



Universidad
Rey Juan Carlos

**ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA INFORMÁTICA**

GRADO EN INGENIERÍA DEL SOFTWARE

Curso Académico 2023/2024

Trabajo Fin de Grado

Pathfinder Puzzle:

Aplicación educativa para potenciar el pensamiento computacional en
edades tempranas



Autor: Javier Gaspariño Muñoz

Tutora: Liliana Patricia Santacruz Valencia

Agradecimientos

Gracias a mi familia y amigos por su apoyo durante el desarrollo del proyecto. Gracias a compañeros y profesores por hacer más fácil el camino. Gracias a mi tutora Liliana Patricia Santacruz Valencia por su paciencia, ayuda y orientación durante todo el proyecto. Por último también gracias a mi padre, que vio cómo empezaba la carrera y no podrá ver cómo la termino, aun siendo este una gran motivación.

Índice de Contenidos

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Estructura de la memoria	2
2. Contexto	3
2.1. STEM	3
2.2. Pensamiento Computacional	3
2.3. Relación del proyecto con el Pensamiento Computacional	6
2.4. Mecánicas del juego	7
2.5. Algoritmos y Formulas	8
3. Descripción informática	12
3.1. Metodología	12
3.1.1. Metodologías tradicionales	12
3.1.2. Metodologías ágiles	13
3.1.3. Metodología utilizada	15
3.2. Análisis de Requisitos	16
3.3. Planteamiento	19
3.4. Tecnologías y herramientas utilizadas	21
4. Pathfinder Puzzle	28
4.1. Prototipo de la aplicación	28
4.2. Diagrama de flujo de la aplicación	29
4.3. Diagrama de casos de uso de la aplicación	30
4.4. Arquitectura de la aplicación	30
4.5. Diseño e implementación	31
4.5.1. Generación de Laberintos Aleatorios	31
4.5.2. Pintado gráfico del laberinto	31
4.5.3. Pseudo-bactracking	33
4.5.4. Preguntas	35
4.5.5. Puntuaciones y Tabla de Puntuaciones	42
4.5.6. Música	47
4.5.7. Pantalla principal, Explicación del juego y Otros	51
5. Estudio de caso	53
5.1. Descripción del estudio de caso	53
5.2. Análisis de los resultados	54
6. Conclusión y Lineas futuras	59
6.1. Objetivos cumplidos	59
6.2. Conclusiones	59
6.3. Líneas de trabajo futuras	60
Bibliografía	62

Índice de Figuras

1.	<i>Ejemplo de Sudoku</i>	4
2.	<i>Ejemplo de Cubo de Rubik</i>	5
3.	<i>Ejemplo de Laberinto</i>	5
4.	<i>Juego Scratch</i>	6
5.	<i>Distancia Manhattan contra distancia Euclideana. Se pinta en verde la distancia Euclideana, en color amarillo, rojo y azul la distancia Manhattan, siendo igual en los 3</i>	9
6.	<i>Algoritmo de backtracking</i>	10
7.	<i>Algoritmo de backtracking</i>	11
8.	<i>Modelo de tipo Cascada</i>	12
9.	<i>Modelo de tipo Espiral</i>	13
10.	<i>Modelo de tipo Incremental</i>	13
11.	<i>Modelo de tipo Scrum</i>	14
12.	<i>Modelo de tipo Kanban</i>	15
13.	<i>Modelo de tipo Extreme Programming</i>	15
14.	<i>Tablero de Trello para Kanban</i>	20
15.	<i>Estructura básica de gitflow, la cuál se ha seguido</i>	21
16.	<i>Pantalla que muestra los errores que muestra el sonar antes de limpiarlo</i>	23
17.	<i>Pantalla que muestra el sonar, una vez arreglados los errores</i>	24
18.	<i>Rama con los últimos cambios subida con todos los errores arreglados</i>	24
19.	<i>Garantía de sonar de que el código está correcto</i>	24
20.	<i>Prototipo de la aplicación</i>	28
21.	<i>Diagrama de flujo de la aplicación</i>	29
22.	<i>Diagrama de casos de uso de la aplicación</i>	30
23.	<i>Arquitectura de la aplicación</i>	30
24.	<i>Laberinto in-game, con la solución óptima pintada</i>	33
25.	<i>Laberinto in-game, con la reconducción por haber realizado 3 movimientos incorrectos consecutivos</i>	35
26.	<i>Código de inicialización de la base de datos de preguntas</i>	37
27.	<i>Código de creación de la base de datos de preguntas</i>	37
28.	<i>Estructura de la base de datos de preguntas</i>	37
29.	<i>Pantalla con una pregunta de nivel medio</i>	39
30.	<i>Pantalla con una pregunta de nivel difícil</i>	41
31.	<i>Pantalla con el menú de opciones</i>	44
32.	<i>Pantalla con la opción de Seleccionar Dificultad dentro del menú de opciones</i>	44
33.	<i>Pantalla que muestra la tabla de Puntuaciones, con los datos iniciales</i>	46
34.	<i>Pantalla que muestra el control de la música en MainActivity, marcado con recuadro rosa</i>	48
35.	<i>Pantalla que muestra el control de la música en GameActivity</i>	49
36.	<i>Pantalla Principal, se marca el botón para la explicación de como jugar</i>	52
37.	<i>Resultado de la primera pregunta a niños</i>	54
38.	<i>Resultado de la segunda pregunta a niños</i>	55
39.	<i>Resultado de la tercera pregunta a niños</i>	55
40.	<i>Resultado de la cuarta pregunta a niños</i>	56
41.	<i>Resultado de la quinta pregunta a niños</i>	56
42.	<i>Resultado de la primera pregunta a docentes</i>	57

43.	<i>Resultado de la segunda pregunta a docentes</i>	57
44.	<i>Resultado de la tercera pregunta a docentes</i>	58
45.	<i>Resultado de la cuarta pregunta a docentes</i>	58

Índice de Tablas

1.	Habilidades del pensamiento computacional y relación con el proyecto . .	6
2.	Tabla de Historias de Usuario con Riesgo, Prioridad y Tipo de Requisito	16
3.	Resumen del uso de tecnologías y herramientas en el proyecto	27

Resumen

El enfoque de STEM (Science, Technology, Engineering and Mathematics) es crucial para impulsar la innovación, resolver problemas, preparar a personas para carreras del ámbito tecnológico, al que va muy enfocado, y desarrollar habilidades críticas como el pensamiento analítico y la resolución de problemas características del pensamiento computacional. Aunque el término “pensamiento computacional” puede no ser familiar para todos, desarrollarlo es crucial, ya que fortalece habilidades esenciales para diversos ámbitos de la vida diaria y profesional.

En este contexto, se ha creado la aplicación Pathfinder Puzzle para trabajar algunas de las habilidades, como (i) el reconocimiento de patrones usado en el laberinto y en algunas preguntas, (ii) el diseño de algoritmos, explicado en una de las pantallas y usado para encontrar la solución óptima, (iii) la evaluación, el pensamiento lógico y el pensamiento crítico necesarios para resolver de la mejor forma posible los laberintos y (iv) la abstracción y descomposición, necesarias para solucionar de manera más eficiente el laberinto o resolver preguntas que contienen pequeños minijuegos, y las cuales fueron sacadas de la guía Bebras de 2018 [1].

Además, se ha realizado un estudio de caso y aunque los resultados obtenidos no se pueden considerar concluyentes, debido al tamaño de la muestra, estos han servido para hacerse una idea del alcance de los objetivos propuestos en cuanto a la posibilidad de potenciar la adquisición de habilidades de pensamiento computacional.

Finalmente, este documento recoge el proceso de desarrollo del proyecto, así como las conclusiones derivadas de su realización.

1. Introducción

La tecnología es esencial en todos los descubrimientos contemporáneos y se ha vuelto omnipresente en diversos sectores sociales y profesionales, incluyendo la educación de los más jóvenes, donde aplicaciones diseñadas para potenciar el pensamiento computacional están revolucionando la forma en que los estudiantes abordan la resolución de problemas y el análisis lógico [2].

El pensamiento computacional es fundamental en la educación moderna. El concepto de pensamiento computacional se popularizó principalmente gracias al artículo de Jeanette M. Wing titulado “Computational Thinking”, publicado en marzo de 2006 en la revista “Communications of the ACM”. En este artículo, Wing argumenta que el pensamiento computacional debe ser una parte fundamental de la educación de todos, no solo de los informáticos. También comenta esto sobre el pensamiento computacional:

“El pensamiento computacional implica resolver problemas, diseñar sistemas y comprender el comportamiento humano, basándose en los conceptos fundamentales de la informática. El pensamiento computacional incluye una gama de herramientas mentales que reflejan la amplitud del campo de la informática.” [3]

Varios de los métodos más comunes y tradicionales en ciencias de la computación son [4]:

- “Conferencias”, donde un profesor explica conceptos desde el frente del aula, mientras los estudiantes escuchan y toman notas.
- Demostraciones prácticas, donde el profesor demuestra cómo resolver un problema o cómo utilizar un software específico mientras los estudiantes observan.
- Ejercicios en papel, donde estudiantes resuelven problemas de programación o lógica en papel.
- Exámenes y pruebas escritas, que se llevan a cabo mediante exámenes escritos que miden la comprensión del estudiante sobre los temas tratados en clase.

La aplicación “Pathfinder Puzzle” utiliza laberintos aleatorios y simula el comportamiento de “backtracking” para desarrollar el pensamiento computacional. Los usuarios que cometan tres errores consecutivos son devueltos al último punto correcto. Si vuelven a cometer los errores, enfrentan una pregunta de pensamiento computacional; fallar en cualquier pregunta conlleva perder la partida. Este mecanismo se repite hasta dos veces, con preguntas que van creciendo en dificultad. Este diseño promueve un aprendizaje activo y crítico, manteniendo el juego desafiante y evitando la monotonía, reflejando la evolución hacia métodos educativos más centrados en el estudiante en ciencias de la computación.

1.1. Motivación

La motivación por la que se ha elegido este tema surge por la afición por los videojuegos, por lo mucho que han aportado y por la creencia de que una manera de enseñar de forma interactiva a través de ellos es más agradable y facilita la comprensión de los temas que trata.

También se ha querido trabajar con el concepto del pensamiento computacional, en que se basa y cómo usarlo de una forma entretenida para el usuario, tal y como se hace en el juego PathFinder Puzzle con laberintos y una serie de preguntas con minijuegos sacados de la guía Bebras de 2018, la cual es una competencia internacional destinada a promover la educación en informática y el pensamiento computacional entre estudiantes de todas las edades [1].

1.2. Objetivos

El principal objetivo es:

- Potenciar el desarrollo de competencias asociadas al pensamiento computacional.

Y como objetivos específicos se han definido los siguientes:

- Crear una aplicación para dispositivos móviles que permita trabajar conceptos de pensamiento computacional en edades tempranas.
- Aplicar mecánicas del juego en el diseño de la aplicación.
- Propiciar un ambiente de interacción cómodo para el niño, con el fin de evitar el estrés.
- Fomentar la resolución de laberintos como una herramienta para la comprensión de algoritmos típicos, como es el caso de “backtracking” [5].

