolver al alumno.
- setTimeusedapp: permite señalar el tiempo usado en la aplicación.
- addTimeusedresolvexers: permite incrementar el tiempo en resolver los ejercicios en la cantidad indicada.
- Alumno: permite instanciar un objeto de tipo Alumno.

- onPlay: permite realizar el test de la aplicación.
- clone: impide crear más copias del usuario.
- setUser: permite cargar toda la información relativa a las estadísticas de uso.

C.3.4 Clase Problema

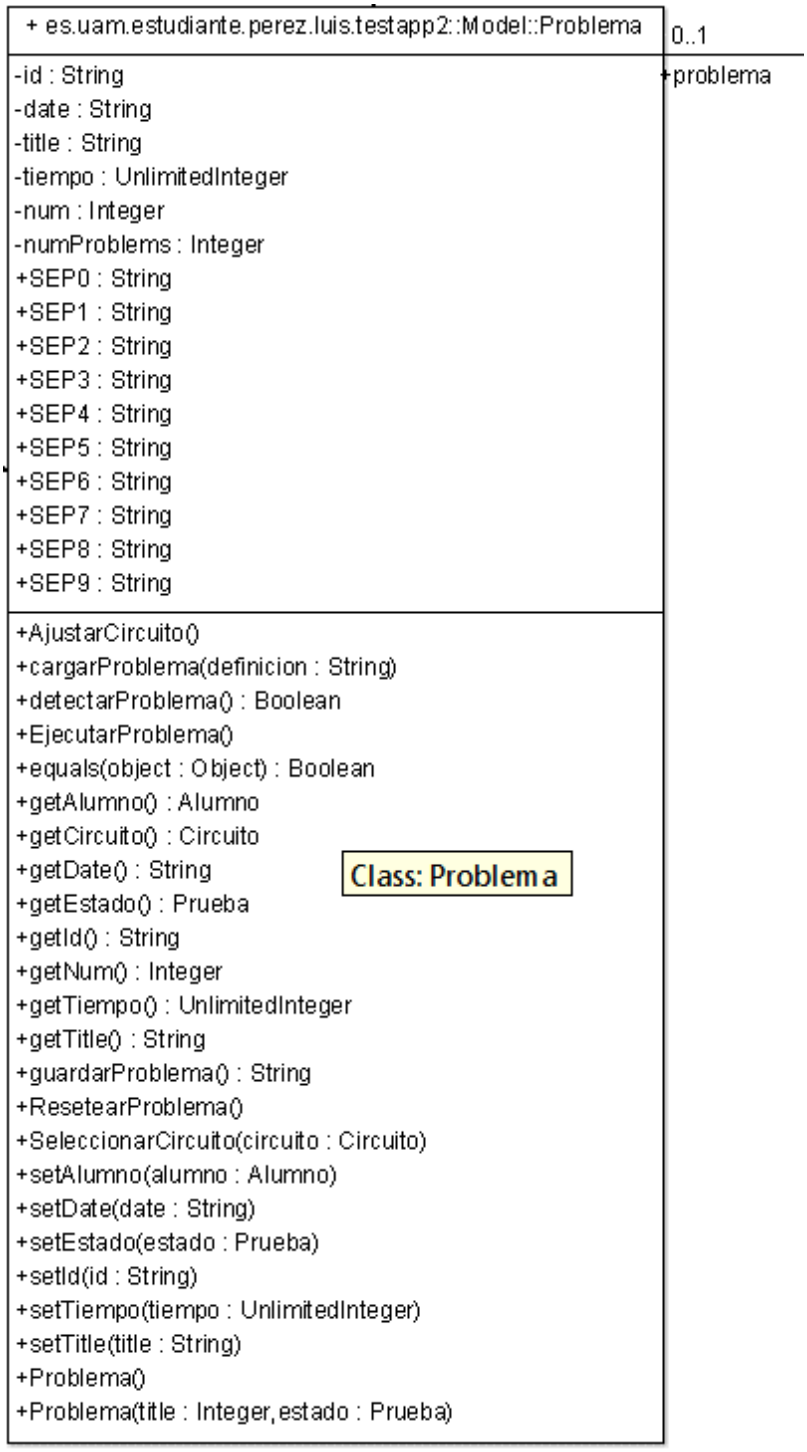

Figura C.3.4-1: Clase Problema en Diagrama de clases

Fuente: Propia

La clase Problema se ha diseñado para almacenar toda la información relativa a un ejercicio.

Entre dicha información se encuentra:

- el estado en el que se puede encontrar el ejercicio (Prueba).
- el circuito que tiene problemas de Stuck-at (Circuito).

Los métodos implementados aparecen en la siguiente página:

- AjustarCircuito: permite detectar errores en la declaración del circuito.
- cargarProblema: permite cargar un ejercicio a través de un String.
- detectarProblema: permite detectar si el usuario se ha equivocado o no a la hora de resolver el ejercicio.
- EjecutarProblema: permite hallar las diferentes salidas del circuito del ejercicio con problemas de Stuck-at.
- Equals: permite determinar si dos ejercicios son iguales.
- getAlumno: permite obtener el Alumno que está resolviendo el ejercicio.
- getCircuito: permite obtener el Circuito con problemas de Stuck-at que está relacionado con el ejercicio.
- getDate: permite obtener la fecha de creación del ejercicio
- getEstado: permite obtener el estado en el que se encuentra el ejercicio.
- getId: permite obtener el identificador único del ejercicio.
- getNum: permite obtener el número que identifica cuando se ha creado el ejercicio.
- getTiempo: permite obtener el tiempo empleado en resolver el ejercicio.
- getTitle: permite obtener el título del ejercicio.
- guardarProblema: permite guardar la información relacionada con el ejercicio en un String.
- ResetearProblema: permite reiniciar las tablas del ejercicio.
- SeleccionarCircuito: permite indicar cual es el circuito del ejercicio.
- setAlumno: permite indicar cual es el Alumno que resuelve el ejercicio.
- setDate: permite indicar cual es la fecha de creación del circuito.
- setEstado: permite indicar cual es el estado en el que se encuentra el ejercicio.
- setId: permite establecer el identificador único del ejercicio.
- setTiempo: permite señalar el tiempo empleado en resolver el ejercicio.
- setTitle: permite señalar el título del ejercicio.
- Problema: permite instanciar un objeto de tipo Problema.