1.3. Estructura de la memoria

El resto de la estructura de la memoria se divide en los siguientes apartados:

- *Apartado 2. Contexto*, se explica en detalle en qué consiste STEM, el Pensamiento Computacional, cuáles son sus principales usos, la relación del proyecto con el Pensamiento Computacional, mecánicas del juego, y algoritmos y fórmulas utilizadas.
- *Apartado 3. Descripción informática*, se describe la metodología empleada en el proyecto, así como los requisitos técnicos, el planteamiento llevado a cabo y las tecnologías y herramientas utilizadas en el mismo.
- *Apartado 4. Pathfinder Puzzle*, se tratan los aspectos más estéticos del proyecto como son el prototipo de la aplicación, los diagramas tanto de flujo como de casos de uso, la arquitectura y el diseño y la implementación de este.
- *Apartado 5. Estudios de caso*, se describe el estudio de caso llevado a cabo, así como las experiencias de este y los resultados obtenidos de las pruebas con niños y docentes con tal de poner a prueba la aplicación con su “target objetivo”.
- *Apartado 6. Conclusiones y Líneas Futuras*, se examinan los objetivos propuestos para así sacar una conclusión de los resultados obtenidos y que objetivos se han cumplido, así como lo aprendido con este proyecto. También se abordan aquellas pequeñas mejoras que se podrían haber implementado, y que no se acabaron realizando por falta de tiempo.

2. Contexto

En este apartado se explica STEM, se habla del Pensamiento Computacional en profundidad, así como de la relación de este con el proyecto y también de los algoritmos y fórmulas utilizadas.

2.1. STEM

Para este proyecto se ha utilizado el enfoque de STEM (Science, Technology, Engineering and Mathematics) donde el Pensamiento computacional trabaja una parte. “Hoy en día, se ha vuelto crucial formar individuos que no solo consuman tecnología, sino que también la produzcan. Para hacer eso, es necesario crear un deseo en los estudiantes de conocer sobre ciencia, tecnología, ingeniería y matemáticas (STEM).” En 2017 se publicó un estudio en New Media Consortium indicando que la tecnología en términos generales, será más importante en los entornos educativos a partir de 2017 [6].

Debido a que el interés en la tecnología se puede potenciar de gran manera en los años escolares, es evidente la importancia de estudios y actividades en estos temas.

La integración del pensamiento computacional en el enfoque STEM es crucial para el desarrollo de estudiantes que no solo consuman tecnología, sino que también la produzcan. Este enfoque promueve habilidades esenciales como la descomposición de problemas, el diseño algorítmico y el reconocimiento de patrones, preparando a los estudiantes para aplicar estas competencias en cualquier disciplina de STEM y en otros ámbitos. Al desarrollar estas habilidades, se facilita la transición de los alumnos de ser usuarios pasivos a que sean innovadores activos en tecnología [7].

2.2. Pensamiento Computacional

La mayoría de lugares en los que se habla del pensamiento computacional, sitúan el origen del concepto en la columna de opinión de Jeanette Wing en marzo de 2006 en la revista Communications of the ACM, donde, como se ha comentado previamente en la introducción comentaba:

“El pensamiento computacional implica resolver problemas, diseñar sistemas y comprender el comportamiento humano, basándose en los conceptos fundamentales de la informática. El pensamiento computacional incluye una gama de herramientas mentales que reflejan la amplitud del campo de la informática.” [3]

De esto se puede entender lo que definía como “herramientas mentales” como: “un conjunto de habilidades y destrezas habituales en los profesionales de las ciencias de la computación, pero que todos los seres humanos deberían poseer y utilizar para resolver problemas, diseñar sistemas y, sorprendentemente, comprender el comportamiento.” [8]

Por ello, el pensamiento computacional debería formar parte en la educación de todos, empezando primero con los más jóvenes. Ha habido varios avances en cuanto a educación para llevar a cabo esto, de los cuales nos centraremos en los ocurridos en España.

Por ejemplo, podemos hablar de lo que sucedió en Enero de 2016, cuando el 22 de Abril “ la Fundación Española para la Ciencia y la Tecnología (FECYT) del Ministerio de Economía y Competitividad del Gobierno de España, Google y Everis publicaron

un informe titulado Educación en Ciencias de la Computación en España 2015 con el objetivo de “analizar la situación actual de la enseñanza de Ciencias de la Computación (CC) en España para niños y niñas de entre 6 y 16 años” y “propone(r) una serie de recomendaciones para la introducción, expansión y mejora de la enseñanza de esta materia en el corto y medio plazo”. Una de las recomendaciones finales del informe fue “Establecer un marco de consenso entre los agentes clave sobre la hoja de ruta a seguir para introducir las CC en el currículo educativo, tanto en Educación Primaria como Secundaria” ” [8].

También cabe mencionar que, en enero de 2018, el Ministerio de Educación, Cultura y Deporte de España, a través del Instituto Nacional de Tecnologías Educativas y de Formación del Profesorado (INTEF) y el Centro Nacional de Innovación e Investigación Educativa (CNIIE), publicó un informe titulado ”Programación, robótica y pensamiento computacional en el aula. Situación en España”. El informe analiza la situación actual de estos temas en el programa educativo nacional y en las comunidades autónomas, destacando diversas iniciativas lideradas por universidades, empresas y la sociedad civil. Se mencionan contribuciones de entidades tecnológicas y educativas, y se incluye una encuesta a 351 docentes, quienes valoraron moderadamente (7,34 sobre 10) el impacto de la enseñanza de programación y robótica en el aprendizaje de los alumnos. A pesar de la escasa presencia curricular formal de estos contenidos, algunas comunidades autónomas han comenzado a integrarlos opcionalmente en la educación secundaria. Sin embargo, solo un tercio de los docentes encuestados había recibido formación específica en estos campos, siendo la mayoría autodidactas. El informe concluye resaltando la necesidad de más investigación en estas áreas en España (Ministerio de Educación, Cultura y Deporte, 2018) [8].

Existen muchos juegos muy conocidos que sirven para el desarrollo de la lógica en la educación de los más jóvenes, como pueden ser: [9].

- *Sudokus*, que consisten en rellenar cuadrículas de 9x9, con números del 1 al 9, sin repetirlos en una misma fila, columna o bloque. Cómo se puede observar en la Figura 1.

Figura 1
Ejemplo de Sudoku

5	3			7				
6			1	9	5			
	9	8			1		6	
			8					3
			4			8		
7				2				
	6						2	8
				3				
				1			7	4

Nota: Tomado de Ng, J. (2023, 6 diciembre). How to play sudoku — 2024 Step-by-Step Guide for Beginners - AhaSlides. AhaSlides.
<https://ahaslides.com/es/blog/how-to-play-sudoku/>.

- *Cubos de Rubik*, que son rompecabezas tridimensionales que consisten en alinear los colores de cada una de sus caras. Cómo se puede observar en la Figura 2.

Figura 2

Ejemplo de Cubo de Rubik

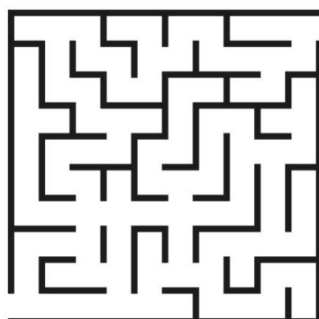


*Nota: Tomado de EL CUBO DE RUBIK – Alberto Pérez Núñez. (s. f.).
<https://albertopereznunez.com.ar/el-cubo-de-rubik/>.*

- *Laberintos*, que consisten en encontrar la salida del laberinto sin equivocarse de camino. Cómo se puede observar en la Figura 3.

Figura 3

Ejemplo de Laberinto



*Nota: Tomado de St, T. (s.f.). Vector labyrinth. Maze or Labyrinth. Vector illustration.
iStock. <https://www.istockphoto.com/es/vector/laberinto-de-vector-maze-o-labyrinth-vector-gm990469834-268465158>.*

También existen distintos juegos para trabajar el Pensamiento Computacional (PC) en la educación, como son los aquí mencionados [10], aunque el juego más conocido para el trabajo del pensamiento computacional (PC) es Scratch, el cual es un lenguaje de programación con una interfaz sencilla que permite a los jóvenes crear historias digitales, juegos y animaciones. Además Scratch promueve el pensamiento computacional y las habilidades en resolución de problemas; enseñanza y aprendizaje creativos, autoexpresión y colaboración. Este juego se puede observar en la Figura 4 [11].

Figura 4
Juego Scratch



Nota: Tomado de Twinkl. (s.f.). Scratch - Twinkl Teaching Wiki. Recuperado de <https://www.twinkl.es/teaching-wiki/scratch>.

2.3. Relación del proyecto con el Pensamiento Computacional

En el concepto conocido como Pensamiento Computacional (PC) se pueden identificar distintas habilidades. A continuación, en la Tabla 1, se detallan cuáles son estas habilidades, en qué consisten y su relación con el proyecto.

Tabla 1: Habilidades del pensamiento computacional y relación con el proyecto

Habilidad	En qué consiste	Relación
Descomposición	Capacidad de descomponer problemas complejos en problemas más pequeños.	Capacidad del usuario para descomponer los problemas de pensamiento computacional en partes más pequeñas resolviéndolo de mejor manera.
Reconocimiento de patrones	Habilidad para Identificar patrones, tendencias o regularidades dentro de los problemas.	Capacidad del usuario para reconocer patrones en el laberinto y en muchas de las preguntas propuestas.
Abstracción	Habilidad para abstraer y centrarse en la información importante, ignorando los detalles irrelevantes.	Capacidad del usuario para saber ignorar los caminos incorrectos del laberinto, así como las soluciones incorrectas de las preguntas.
Diseño de algoritmos	Capacidad para desarrollar pasos y reglas para resolver problemas y completar tareas.	Capacidad del usuario para poder diseñar el camino óptimo a la solución.
Pensamiento lógico	Capacidad para aplicar un razonamiento lógico con el fin de analizar problemas, hacer predicciones y llevar a cabo la solución de problemas mediante la lógica.	Capacidad del usuario para deducir el camino correcto así como deducir la respuesta correcta a las preguntas.

Habilidad	En qué consiste	Relación
Pensamiento crítico	Capacidad para cuestionar constantemente el proceso y los resultados.	Capacidad del usuario para cuestionar una y otra vez la solución del laberinto antes de llegar al fallo, y antes de responder una pregunta.
Evaluación	Capacidad de analizar y evaluar diversas soluciones a un problema, determinando cuál es la más efectiva o eficiente.	Capacidad del usuario para escoger la solución óptima del laberinto entre todas las soluciones posibles.

2.4. Mecánicas del juego

Un videojuego es un sistema complejo definido por mecánicas y reglas que los jugadores siguen para interactuar con el sistema. En este apartado se explican las “mecánicas del juego” que sigue la aplicación. Estas “mecánicas del juego” son: “cualquier acción realizada por el jugador que modifique el game state, es decir, la posición y características concretas de todos los objetos y entornos de un juego en un momento preciso en el tiempo.” Los juegos además tienen una característica muy importante, son interactivos, y debido a ello se modifica el “game state” [12].

La estructura de las “mecánicas del juego” de la aplicación son las siguientes:

- Propósito del juego:
Resolver el laberinto buscando la solución óptima.
- Procedimiento para efectuar la acción:
Usar las flechas de dirección para moverse por la solución óptima.
- Reglas que gobiernan la acción:
El jugador debe moverse por la solución óptima; si no lo hace, es penalizado.
- Número de jugadores:
El número de jugadores por partida es de solamente uno.
- Rol de los jugadores:
El rol del jugador es explorar el laberinto, encontrando la mejor solución posible usando la estrategia que considere adecuada. También deberá solucionar las preguntas en caso de necesitarlo.
- Resultado o recompensa:
Hay distintas recompensas en el juego. La principal son los puntos obtenidos al completar un laberinto con la solución óptima, los cuales varían en función del intento y la dificultad del laberinto. La otra recompensa, es la que se obtiene al acertar una pregunta, por la cual se gana otro intento para seguir jugando, aunque habiendo perdido ciertos puntos por haber fallado en el laberinto.

- Habilidades requeridas:

Este juego requiere de las habilidades ya mencionadas en el apartado anterior, las cuales son las que trabaja el pensamiento computacional.

- Patrones de interacción:

El patrón de interacción principal se basa en la exploración guiada con penalizaciones. El jugador usa las flechas de dirección para moverse y, si comete errores de forma repetida, es desafiado con preguntas de tipo trivial que incrementan en dificultad. Los aciertos le permiten continuar, mientras que los errores llevan a perder el juego.

- Actividad física necesaria para jugar:

La única actividad física necesaria para jugar es pulsar las flechas de dirección que se encuentran en la aplicación del móvil.

- Equipamiento preciso:

Para jugar solo se necesita un dispositivo Android con la aplicación instalada.

2.5. Algoritmos y Formulas

- Cálculo de la distancia mediante la distancia Manhattan:

En el proyecto se simula el comportamiento del algoritmo de “backtracking”, que se explicará más abajo. Para esto, se hace el cálculo de la distancia de Manhattan para saber si te alejas tres movimientos de la solución óptima, comprobando si te acercas o te alejas más de la solución óptima que en el movimiento anterior, verificando así si estás retrocediendo en el movimiento hacia la solución correcta.

La distancia de Manhattan, o geometría del taxista, se describe como: “La distancia entre dos puntos medida a lo largo de los ejes en ángulo recto” [13]. Su fórmula es:

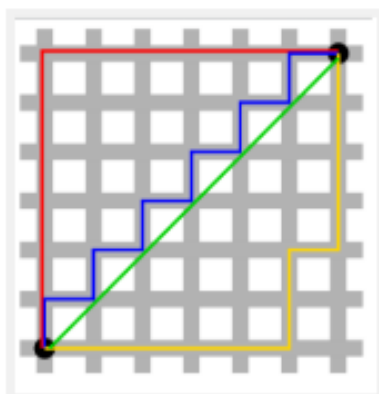
$$distancia = \sum |x - y|$$

Ejemplo: si se quiere calcular la distancia entre dos puntos, (x_1, y_1) y (x_2, y_2) entonces el resultado de la distancia es $(|x_1 - x_2| + |y_1 - y_2|)$ [13] [14].

Aunque la distancia que se suele considerar como la más óptima de un punto A a un punto B siempre se suele considerar como una línea recta de A a B, la cual se calcula con la distancia Euclideana [15], hay situaciones cotidianas como las dadas en el cálculo de la distancia de un punto A a B caminando entre puntos alejados de una ciudad como Manhattan, en la que no se puede trazar una línea recta ya que habría que atravesar edificios. Un ejemplo de esto es la Figura 5.

Figura 5

Distancia Manhattan contra distancia Euclídeana. Se pinta en verde la distancia Euclídeana, en color amarillo, rojo y azul la distancia Manhattan, siendo igual en los 3



*Nota: Tomado de Distancia Manhattan — Interactive Chaos. (s.f.).
<https://interactivechaos.com/es/manual/tutorial-de-machine-learning/distancia-manhattan>.*

- Algoritmo de backtracking:

“Es una estrategia algorítmica que busca todas las posibles soluciones dado un conjunto de variables inicial para encontrar el resultado definido por el problema.

La técnica de “Backtracking” se apoya en el uso de la recursividad para la búsqueda exhaustiva de todas las combinaciones posibles.”

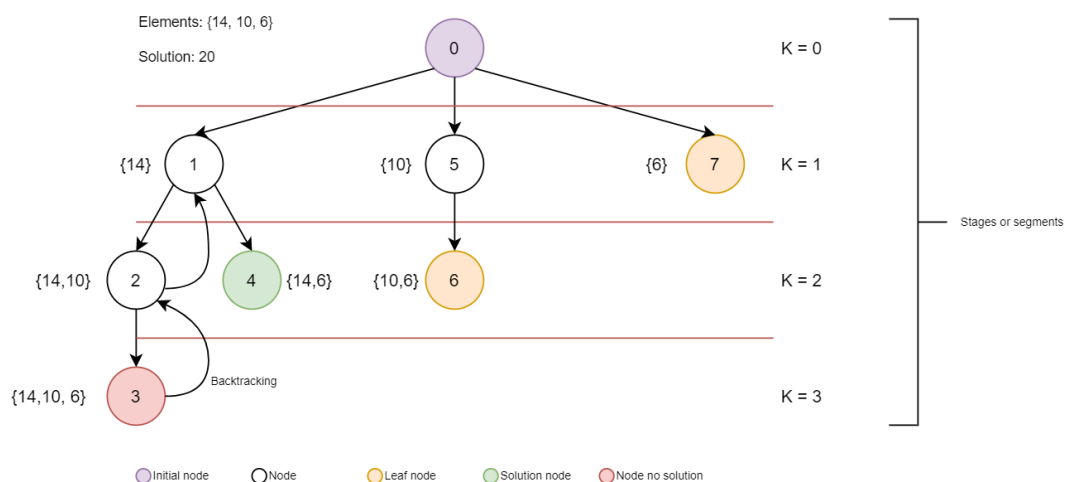
“Además es una técnica algorítmica para hacer una búsqueda exhaustiva y sistemática por todas las configuraciones posibles del espacio de búsqueda del problema.

Se suele aplicar en la resolución de un gran número de problemas, muy especialmente en los de decisión y optimización.”

- “Problemas de decisión: Búsqueda de las soluciones que satisfacen ciertas restricciones.”
- “Problemas de optimización: Búsqueda de la mejor solución en base a una función objetivo.”

Un ejemplo del algoritmo de backtracking es el que se muestra en la Figura 6.

Figura 6
Algoritmo de backtracking



Nota: Tomado de JJ Peleato 2.4 Backtracking — Programación, refactoriza tu mente. (s.f.). <https://docs.jjpeleato.com/algorithmia/backtracking>.

La forma en la que actúa este algoritmo consiste en elegir un camino del conjunto de caminos posibles en cada etapa del proceso de resolución, y si esta no funciona porque no nos lleva a la solución, entonces la búsqueda vuelve al punto donde se realizó la elección, e intenta con otro valor. Cuando se acaban todas las posibles elecciones en un punto, la búsqueda retrocede al último punto de elección y continúa. Si no quedan más puntos de elección, la búsqueda termina.

“Se puede explicar el algoritmo de forma matemática de esta manera:

- El conjunto de soluciones se expresa en tuplas, donde cada una es el valor de la solución.

$$\mathbf{s} = (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n) \tag{1}$$

- El conjunto parcial de soluciones será aquel en que se encuentre en cierto nivel K :

$$\mathbf{s}_p = (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k) \rightarrow \mathbf{K} \leq \mathbf{n} \tag{2}$$

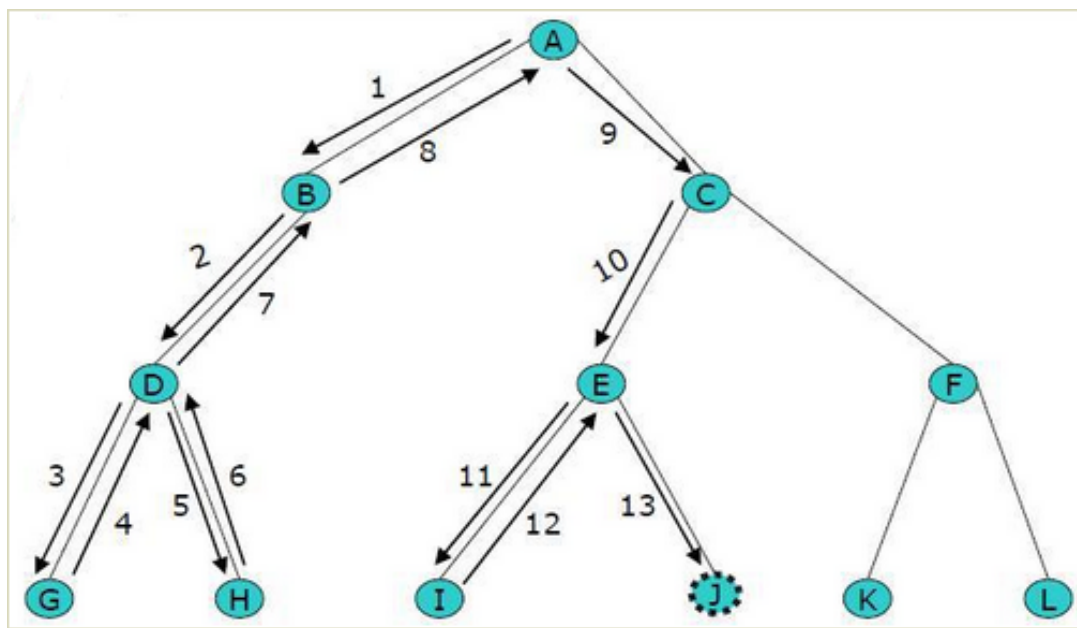
- Si se puede añadir un elemento más, la solución avanza a otro nivel ($K+1$).
- Si no existe ningún valor, se retrocede al valor ($K-1$).
- Se continúa hasta que una solución parcial sea una solución al problema o hasta que no queden más posibilidades a probar.

El resultado es equivalente a hacer una búsqueda en profundidad en el árbol de soluciones. Sin embargo, este árbol es implícito, no se almacena en ningún lugar.”

[5]

- Algoritmo de búsqueda en profundidad (DFS):
 Es un algoritmo de búsqueda no informada en la que se recorren todos los nodos de un grafo de manera ordenada, pero no simultánea. Consiste en ir expandiendo los nodos que va localizando, de forma recurrente, en un camino concreto. Si no quedan más nodos que visitar en ese camino, se regresa (se hace “backtracking”) repitiendo el proceso para cada hermano del nodo que se estaba procesando. Un ejemplo de este algoritmo es el de la Figura 7 [16].