C.3.5 Clase Circuito

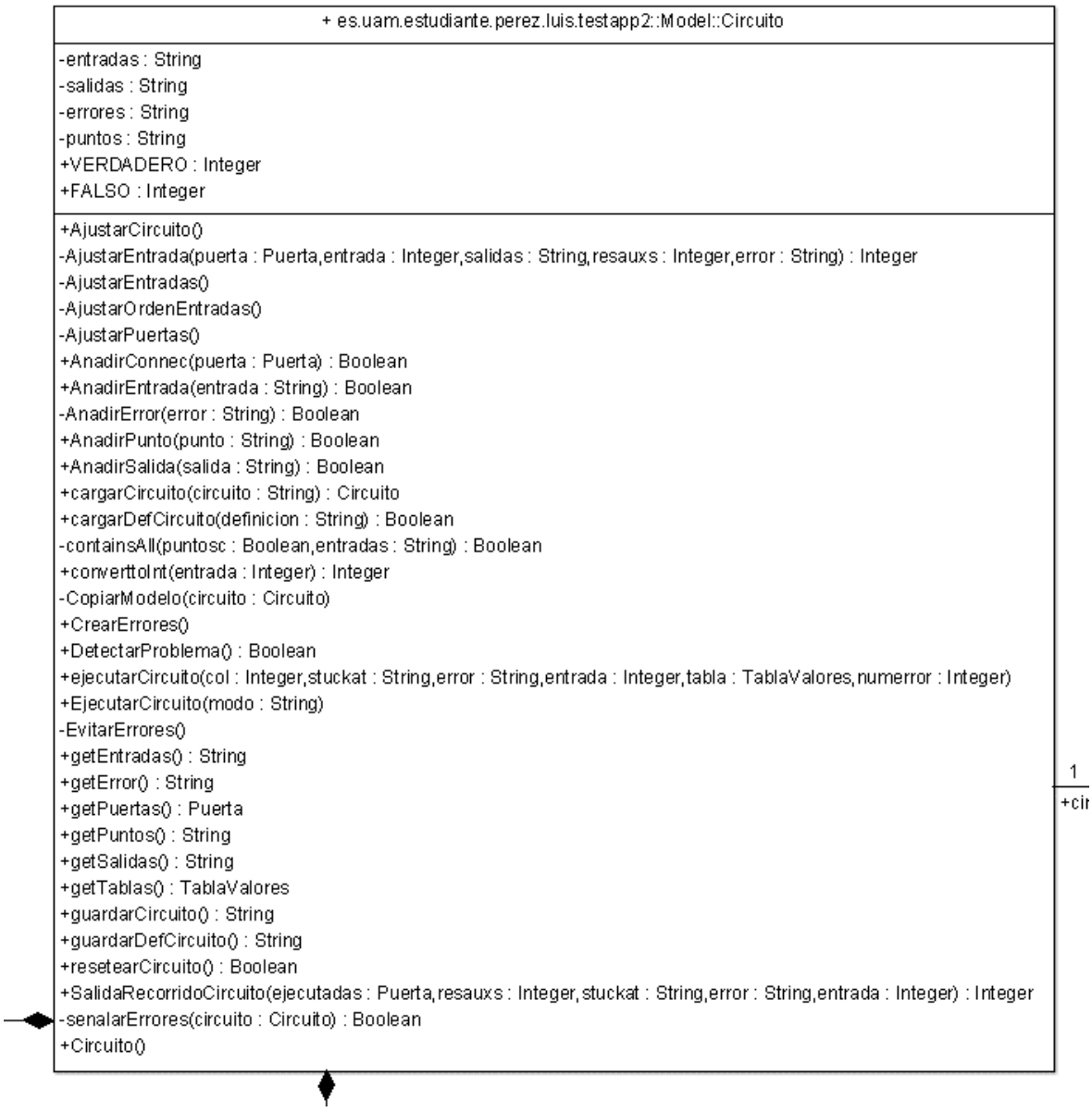

Figura C.3.5-1: Clase Circuito en Diagrama de clases

Fuente: Propia

La clase Circuito se ha diseñado para contener las puertas por las que se compone un circuito y para almacenar las diferentes salidas en función de la entrada del circuito.