Figura 7
 Algoritmo de backtracking



Nota: Tomado de Cevallos, K. (2015, 30 mayo). Resolver problemas mediante búsquedas: Búsqueda de Soluciones. Inteligencia Artificial II. <https://inteligenciartificial2kc.wordpress.com/2015/04/22/resolver-problemas-mediante-busquedas-busqueda-de-soluciones/>.

3. Descripción informática

3.1. Metodología

Una metodología de software es un conjunto estructurado de prácticas, procedimientos y técnicas utilizadas en el proceso de desarrollo de software. Su propósito es guiar y mejorar la eficiencia, calidad y éxito del proyecto de software. Las metodologías de software proporcionan un marco de trabajo que ayuda a planificar, gestionar, ejecutar y controlar el desarrollo de software, asegurando que se sigan los mejores estándares y prácticas en cada etapa del ciclo de vida del software.

Las metodologías más usadas en la ingeniería del software se dividen principalmente en dos categorías: las metodologías tradicionales y las metodologías ágiles [17].

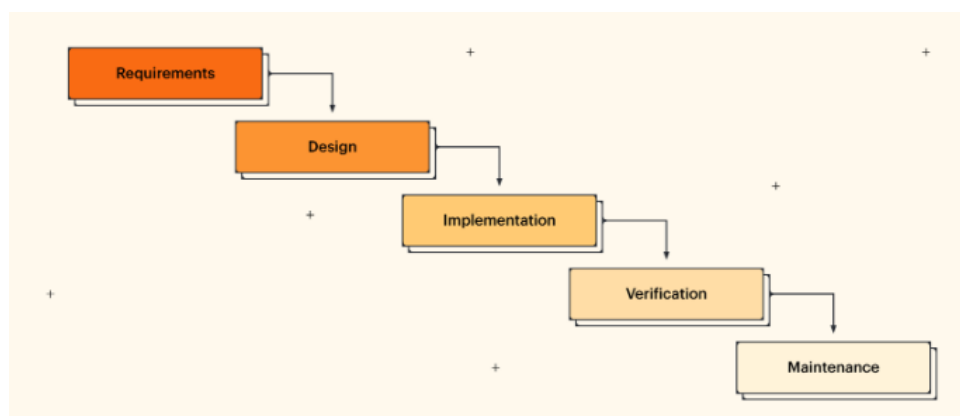
3.1.1. Metodologías tradicionales

Este tipo de metodologías son poco flexibles y, además, no suelen permitir realizar cambios. También son lineales, lo que quiere decir que no permiten pasar a una siguiente etapa sin terminar la anterior, ni tampoco volver a una etapa anterior una vez se ha pasado a la siguiente etapa. Cabe destacar los siguientes tipos:

- *Waterfall o Cascada*, es una metodología con un enfoque secuencial, donde cada fase del ciclo de vida del desarrollo de software debe completarse antes de que comience el siguiente. Recibe este nombre ya que su flujo de trabajo fluye de arriba hacia abajo entre cada etapa, asemejándose a una cascada. Es rígido y difícil de adaptarse a cambios una vez que una fase está completada. Un ejemplo de esta es el de la Figura 8.

Figura 8

Modelo de tipo Cascada



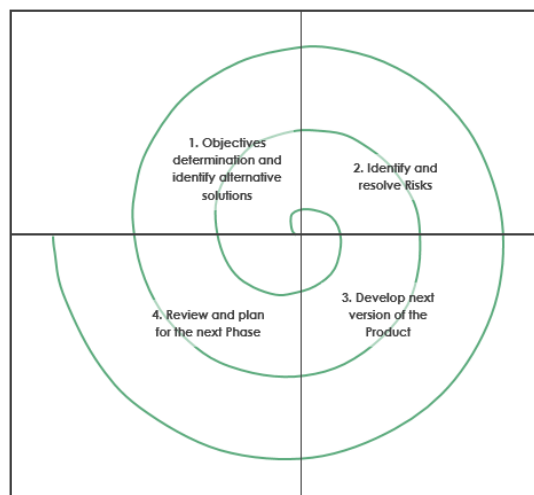
Nota: Tomado de Qué te permite hacer (y qué no) la metodología de cascada para la gestión de proyectos. (2018, 5 septiembre).

<https://www.lucidchart.com/blog/es/metodologia-gestion-proyectos-cascada>.

- *Espiral*, combina elementos del modelo en Cascada, pero añadiendo un nuevo concepto, el análisis de riesgos. Tiene cuatro etapas que se desarrollan en ciclos, por los cuales recibe su nombre. Las etapas son, Planificación, Análisis de Riesgos, desarrollo e implementación y evaluación. Este modelo es más flexible que el cascada

y SI permite cambios. Fue definido por primera vez por Barry Boehm en 1986. Un ejemplo de esta es el de la Figura 9 [18].

Figura 9
Modelo de tipo Espiral



Nota: Tomado de Vpvera. (2022, 24 febrero). Descripción general del ciclo de vida del desarrollo de software (SDLC) - Cybermedio. Cybermedio. <https://www.cybermedian.com/es/overview-of-software-development-lifecycle-sdlc/>.

- *Incremental*, es una metodología con un enfoque en el cual producto final se va construyendo de manera incremental. Se van haciendo entregas de pequeñas partes funcionales haciendo un incremento en cada etapa. Además es más flexible que las demás metodologías tradicionales. Un ejemplo de esta es el de la Figura 10.

Figura 10
Modelo de tipo Incremental

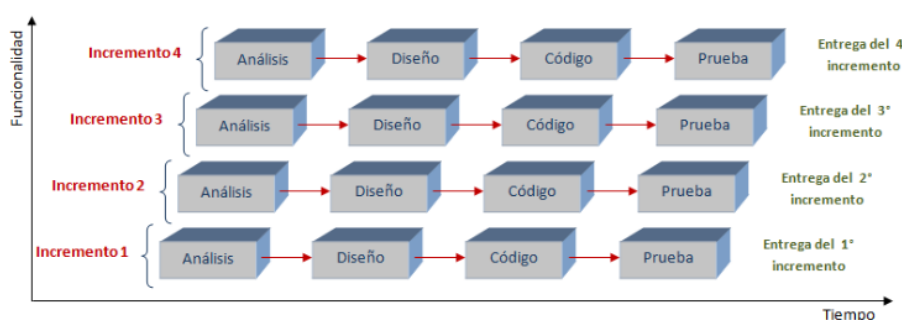


Figura 1: El Modelo Incremental

Nota: Tomado de De Informaticametodologias, V. T. L. E. (2017, 11 septiembre). Metodología incremental. Informatica I. <https://informaticametodologias.wordpress.com/2017/09/11/metodologia-incremental/>.

3.1.2. Metodologías ágiles

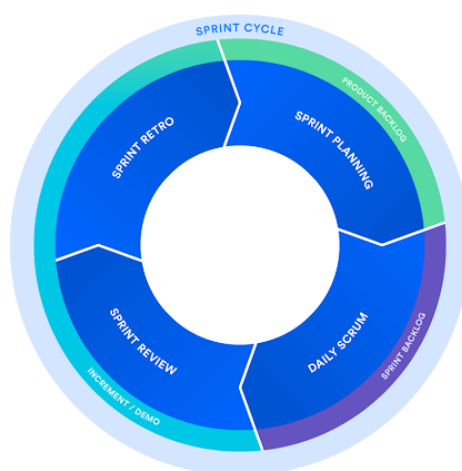
Este tipo de metodologías son las más utilizadas debido a que son muy flexibles y ágiles, como su propio nombre indica. Permiten adaptar el software a los cambios

necesarios que vayan surgiendo a lo largo del proyecto.

- *Scrum*, es una metodología con un enfoque incremental, en la cual se divide el trabajo en sprints cortos de 2 a 4 semanas, donde en cada uno se propone un objetivo claro y se entrega un incremento de software funcional al final de cada iteración. Cada iteración se divide en las siguientes etapas: Planificación del Sprint (Sprint Planning), Reunion Diaria (Daily Scrum), demostración de resultados (Sprint Review), retrospectiva del Sprint (Sprint Retrospective). Los roles principales son Product Owner, Scrum Master y el equipo de desarrollo. Un ejemplo de esta es el de la Figura 11.

Figura 11

Modelo de tipo Scrum



Nota: Tomado de Atlassian. (s.f.). *¿Qué es scrum? [+ Cómo empezar]* — Atlassian.
<https://www.atlassian.com/es/agile/scrum>.

- *Kanban*, es un método de gestión visual de proyectos inventado por “Toyota”. Se utiliza un tablero para gestionar el flujo de trabajo, donde se usan tres columnas llamadas: Por hacer, en curso y Hecho, que son rellenadas con tarjetas que identifican las tareas. Se hacen entregas continuas sin usar sprints y se trabaja primero con las tareas más prioritarias. Un ejemplo de este es el de la Figura 12.

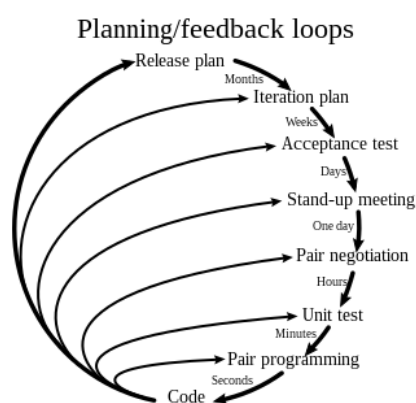
Figura 12
Modelo de tipo Kanban



Nota: Tomado de Colombo, D. (2021, 16 julio). Primero lo primero: Cómo ordenar el trabajo y aumentar tu productividad con el método Kanban. Por Daniel Colombo. Daniel Colombo. <https://www.danielcolombo.com/primero-lo-primero-como-ordenar-el-trabajo-y-aumentar-tu-productividad-con-el-metodo-kanban-por-daniel-colombo/>.

- *Extreme Programming (XP)*, es una metodología con un fuerte énfasis en el trabajo en equipo, en la calidad del software y en la capacidad de respuesta a los cambios. Las prácticas más comunes en esta metodología son: Pair Programming, Desarrollo Dirigido por Pruebas(TDD), diseño sencillo, refactorización, integración continua y pequeñas liberaciones de trabajo de manera frecuente. Se promueve el feedback con el cliente, al que se le da también mucha importancia. Un ejemplo de esta es el de la Figura 13.

Figura 13
Modelo de tipo Extreme Programming



Nota: Tomado de Wells, D., & Wells, D. (s.f.). Introducing extreme programming. <http://www.extremeprogramming.org/introduction.html>.