Los métodos implementados aparecen en la siguiente página:

- AjustarCircuito: permite ajustar el circuito declarado para que la aplicación funcione correctamente.
- AjustarEntrada: permite ajustar los valores de entrada de una puerta del circuito en función de los valores conocidos.
- AjustarEntradas: permite ajustar las entradas del circuito declarado.
- AjustarOrdenPuertas: permite ordenar las puertas del circuito declarado.
- AjustarPuertas: permite ajustar las entradas de cada una de las puertas del circuito declarado.
- AnadirConnec: permite añadir una puerta al circuito.
- AnadirEntrada: permite añadir una entrada al circuito.
- AnadirError: permite añadir un error de tipo Stuck-at al circuito definido.
- AnadirPunto: permite señalar puntos intermedios del circuito
- AnadirSalida: permite añadir una salida al circuito.
- cargarCircuito: permite cargar un circuito a través de un String.
- cargarDefCircuito: permite cargar la definición básica de un circuito a través de un String.
- containsAll: permite comprobar si las entradas de una puerta del circuito se conoce sus valores o no.
- convertoInt: permite convertir un numero en base 2 a base 10.
- CopiarModelo: permite copiar las características de un circuito a otro.
- CrearErrores: permite señalar todos los errores posibles en el circuito.
- DetectarProblema: permite detectar si el usuario se ha equivocado o no a la hora de resolver el ejercicio.
- ejecutarCircuito: permite hallar las salidas de un circuito con unas características determinadas.
- EjecutarCircuito: permite obtener la salida del circuito dependiendo de si tiene errores o no.
- EvitarErrores: permite detectar salidas duplicadas en puertas distintas
- getEntradas: permite obtener las entradas del circuito.
- getError: permite obtener el error indicado en el circuito.
- getPuertas: permite obtener las puertas incluidas en el circuito.
- getPuntos: permite obtener los puntos intermedios del circuito.
- getSalidas: permite obtener las salidas del circuito.
- getTablas: permite obtener las tablas a rellenar del ejercicio.
- guardarCircuito: permite guardar el circuito en un String.
- guardarDefCircuito: permite guardar la definición del circuito en un String.
- resetearCircuito: permite reiniciar las tablas del ejercicio.
- SalidaRecorridoCircuito: permite obtener la salida del circuito para una determinada entrada.
- senalarErrores: permite indicar el error activo del circuito.
- Circuito: permite instanciar un objeto de tipo Circuito.

C.3.6 Clase TablaValores

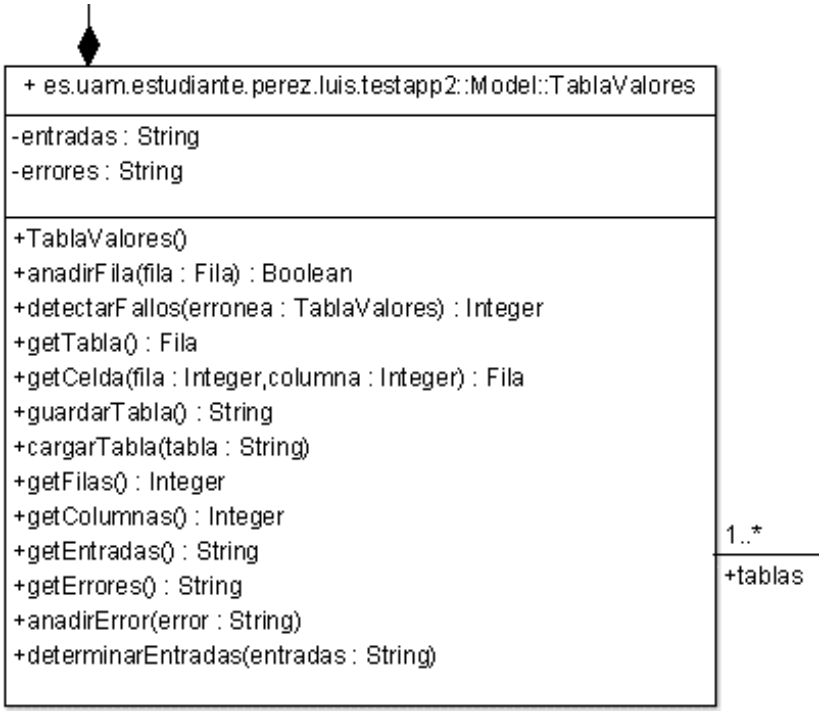

Figura C.3.6-1: Clase TablaValores en Diagrama de clases

Fuente: Propia

La clase TablaValores se ha diseñado para contener cada una de las salidas de un determinado circuito en función de la entrada.

Los métodos implementados son los siguientes:

- TablaValores: permite instanciar un objeto de tipo TablaValores.
- anadirFila: permite añadir una fila a la tabla definida del ejercicio.
- detectarFallos: permite detectar si el usuario se ha equivocado o no a la hora de resolver el ejercicio.
- getTabla: permite obtener las filas que componen la tabla.
- getCelda: permite obtener la celda de la tabla a partir de la fila y la columna.
- guardarTabla: permite guardar la tabla en un String.
- cargarTabla: permite cargar la tabla a partir de un String.
- getFilas: permite obtener el número de filas de la tabla.
- getColumnas: permite obtener el número de columnas de la tabla.
- getEntradas: permite obtener las entradas del circuito.
- getErrores: permite obtener los errores del ejercicio indicados en la tabla.
- anadirError: permite indicar un error en la tabla.
- determinarEntradas: permite indicar las entradas del circuito.