3.1.3. Metodología utilizada

Durante el desarrollo de la aplicación, se utilizó un enfoque que combinaba el desarrollo incremental y los principios de Kanban. El proyecto fue dividido en tareas manejables que incluían funcionalidades completas, partes de funcionalidades, cambios de interfaz y corrección de errores. Se utilizó un tablero Kanban para visualizar y gestionar el flujo

de trabajo, permitiendo que el estado de cada tarea fuera claramente visible y que se pudieran hacer adaptaciones rápidas a las necesidades cambiantes del proyecto. Se iban poniendo todas las tareas en la columna de “Por Hacer” del tablero Kanban, y se iba dando prioridad a las principales, más importantes o de las que dependían otras tareas. Estas tareas con más prioridad eran las primeras que se movían a la columna de “En Proceso”, para así realizarlas, una vez acabadas, se movían a la columna “Hecho”.

El valor fue agregado al sistema con cada incremento, ya fuera a través de nuevas funcionalidades, mejoras en la interfaz de usuario o la solución de problemas existentes. Estos incrementos se iban subiendo a la herramienta de control de versiones Github, para así llevar un orden y un seguimiento de lo que se iba realizando. Este enfoque permitió asegurar una entrega continua y eficiente, integrando y probando cada parte del proyecto de manera regular. La combinación de desarrollo incremental y Kanban facilitó la incorporación de feedback y ajustes rápidos, garantizando que un producto final de alta calidad cumpliera con los requisitos del usuario.

3.2. Análisis de Requisitos

Se ha realizado el análisis de requisitos de la aplicación, donde se han identificado y clasificado los requisitos funcionales y no funcionales. A continuación, en la Tabla 2, se presentan las historias de usuario que describen estos requisitos:

Tabla 2: Tabla de Historias de Usuario con Riesgo, Prioridad y Tipo de Requisito

ID	Historia de Usuario	Prioridad	Criterios de Aceptación	Riesgo	Tipo
1	Como usuario quiero tener un menú de inicio intuitivo y fácil de usar.	Media	El usuario puede acceder al menú de inicio al iniciar la app o al acabar la partida.	Medio	RF
2	Como usuario quiero que exista una explicación de como jugar y esta aparezca la primera vez que se usa la aplicación o al pulsar en el botón correspondiente de la pantalla principal para entender fácilmente cómo funciona el juego.	Media	La aplicación debe mostrar una explicación de como jugar, con los conceptos básicos del juego, el flujo que sigue y como moverse por el laberinto, si es la primera vez que se inicia o si se pulsa el botón desde el menú principal.	Medio	RF
3	Como usuario, quiero poder iniciar una partida con un nombre y una dificultad.	Alta	El usuario puede iniciar una partida introduciendo un nombre y seleccionando una dificultad.	Medio	RF

ID	Historia de Usuario	Prioridad	Criterios de Aceptación	Riesgo	Tipo
4	Como usuario, quiero poder poner música de fondo o quitarla, para adaptar mi experiencia de juego.	Baja	El usuario puede poner o quitar la música desde el menú de inicio de una manera intuitiva.	Bajo	RF
5	Como usuario, quiero ver la tabla de puntuaciones	Media	El usuario puede consultar la tabla de puntuaciones antes de iniciar una partida.	Medio	RF
6	Como usuario, quiero ver distintos laberintos en cada partida que cambien con la dificultad.	Alta	El sistema genera automáticamente laberintos aleatorios en función de la dificultad.	Alto	RF
7	Como usuario, quiero que exista un menú fácil de usar durante las partidas para acceder a varias opciones de control del juego.	Alta	El menú es accesible en cualquier momento durante la partida y tiene una interfaz intuitiva y fácil de manejar.	Medio	RF
8	Como usuario, quiero tener la opción de reiniciar el nivel actual desde el menú de las partidas para empezar el nivel de nuevo si me quedo atascado.	Alta	El usuario puede seleccionar la opción de reiniciar nivel en cualquier momento de la partida para reiniciar el nivel en caso de necesitarlo.	Alto	RF
9	Como usuario, quiero poder seleccionar el nivel de dificultad desde el menú para ajustar de nuevo el reto del juego durante la partida.	Alta	El usuario puede seleccionar entre las opciones de dificultad (fácil, medio, difícil) en el menú, y el juego se adaptara a ella.	Alto	RF
10	Como usuario, quiero poder consultar la solución óptima en el menú de la partida para saber como encontrar la solución si me quedo atascado.	Alta	El usuario puede consultar la solución óptima en cualquier momento desde el menú de una manera sencilla.	Alto	RF

ID	Historia de Usuario	Prioridad	Criterios de Aceptación	Riesgo	Tipo
11	Como usuario, quiero poder ajustar el volumen del juego desde el menú para controlar el sonido según mis necesidades.	Media	El usuario puede acceder a los controles de volumen en el menú para aumentar, disminuir, parar o reproducir la música.	Medio	RF
12	Como usuario, quiero poder moverme por el laberinto de una manera cómoda e intuitiva utilizando flechas que aparecen en la pantalla para navegar fácilmente por este.	Alta	El usuario puede moverse por el laberinto de una manera cómoda con las flechas que aparecen en pantalla de una manera bastante visible.	Alto	RF
13	Como usuario, quiero que no se me permita desviarme más de tres movimientos de la solución óptima para mantenerme en el camino correcto del laberinto.	Alta	El sistema calcula la distancia del usuario a la solución óptima, y si esta a más de tres casillas devuelve al usuario al último punto de la solución óptima visitado.	Alto	RF
14	Como usuario, quiero al fallar y volver atrás por primera vez, salga una explicación de lo que ocurre y que significa para poder entender el concepto con el que fui redirigido.	Media	El sistema debe mostrar una explicación cuando un usuario se desvía tres movimientos seguidos por primera vez explicando que este concepto se puede reconocer como “backtracking”.	Medio	RF
15	Como usuario, quiero que aparezcan preguntas de dificultad media al cometer tres movimientos incorrectos seguidos por segunda vez para aumentar el reto y mi aprendizaje.	Alta	El sistema debe mostrar una pregunta de pensamiento computacional de dificultad media al cometer tres movimientos incorrectos seguidos por segunda vez, debiendo responder alguna opción.	Alto	RF

ID	Historia de Usuario	Prioridad	Criterios de Aceptación	Riesgo	Tipo
16	Como usuario, quiero que aparezcan preguntas de dificultad difícil al cometer tres movimientos incorrectos seguidos por tercera vez para aumentar el reto y mi aprendizaje.	Alta	El sistema debe mostrar una pregunta de pensamiento computacional de dificultad difícil al cometer tres movimientos incorrectos seguidos por tercera vez, debiendo responder alguna opción.	Alto	RF
17	Como usuario, quiero que la partida termine si se fallan las preguntas de pensamiento computacional o si se consigue solucionar el laberinto de manera correcta para saber claramente cuándo el juego ha terminado.	Alta	El sistema debe detectar si un usuario falla una pregunta o ha solucionado el laberinto, volviendo en cualquiera de estos casos a la pantalla principal, dando la partida por finalizada.	Medio	RF
18	Como usuario, quiero que existan unas puntuaciones en función de la dificultad, si se ha solucionado el laberinto, y cuántos intentos se han utilizado en solucionarlo para poder evaluar mi rendimiento y compararlo con otros intentos.	Media	El sistema debe calcular la puntuación en función de la dificultad seleccionada, si se ha solucionado el laberinto y el número de intentos necesarios en una misma partida para resolverlo.	Alto	RF
19	Como usuario, quiero que los botones, los menús y demás elementos de las interfaces sean bonitos, intuitivos y fáciles de usar para mejorar mi experiencia de usuario.	Media	El sistema debe incorporar botones, menús y demás elementos de la interfaz con un diseño visualmente atractivo con colores, fuentes y estilos consistentes y agradables a la vista.	Bajo	RNF

3.3. Planteamiento

Como se ha explicado anteriormente, se ha aplicado el método de visualización y organización de proyectos “*Kanban*” combinada con la metodología tradicional de tipo “*Incremental*”.

Para implementar y visualizar el seguimiento del método visual de gestión de pro-

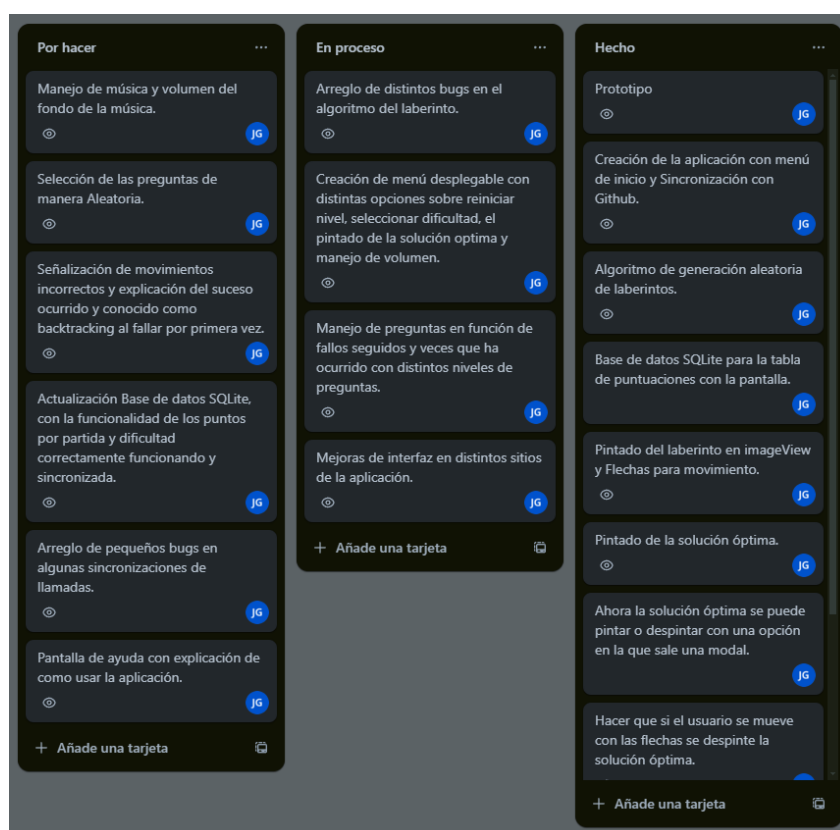
yectos Kanban, existen distintas herramientas muy útiles que se pueden usar. Algunas de estas herramientas son Trello y Miro. Trello es “una herramienta visual que permite a los equipos gestionar cualquier tipo de proyecto y flujo de trabajo, así como supervisar tareas.” Utiliza el paradigma de tablero Kanban para organizar tareas [19]. Miro es “una aplicación para desarrollar flujos de trabajo en equipo de forma remota a través de una pizarra virtual infinita” [20]. Además de hacer un tablero Kanban, se pueden hacer muchos mas tipos de “boards” adicionales.

En este caso, se ha elegido Trello debido a su simplicidad y adecuación para el trabajo individual. Aunque Miro es extremadamente útil y versátil, es más adecuado para equipos y la gestión de múltiples tableros. Dado que este proyecto es individual y solo se requiere el uso de un tablero Kanban, Trello ha sido la opción preferida.

El tablero Kanban en Trello se ha configurado como se puede observar en la Figura 14. Este tablero es público, por lo que se puede consultar si es necesario [21].

Figura 14

Tablero de Trello para Kanban



Nota: Imagen de elaboración propia.

Para la metodología incremental, se ha utilizado Github, donde se han ido subiendo las tareas que se iban completando. Estas tareas podían ser: una funcionalidad completa, parte de una funcionalidad, cambios gráficos o soluciones de bugs. Estas tareas normalmente equivalían a las historias de usuario propuestas. Para estas subidas de tareas, se ha trabajado con un semi-control de flujo de Git, donde habían dos ramas principales: la rama *master* actuando como producción y la rama *develop* para el desarrollo.

Lo ideal habría sido implementar un control de flujo de Git más completo, con ramas adicionales como *feature* para nuevas funcionalidades y mejoras de interfaz, y *bugfix*

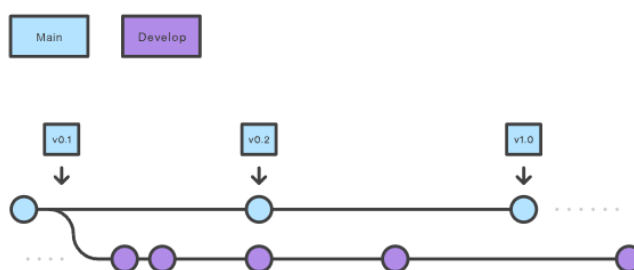
para la corrección de errores. Estas ramas se integrarían en *develop* y posteriormente en *master*. Sin embargo, dado que solo hubo un par de correcciones de errores, se consideró innecesario crear una rama *bugfix* por separado.

Además del control de tareas con Kanban, donde se iban apuntando las tareas por hacer, en curso o realizadas, y Github donde se iban subiendo estas tareas, se ha ido llevando un registro de las tareas importantes, anotando que se subió y cuándo. Aunque estos detalles pueden observarse también en Github, se ha creído conveniente esto para las tareas importantes para un mejor control y organización de las tareas más significativas. Si se quisiera ver en detalle todas las tareas subidas y cuando fueron subidas, se podría consultar Github igualmente.

En la Figura 15 se muestra el giflow básico que se ha seguido, con solo ramas de master y develop:

Figura 15

Estructura básica de gitflow, la cuál se ha seguido



Nota: Tomado de Atlassian. (s.f.). Flujo de trabajo de Gitflow — Atlassian Git Tutorial. <https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow>.

3.4. Tecnologías y herramientas utilizadas

- **AndroidStudio** [22]

Es un IDE (Entorno de Desarrollo Integrado) oficial para el desarrollo de aplicaciones en Android. Fue creado y distribuido por Google y usa IntelliJ IDEA como base. En este entorno se puede desarrollar en Java o Kotlin.

Se ha usado este IDE para realizar la aplicación del proyecto y, aunque lo más recomendable es usar Kotlin, se ha utilizado Java, que también es muy popular, ya que es un lenguaje muy utilizado en general, fuera de Android Studio.

- **Github** [23]

Es una plataforma de desarrollo colaborativo y de gestión de versiones basada en Git [24]. Fue fundada en 2008 y es ampliamente utilizada por desarrolladores y equipos de software para colaborar en proyectos de programación, gestionar el código fuente y rastrear cambios y versiones, entre otras funciones.

Se ha utilizado Github como la herramienta principal para el control de versiones y gestión del código, facilitando el seguimiento de los cambios y la organización del trabajo. Como se ha comentado anteriormente, gracias a esta herramienta se ha seguido una metodología incremental, subiendo tareas o partes de tareas que se iban realizando.

- **Firebase** [25]

Es una plataforma de desarrollo de aplicaciones móviles y web proporcionada por Google. Ofrece una variedad de herramientas y servicios para ayudar a los desarrolladores a crear experiencias atractivas sin preocuparse por la infraestructura interna. Firebase cuenta con distintos servicios de base de datos. En estos servicios se manejan datos NoSQL en la nube, ya que son altamente escalables y fáciles de usar. En concreto nos centraremos en Realtime Database, donde se almacenan los datos en formato JSON, ya que son datos muy fáciles de gestionar de este modo.

Se ha utilizado Firebase con el servicio de Realtime Database como API para las preguntas de pensamiento computacional, ya que se guardan en la nube los datos en formato JSON, siendo así fáciles de acceder desde la app, fácilmente manipulables por si se quieren añadir o eliminar preguntas y fáciles de entender.

- **SQLite** [26]

Es una biblioteca de Software que proporciona un sistema de gestión de base de datos relacional embebido. Se integra directamente en la aplicación que lo utiliza, haciendo que los datos sean locales y no en la nube, pero más rápidos de acceder y de usar, haciendo que su uso sea mas ligero. Los datos se almacenan en un unico archivo, facilitando su portabilidad.

Se ha utilizado para la tabla de puntuaciones de la aplicación, debido a su fluidez, su fácil creación y su alta compatibilidad con Android Studio. Esto hacía que las consultas a la base de datos requeridas para la tabla de Puntuaciones fueran muy rápidas, lo cual era un requisito necesario para esta tabla.

- **Trello** [27]

“Es una herramienta flexible para la gestión del trabajo, con la que los equipos pueden diseñar planes, colaborar en proyectos, organizar flujos de trabajo y hacer un seguimiento del progreso de una manera visual, productiva y gratificante.” Utiliza el paradigma de tablero Kanban para organizar tareas y flujos de trabajo. Fue creada por Fog Creek Software en 2011, aunque más tarde, en 2017, fue comprado por Atlassian. Es muy utilizado por equipos e individuos para la gestión de proyectos de todo tipo, desde tareas personales hasta grandes proyectos colaborativos.

Se ha utilizado para la creación del tablero Kanban, llevando asi el seguimiento de las tareas realizadas, en curso o por realizar, y siguiendo con ello la el método visual de gestión de proyectos Kanban explicada anteriormente.

- **Mockflow** [28]

Es una herramienta en línea para la creación de wireframes, prototipos y diagramas de interfaces de usuario tanto para aplicaciones móviles como para aplicaciones web. La interfaz permite “Drag and Drop”, siendo muy fácil de usar. Además, los componentes se pueden personalizar fácilmente.

Se ha utilizado para la creación del prototipo de la aplicación, debido a su interfaz fácil de usar y sus útiles herramientas de libre acceso para usuarios, que son ideales para la creación de prototipos de aplicaciones móviles, como ha sido en el caso de esta aplicación.

- **Drawio** [29]

Es una herramienta en línea para la creación de diagramas y gráficos, como diagra-

mas de flujo, diagrama de casos de uso, arquitectura, etc. Es una herramienta muy fácil de usar que se suele usar en educación, ingeniería y diseño.

Se ha utilizado para la creación de los distintos diagramas de esta memoria.

- **Sonar Cloud** [30]

Es una plataforma en línea que proporciona herramientas para el análisis de la calidad del código y la gestión de la deuda técnica. Fue desarrollado por SonarSource y se utiliza para mantener el código limpio y mantenible, así como seguro y eficaz.

Se ha utilizado para analizar el código de la aplicación en busca de problemas de seguridad, vulnerabilidades, code smells, o en general malas prácticas.

- Cuando se ha utilizado Sonar en el código se han descubierto los errores que se muestran en la Figura 16.

Figura 16

Pantalla que muestra los errores que muestra el sonar antes de limpiarlo

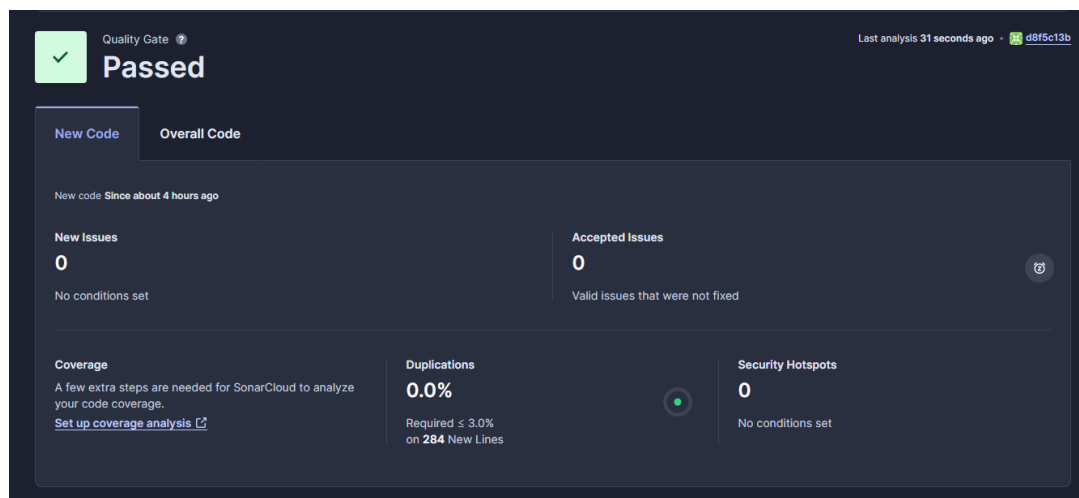


Nota: Imagen de elaboración propia.

- Una vez ya se habían arreglado los errores, la pantalla de Sonar aparecía como se puede observar en la Figura 17.

Figura 17

Pantalla que muestra el sonar, una vez arreglados los errores

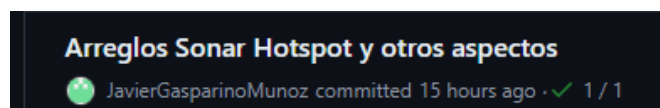


Nota: Imagen de elaboración propia.

Cuando se subió la rama que contenía los últimos arreglos del Sonar, Github mostraba lo que se puede ver en la Figura 18, debido a que al estar conectado a Sonar, cuando se sube una rama, Sonar analiza el código buscando errores y los marca. Si estos errores están solucionados, se muestra el tick verde que se puede observar, el cual significa que el código no tiene errores detectados. El tick verde el que se muestra en la Figura 19.

Figura 18

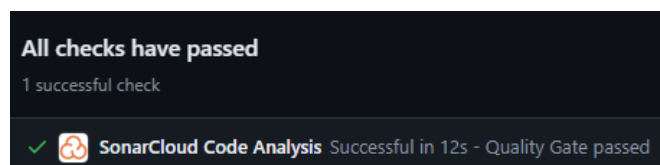
Rama con los últimos cambios subida con todos los errores arreglados



Nota: Imagen de elaboración propia.

Figura 19

Garantía de sonar de que el código está correcto



Nota: Imagen de elaboración propia.

- **DeepL [31]**

Es un servicio de traducción automática que utiliza técnicas avanzadas de Deep Learning para proporcionar traducciones de alta calidad y más naturales en comparación con otros servicios tradicionales de traducción. Fue desarrollado y lanzado por la empresa alemana DeepL GmbH en 2010.