C.3.7 Clase Fila

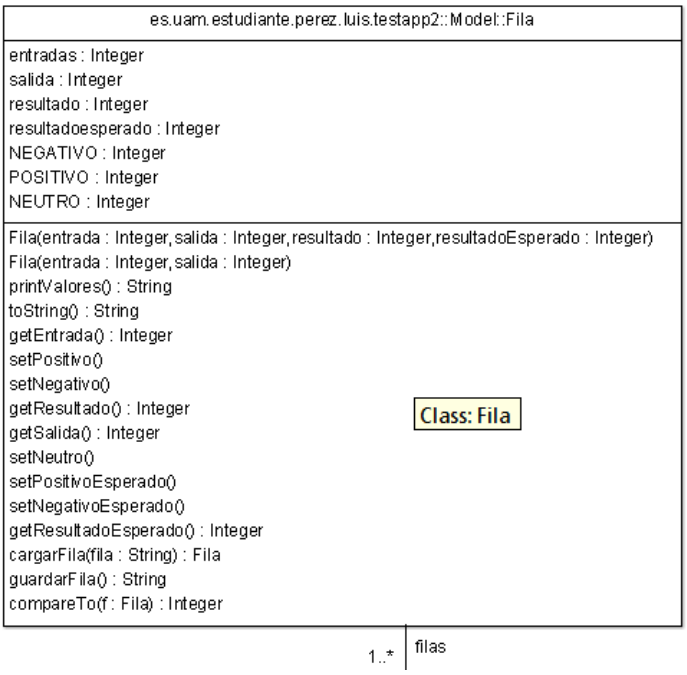

Figura C.3.7-1: Clase Fila en Diagrama de clases

Fuente: Propia

La clase Fila se ha diseñado para contener la salida; el resultado del test de Stuck-at hallado por el propio sistema y el resultado del test indicado por el alumno para una determinada entrada de un determinado circuito.

Los métodos implementados son los siguientes:

- Fila: permite instanciar un objeto de tipo Fila.
- printValores: permite convertir a String la entrada de la fila.
- toString: permite convertir a String las características de la fila.
- getEntrada: permite obtener la entrada de la fila.
- setPositivo: permite indicar como positivo el resultado del test indicado por el sistema.
- setNegativo: permite indicar como negativo el resultado del test indicado por el sistema.
- getResultado: permite obtener el resultado del test indicado por el sistema.
- getSalida: permite obtener la salida del circuito para la entrada de la fila.
- setNeutro: permite indicar como neutro el resultado del test obtenido por el sistema.
- setPositivoEsperado: permite indicar como positivo el resultado del test indicado por el usuario.
- setNegativoEsperado: permite indicar como negativo el resultado del test indicado por el usuario.
- getResultadoEsperado: permite obtener el resultado del test indicado por el usuario.
- cargarFila: permite cargar una instancia Fila a partir de un String.
- guardarFila: permite guardar una instancia Fila en un String.

- compareTo: permite comparar 2 instancias de tipo Fila.

C.3.8 Clase Abstracta Puerta

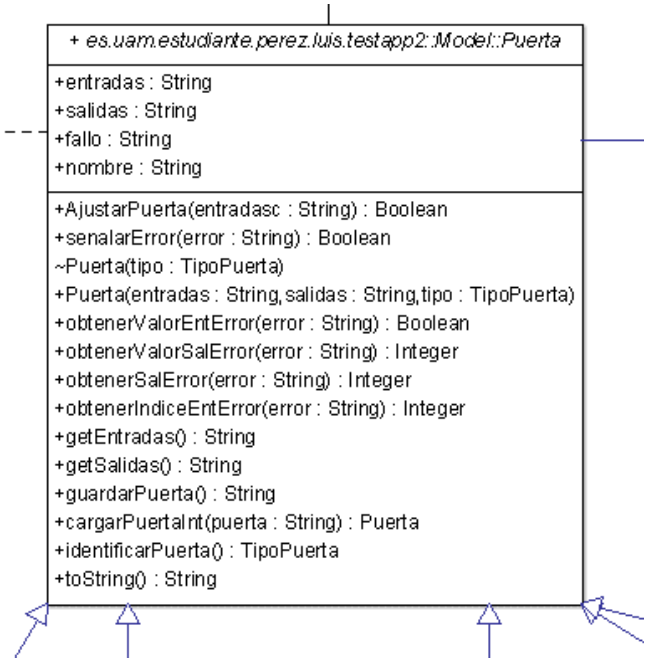

Figura C.3.8-1: Clase Abstracta Puerta en Diagrama de clases

Fuente: Propia

La clase abstracta Puerta se ha diseñado para implementar todos los métodos relativos al funcionamiento de cualquier puerta lógica.

Los métodos implementados son los siguientes:

- AjustarPuerta: permite ajustar la Puerta a un orden determinado.
- senalarError: permite indicar el error de tipo Stuck-at que contiene la puerta.
- Puerta: permite instanciar un objeto de tipo Puerta.
- obtenerValorEntError: permite obtener el valor del cable de entrada con problema de Stuck-at.
- obtenerValorSalError: permite obtener el valor del cable de salida con problema de Stuck-at.
- obtenerSalError: permite identificar si la salida que identifica el error pertenece a la salida de la puerta.
- obtenerIndiceEntError: permite identificar si el fallo identificado pertenece a la entrada de la puerta o no.
- getEntradas: permite obtener las entradas de la puerta.
- getSalidas: permite obtener las salidas de la puerta.
- guardarPuerta: permite convertir la puerta en un String que luego podrá ser cargado.

- cargarPuertaInt: permite cargar la puerta a partir de un String que contiene la información relativa a una puerta.
- identificarPuerta: permite identificar el tipo de Puerta.
- toString: permite convertir a String la información relativa a una puerta.

C.3.9 Clase FabricaRepositorioProblema

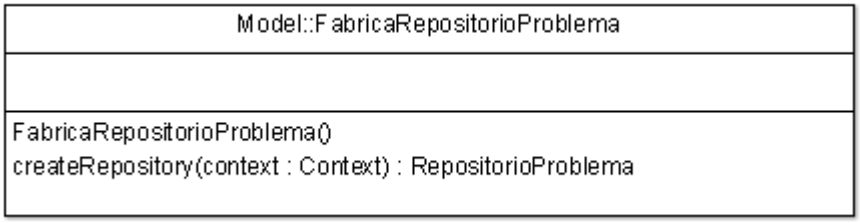

Figura C.3.9-1: Clase FabricaRepositorioProblema en Diagrama de clases

Fuente: Propia

La clase FabricaRepositorioProblema se ha diseñado para en un futuro permitir la incorporación de nuevas funcionalidades en la aplicación. También se ha diseñado para permitir acceder a la base de datos en aquellas partes de la aplicación que sea necesario.

Los métodos implementados son los siguientes:

- FabricaRepositorioProblema: permite instanciar un objeto de tipo FabricaRepositorioProblema.
- createRepository: permite obtener la base de datos necesaria para la aplicación y en caso de no existir crearla.

C.3.10 Clase PuertaAnd

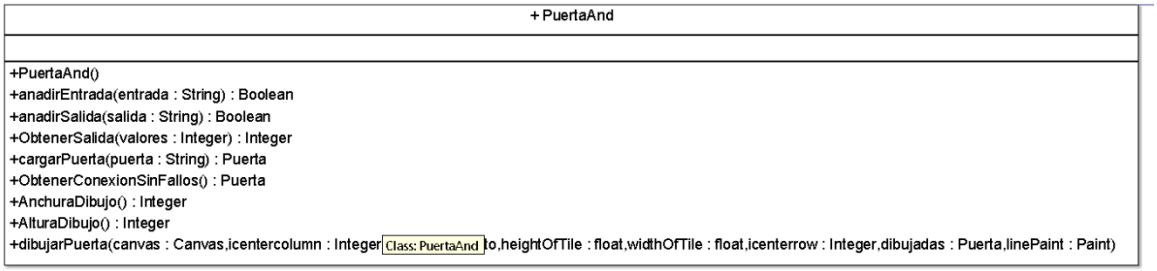

Figura C.3.10-1: Clase PuertaAnd en Diagrama de clases

Fuente: Propia

La clase PuertaAnd se ha diseñado como heredera de la clase abstracta Puerta.
La clase PuertaAnd especifica las características, el tamaño y el comportamiento de una puerta And en un circuito; implementando los métodos de la interfaz PuertaLogica que la clase abstracta Puerta no implementa.

C.3.11 Clase PuertaNand

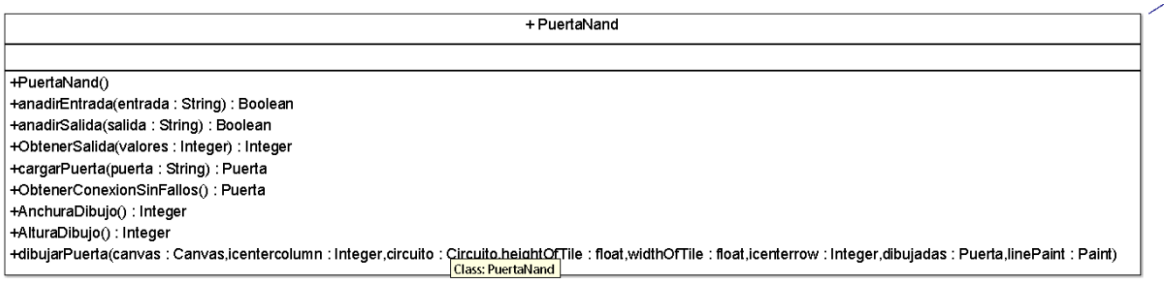

Figura C.3.11-1: Clase PuertaNand en Diagrama de clases

Fuente: Propia

La clase PuertaNand se ha diseñado como heredera de la clase abstracta Puerta.
La clase PuertaNand especifica las características, el tamaño y el comportamiento de una puerta Nand en un circuito; implementando los métodos de la interfaz PuertaLogica que la clase abstracta Puerta no implementa.

C.3.12 Clase PuertaOr

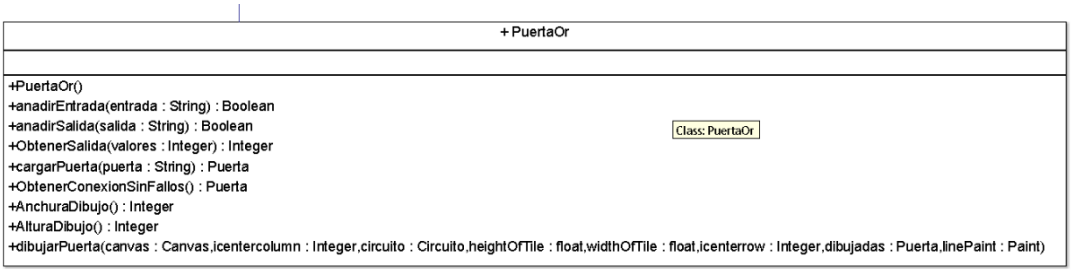

Figura C.3.12-1: Clase PuertaOr en Diagrama de clases

Fuente: Propia

La clase PuertaOr se ha diseñado como heredera de la clase abstracta Puerta.
La clase PuertaOr especifica las características, el tamaño y el comportamiento de una puerta Or en un circuito; implementando los métodos de la interfaz PuertaLogica que la clase abstracta Puerta no implementa.