Se ha utilizado para la traducción de distintos literales de la aplicación, al igual que para distintos conceptos expuestos en la memoria.

- **ILoveIMG** [32]

Es una herramienta en línea versátil y fácil de usar para la edición y manipulación de imágenes. Ofrece varias funciones útiles que facilitan la compresión, redimensionamiento, recorte, conversión y edición de imágenes. Es parte del conjunto de herramientas “I Love”. [33]

Se ha utilizado para adaptar varias de las imágenes utilizadas tanto en la aplicación como en esta memoria.

- **Pixelcut** [34]

Pixelcut es una herramienta versátil y fácil de usar para la creación y edición de imágenes y diseños gráficos. Debido a sus múltiples funcionalidades, facilita entre otras cosas mejorar la calidad de las imágenes, aumentando su resolución.

Se ha utilizado para mejorar la calidad de varias de las imágenes utilizadas tanto en la aplicación como en esta memoria. En concreto, se ha utilizado esta funcionalidad. [35]

- **Removebg** [36]

Es una herramienta en línea especializada en la eliminación automática de fondos de imágenes. Utiliza IA (Inteligencia Artificial) para identificar el sujeto principal y eliminar así el fondo.

Se ha utilizado esta herramienta para eliminar el fondo de algunas de las imágenes utilizadas tanto en la aplicación como en esta memoria. Aunque varias de las herramientas previamente mencionadas contaban con esta característica, se ha decidido hacer uso de esta debido a su facilidad de uso, eficiencia y alta velocidad.

- **Google Forms** [37]

Es una herramienta gratuita de encuestas y cuestionarios desarrollada por Google. Permite crear encuestas y formularios en línea de manera rápida y sencilla, facilitando la recolección y el análisis de datos.

Se ha utilizado para hacer una encuesta de satisfacción de la aplicación a distintos niños de Primaria de edad entre 8 y 10 años. Y a docentes que impartieran clases a niños en este rango de edad. Esta encuesta se encuentra en el apartado de estudio de caso, al igual que el estudio realizado a través de esta.

- **Google Drive** [38]

Es un servicio de almacenamiento en la nube desarrollado por Google. Permite almacenar y sincronizar archivos en línea, además de compartirlos con otras personas.

Se ha utilizado para almacenar la apk de la aplicación, pudiendo obtener un enlace para compartirla.

- **QR Code Generator** [39]

Es una herramienta que permite crear códigos QR mediante una URL. Tiene distintas versiones; en la gratuita, el usuario tiene 14 días de prueba.

Se ha utilizado para obtener un QR con el cual los usuarios pueden descargar la aplicación. Para ello, solo se ha necesitado registrarse y utilizar la URL obtenida de la apk almacenada en Google Drive.

- **Overleaf** [40]

Overleaf es una plataforma de edición y colaboración en línea para la creación de documentos científicos y técnicos que utiliza LaTeX. Es muy utilizado por científicos, estudiantes e ingenieros. [41].

Se ha utilizado como editor para el desarrollo de esta memoria, desarrollando de esta manera la memoria del TFG en LaTeX.

A continuación, en la Tabla 3, se muestra el resumen de las tecnologías utilizadas y su función dentro del proyecto:

Tabla 3: Resumen del uso de tecnologías y herramientas en el proyecto

Tecnologías y herramientas	Uso en el proyecto
Android Studio	Se ha utilizado para crear el código de la aplicación.
Github	Se ha utilizado para el control de versiones de la aplicación.
Firebase y SQLite	Se han utilizado para crear la base de datos para las preguntas y la base de datos para las puntuaciones respectivamente.
Trello	Se ha utilizado para la creación del tablero Kanban.
MockFlow y Drawio	Se han utilizado para la creación del prototipo y los diagramas de flujo respectivamente.
Sonar Cloud	Se ha utilizado para analizar el código en busca de errores, vulnerabilidades, code smells y otros problemas de calidad, con el fin de mejorar el código y asegurar la calidad de este,
Deep L	Se ha utilizado para la traducción de distintas partes del proyecto.
PixelCut, I Love IMG y Removebg	Se han utilizado para la modificación de las distintas imágenes utilizadas.
Google Forms	Se ha utilizado para la creación de las encuestas de satisfacción para niños y docentes.
Google Drive y Qr Code Generator	Se han utilizado para almacenar la apk de la aplicación y para la creación del QR a partir de este.
Overleaf	Se ha utilizado para la creación de esta memoria.

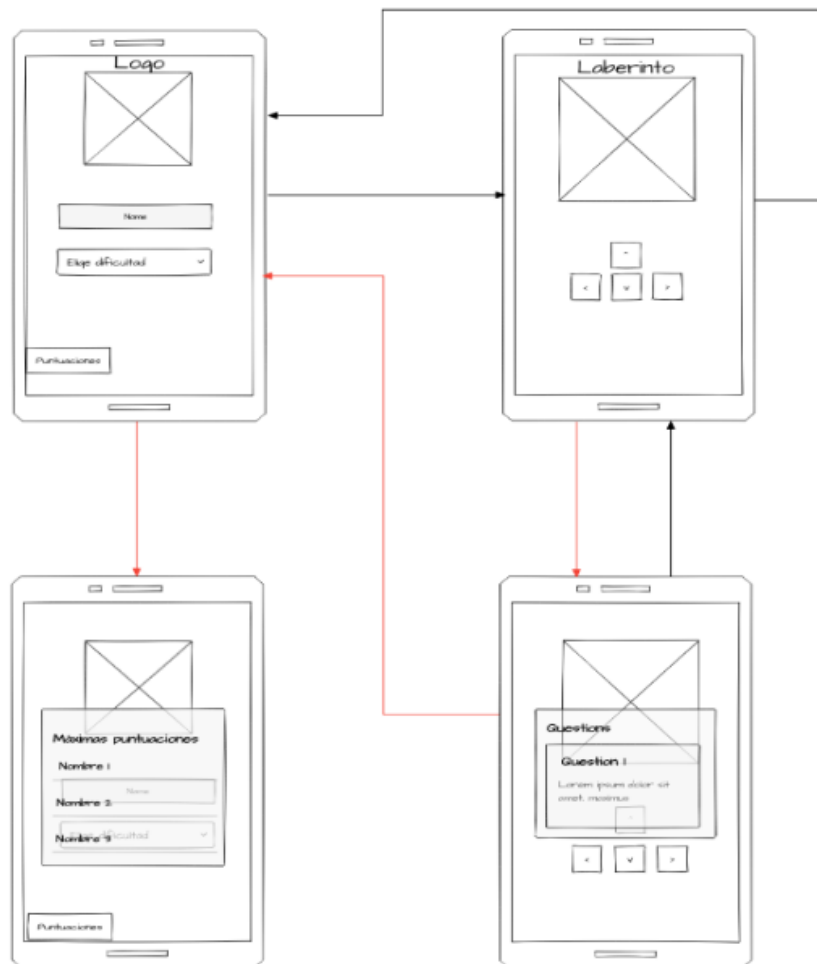
4. Pathfinder Puzzle

4.1. Prototipo de la aplicación

La Figura 20 representa el prototipo de la aplicación:

Figura 20

Prototipo de la aplicación



Nota: Imagen de elaboración propia.

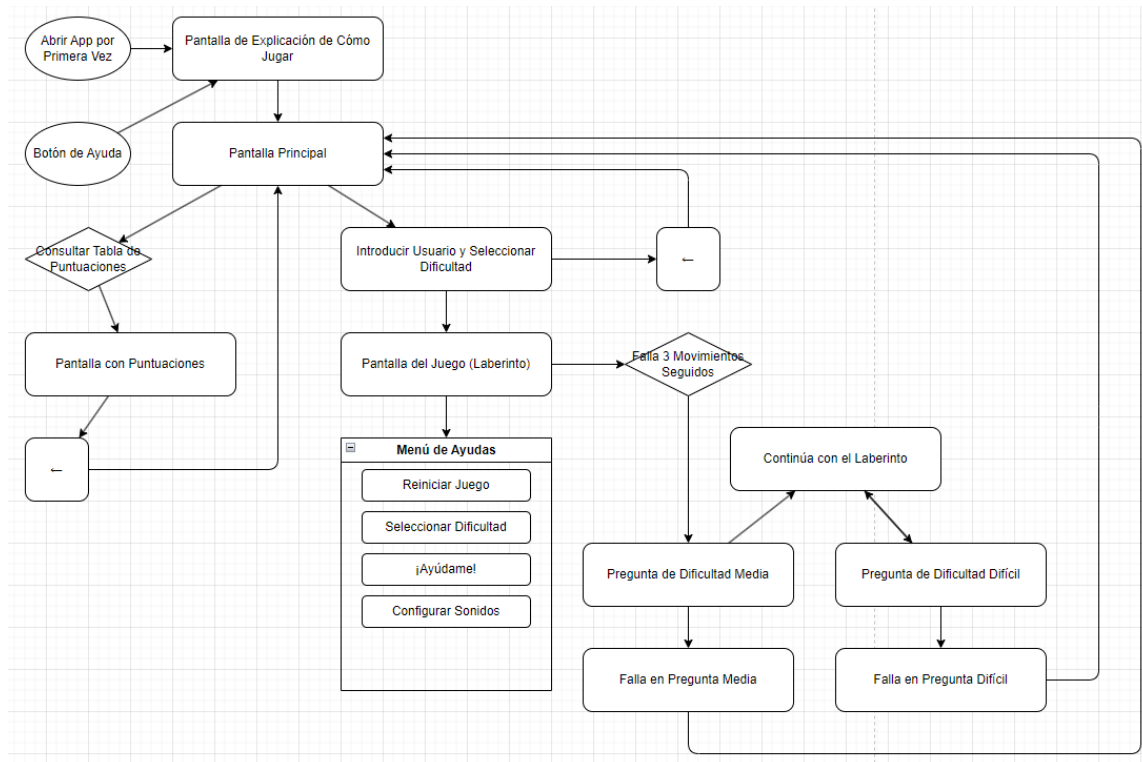
(Flujo principal en negro, flujo alternativo en rojo)

4.2. Diagrama de flujo de la aplicación

La Figura 21 representa el diagrama de flujo de la aplicación:

Figura 21

Diagrama de flujo de la aplicación



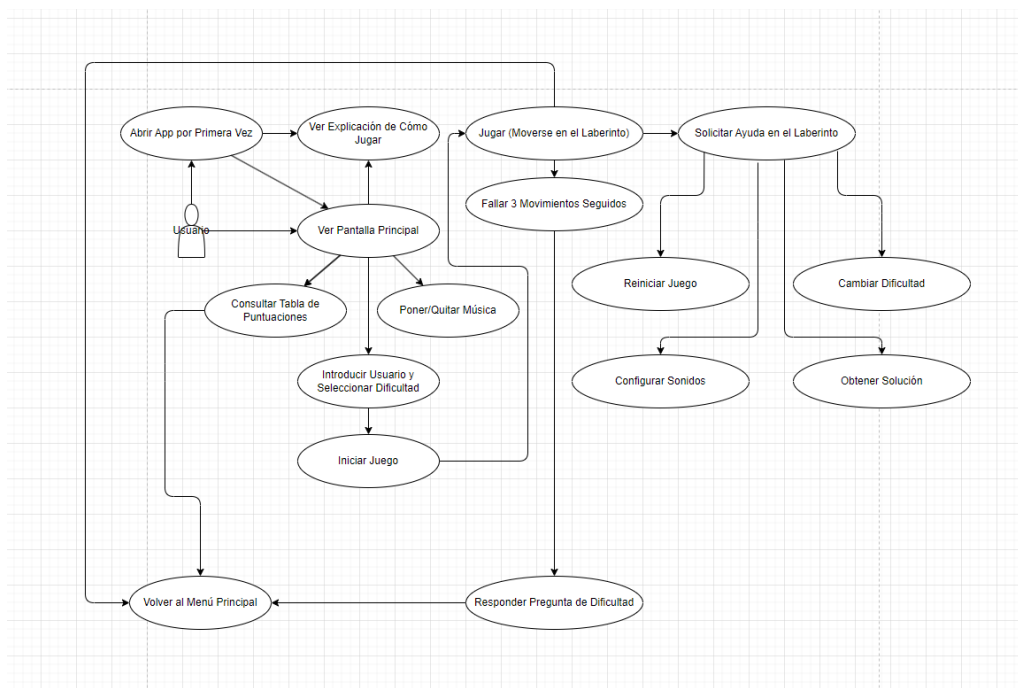
Nota: Imagen de elaboración propia.