C.3.13 Clase PuertaXor

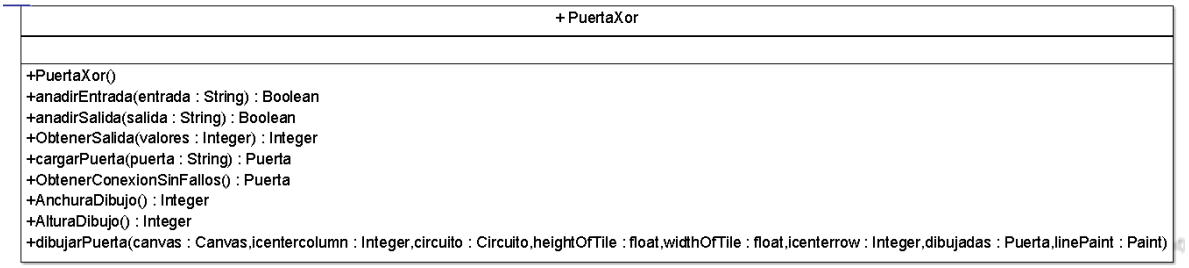

Figura C.3.13-1: Clase PuertaXor en Diagrama de clases

Fuente: Propia

La clase PuertaXor se ha diseñado como heredera de la clase abstracta Puerta.
La clase PuertaXor especifica las características, el tamaño y el comportamiento de una puerta Xor en un circuito; implementando los métodos de la interfaz PuertaLogica que la clase abstracta Puerta no implementa.

C.3.14 Clase PuertaInv

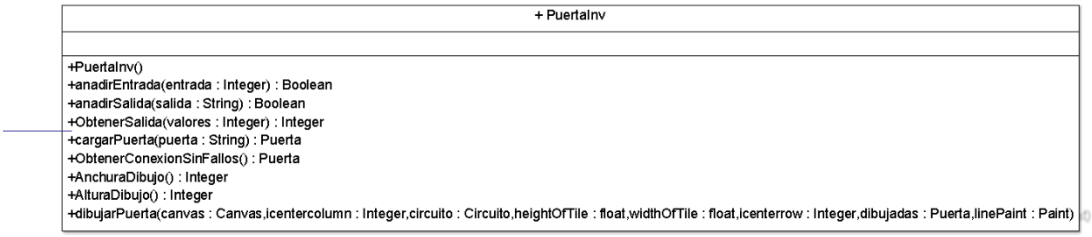

Figura C.3.14-1: Clase PuertaInv en Diagrama de clases

Fuente: Propia

La clase PuertaInv se ha diseñado como heredera de la clase abstracta Puerta.
La clase PuertaInv especifica las características, el tamaño y el comportamiento de una puerta Inv en un circuito; implementando los métodos de la interfaz PuertaLogica que la clase abstracta Puerta no implementa.

C.3.15 Clase Ejercicios

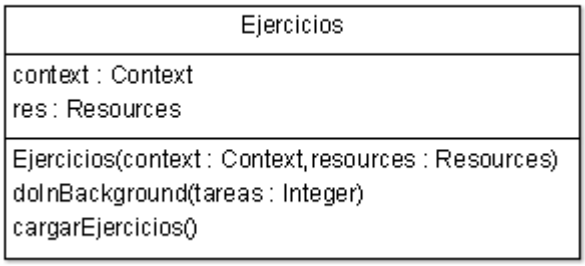

Figura C.3.15-1: Clase Ejercicios en Diagrama de clases

Fuente: Propia

La clase Ejercicios se ha diseñado para permitir la carga en segundo plano de los ejercicios que incluye la aplicación por defecto.

Los métodos implementados son los siguientes:

- Ejercicios: permite instanciar un objeto de tipo Ejercicios.
- doInBackground: permite realizar una serie de tareas en segundo plano.
- cargarEjercicios: permite cargar en la base de datos de la aplicación los ejercicios predefinidos de la aplicación.

C.3.16 Clase DB

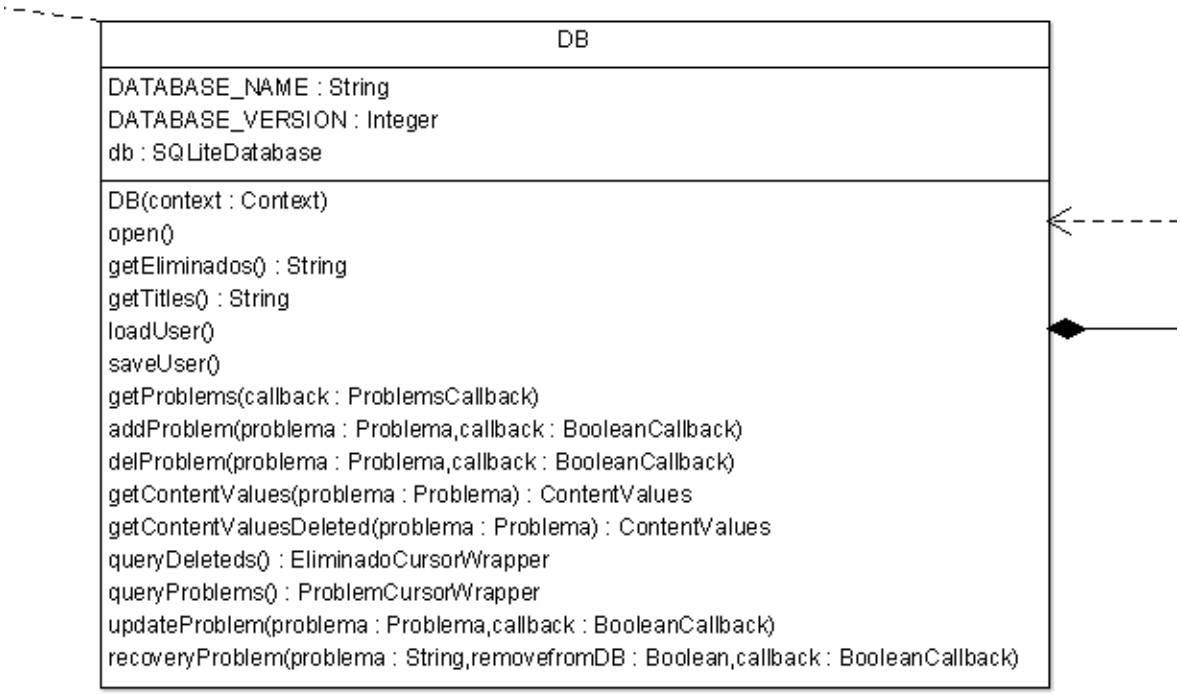

Figura C.3.16-1: Clase DB en Diagrama de clases

Fuente: Propia

La clase DB se ha definido para implementar los métodos de la interfaz RepositorioProblema y de esta forma permitir que la aplicación funcione sin necesidad de una conexión a internet.

Los métodos implementados que no pertenecen a la interfaz RepositorioProblema son:

- getContentValues: permite estructurar la información de un Problema o ejercicio para ser cargado en la base de datos.
- getContentValuesDeleted: permite estructurar la información de un Problema o ejercicio de la papelera de reciclaje para ser cargado en la base de datos.
- queryDeleteds: permite obtener de la base de datos aquellos ejercicios que han sido eliminados.
- queryProblems: permite obtener de la base de datos aquellos ejercicios que pueden ser resueltos por el usuario.

C.3.17 Clase DatabaseHelper

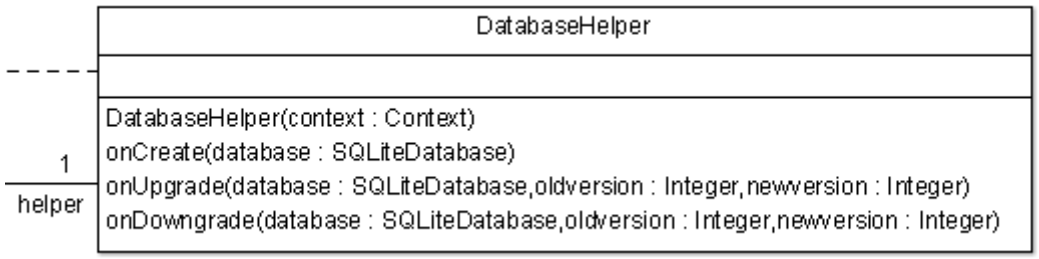

Figura C.3.17-1: Clase DatabaseHelper en Diagrama de clases

Fuente: Propia

La clase DatabaseHelper se ha definido para definir el comportamiento de la clase DB cuando esta se crea, se actualiza o se desactualiza.

Los métodos implementados son los siguientes:

- DatabaseHelper: permite instanciar un objeto de tipo DatabaseHelper.
- onCreate: permite crear la arquitectura de una base de datos de tipo SQLiteDatabase.
- onUpgrade: permite actualiza la arquitectura de una base de datos de tipo SQLiteDatabase.
- onDowngrade: permite desactualizar la arquitectura de una base de datos de tipo SQLiteDatabase.

D. Vistas de las actividades y de los fragmentos

D.1AboutActivity

La siguiente imagen muestra como se ve la actividad en la aplicación.

Figura D.1-1: Vista About

Fuente: Propia

D.2AddCircuitActivity

La siguiente imagen muestra como se ve la vista de la actividad en la aplicación.

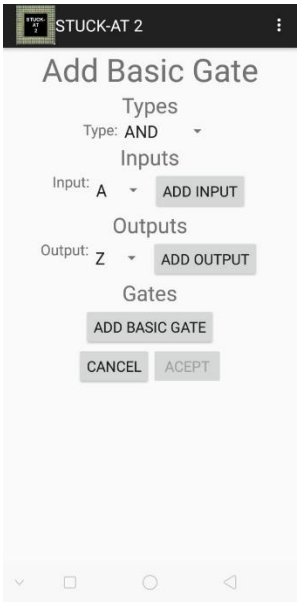

Figura D.2-1: Vista Añadir Puertas Lógicas

Fuente: Propia

D.3StatisticsActivity

La siguiente imagen muestra como se ve la vista de la actividad en la aplicación.

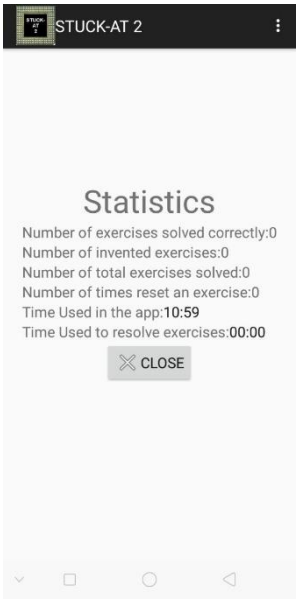

Figura D.3-1: Vista Añadir Puertas Lógicas

Fuente: Propia

D.4AddProblemaActivity

La siguiente imagen muestra como se ve la vista de la actividad en la aplicación.

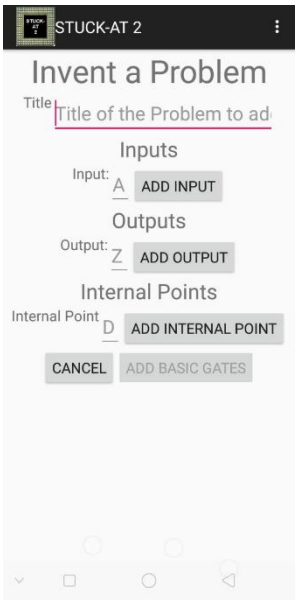

Figura D.4-1: Vista Añadir Problema

Fuente: Propia

D.5ResetActivity

La siguiente imagen muestra como se ve la vista de la actividad en la aplicación.

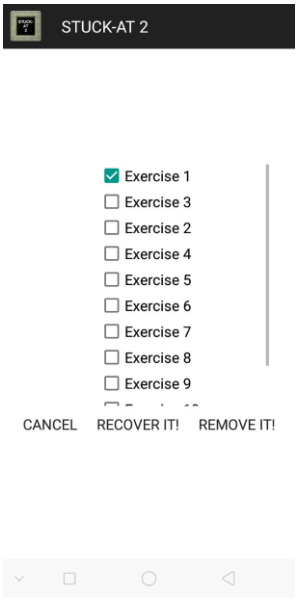

Figura D.5-1: Vista Papelera de reciclaje

Fuente: Propia

D.6SchemeActivity

La siguiente imagen muestra como se ve en el programa:

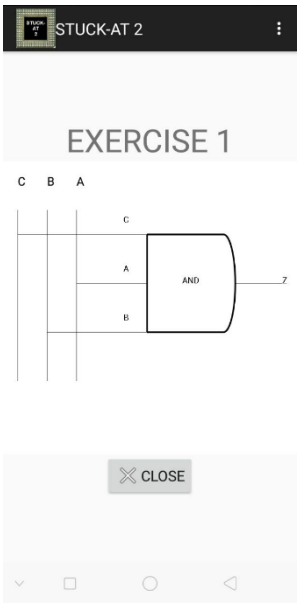

Figura D.6-1: Vista esquema

Fuente: Propia

D.7HelpActivity

La siguiente imagen muestra como se ve la vista de la actividad en la aplicación:

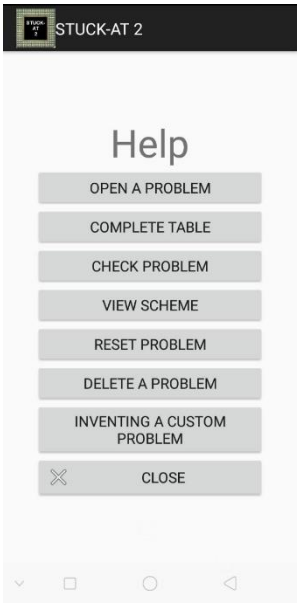

Figura D.7-1: Menú de ayuda

Fuente: Propia

D.8MainActivity

La siguiente imagen muestra como se ve el menú principal.

Figura D.8-1: Menú principal

Fuente: Propia

D.9FragmentoProblema

La siguiente imagen muestra la vista del fragmento:

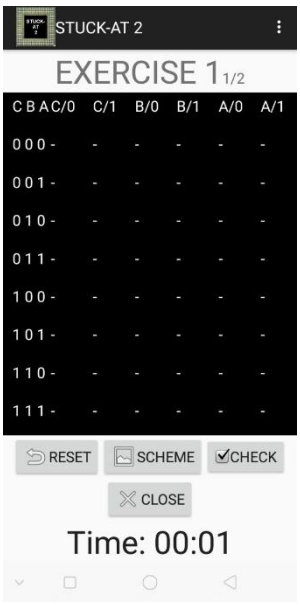

Figura D.9-1: Vista ejercicio

Fuente: Propia

D.10 RepositorioFragment

La siguiente imagen muestra como se ve el fragmento con problemas cargados:

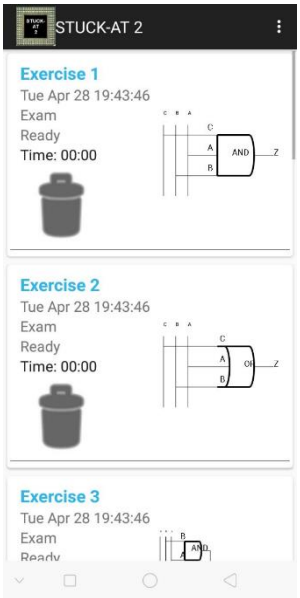

Figura D.10-1: Vista de ejercicio

Fuente: Propia

La siguiente imagen muestra como se ve el fragmento en caso de que no haya ningún ejercicio disponible.

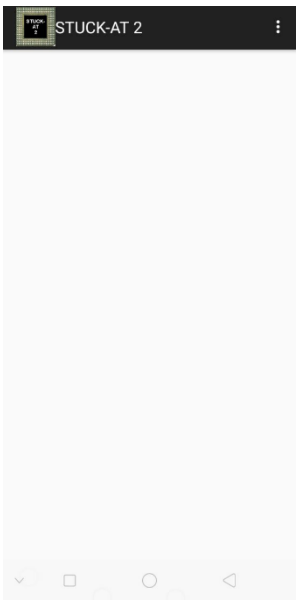

Figura D.10-2: Vista sin ejercicios

Fuente: Propia