4.3. Diagrama de casos de uso de la aplicación

La Figura 22 representa el diagrama de casos de uso de la aplicación:

Figura 22

Diagrama de casos de uso de la aplicación



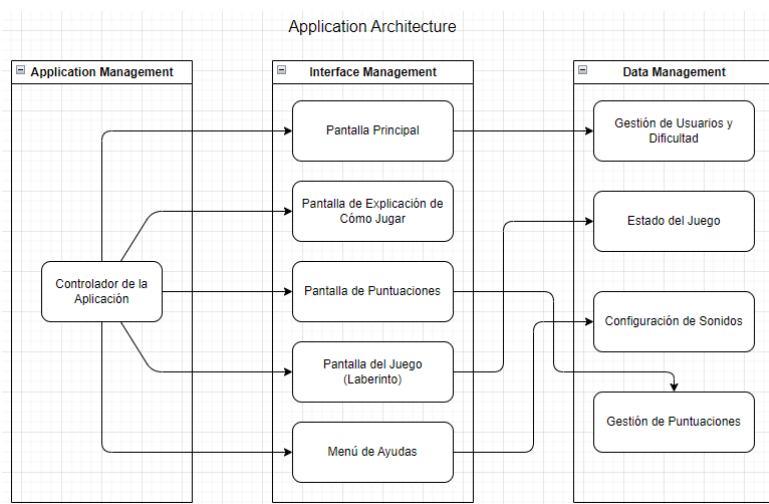
Nota: Imagen de elaboración propia.

4.4. Arquitectura de la aplicación

La Figura 23 representa la arquitectura de la aplicación:

Figura 23

Arquitectura de la aplicación



Nota: Imagen de elaboración propia.

4.5. Diseño e implementación

Se explica a grandes rasgos el diseño y la implementación de la aplicación. Se explicarán las funcionalidades más complicadas, así como las más importantes. El código de la aplicación se puede encontrar en este superíndice ¹.

4.5.1. Generación de Laberintos Aleatorios

Las partes que más han costado de implementar son, en su mayoría, las relacionadas con el laberinto. Esta es la primera que se hizo y que dio bastantes problemas.

Para la generación de estos laberintos se ha utilizado la clase **MazeGenerator**, en donde se usan tres métodos clave para esto:

- *generateMaze*. Es el primero de los métodos y el método principal de generación. Las clases estándar utilizadas aquí son **SecureRandom** para la aleatoriedad del laberinto y **Stack**, que hace de pila y se ha utilizado para ir guardando las celdas. También se utiliza una clase propia **Cell**, que representa las celdas, y distintas variables para la representación de las coordenadas x e y, y para la matriz. Se comprueba que el laberinto tenga solución y se busca la mejor solución con el método **findShortestSolutionPath**.
- *findShortestSolutionPath*. Este método se usa para calcular la mejor solución, es decir, la solución más corta. Se usa la clase estándar de Java **ArrayList**, en dos ocasiones: una es **solutions**, que se usa para ir guardando las distintas soluciones. Esto se hace a través del método de **findPathDFS**. Se recorre **solutions** y se compara cada solución para encontrar la más corta. Si durante una iteración se encuentra una solución más corta que la actual, se asigna a **shortestSolution**, manteniendo en esta variable la mejor solución. La otra ocasión es con **shortestSolution**, que se usa para ir guardando la solución más corta. Cuando se ha encontrado la mejor solución, se van marcando cada punto de esta con -1. Esto se usa más tarde para el pintado de la mejor solución.
- *findPathDFS*. En este método se buscan todas las soluciones. Se usa la clase estándar de Java **ArrayList** en dos ocasiones: una es **solutions**, que se usa para ir guardando las distintas soluciones, y la otra es **path**, que almacena el camino. También se usan dos variables, una para la coordenada x y otra para la y. Se van comprobando todas las direcciones, si has llegado a la solución, si no te puedes mover hacia la dirección o si ya se ha pasado por la celda.

También se usa la actividad **GameActivity**, que es la encargada del juego. En esta clase se usa el método **mazeGenerate**, el cual se llama al iniciar la aplicación, y se encarga de, según la dificultad seleccionada en el menú principal, establecer un tamaño u otro para el laberinto. Se resetean las variables del backtracking, se genera el laberinto con la clase **MazeGenerator** y el tamaño establecido, se prepara el pintado del laberinto con el método **mazePrinter** y se pinta con **drawMaze**, aunque el funcionamiento del pintado se explica mejor en el apartado 4.5.2.

4.5.2. Pintado gráfico del laberinto

Esta es otra de las partes que más ha costado de implementar, y es debido a que había que implementar el laberinto generado en una imagen, y que este a su vez se ajuste en

tamaño y movimiento. Esto se puede observar en la Figura 24. Seguramente fue la que más costo implementar, y esto se debe a que se ha tenido que hacer uso de tres clases estándares de Android para el pintado, donde había que hacer que se sincronizaran bien entre ellas, entre otras cosas.

Esta implementación se hace en la actividad **GameActivity**, y se hace uso de tres métodos principales, importantes de mencionar, para conseguir esto:

- *mazePrinter*. Este método es llamado en **mazeGenerate** al generarse el laberinto. Es el método encargado de la configuración inicial del pintado en la imagen. Se hace uso de dos clases estándar de Android importantes para lograr el objetivo del pintado, como son **Canvas** y **Bitmap**. También se hace uso de dos variables que representan las coordenadas del jugador, una para la coordenada X y otra para la coordenada Y. Por último, es importante mencionar que se hace uso de otra variable para la imagen, **mazeImageView**, a la que se le atribuye tanto el bitmap como el canvas. En este método se comprueba si la imagen estaba bien configurado en el momento de la llamada, si no lo estaba, se configura de nuevo y se hace la llamada al método **drawMaze**.
- *drawMaze*. Este método es llamado en **mazeGenerate** al generarse el laberinto. Es el método encargado de la visualización completa del laberinto en la imagen, en la cual se incluye al jugador y, si es necesario, el camino óptimo. Se hace uso de dos clases estándar importantes para lograr el objetivo del pintado, como son **Canvas** y **Bitmap**. Se verifican que las variables **canvas** y **mazeBitmap** para el pintado estén inicializadas, al igual que el laberinto **maze** y **lastOptimalCell** para la última casilla óptima. Se limpia la variable **canvas** y se hace uso de la clase estándar de Android **Paint**, en la cual se establece el estilo en “Fill” para que se rellene. Esta variable creada se usa en el método **mazeAndSolutionPrint**, el cual recibe **paint** y dos variables **cellWidth** y **cellHeight**, que corresponden al ancho y alto de las celdas. Después de este método se hace uso del método **drawPlayer** para dibujar al jugador y, por último, se actualiza la variable para la imagen, **mazeImageView**, con el valor del bitmap.
- *mazeAndSolutionPrint*. Se encarga de ir recorriendo el laberinto mediante la variable **maze**. Se hace uso de las clases estándar de Android **Canvas** y **Paint**. En este método se van pintando las paredes en el laberinto como bloques negros gracias a la variable **paint** y se dibujan gracias a la variable **canvas**. Estos dibujos dependen de si la posición en x e y del laberinto es 1, que corresponde a una pared, si es 3, que corresponde a la salida, o si se requiere ver la solución, donde los puntos óptimos corresponden a -1.

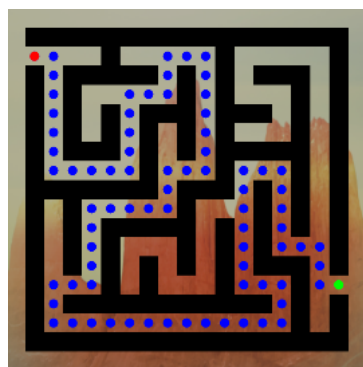
Las clases estándar de Android que se han utilizado, **Bitmap**, **Canvas** y **Paint**, se corresponden a:

- **Bitmap**, es una matriz de píxeles que representa una imagen. Se crea un bitmap del tamaño de la imagen [42].
- **Canvas**, es el área de dibujo en la que se puede pintar. Está asociado a un “Bitmap” específico. Cuando se altera el “Canvas” utilizando cualquier método para el dibujo, se altera su “Bitmap” asociado [43].

- **Paint**, especifica el estilo y color de lo que se va a dibujar, siendo en este caso bloques para las paredes o círculos para los puntos de la solución óptima, la solución o el jugador [44].

Figura 24

Laberinto in-game, con la solución óptima pintada



Nota: Imagen de elaboración propia.

4.5.3. Pseudo-backtracking

Esta parte se refiere a lo que sucede cuando un jugador comete tres fallos consecutivos seguidos, como se puede observar en la Figura 25. Fue una de las más difíciles de implementar, ya que no solo se quería que ocurriera al hacer tres fallos, sino que estos tenían que ser consecutivos. Esto permite que, si el jugador hace un “missclick” y pulsa en otro botón de movimiento, no vuelva hacia atrás al contarle como fallo, y que si el jugador hace un movimiento erróneo, pero el siguiente movimiento recula, volviendo al último punto de la solución óptima que había descubierto, no le cuente como fallo igualmente. Para esta funcionalidad se usa la “distancia de Manhattan”, previamente explicada, el pintado de los puntos incorrectos seguidos, el pintado de varios fragments donde sucede la explicación de lo ocurrido y la inicialización del activity de las preguntas en caso de ser necesario.

Esta funcionalidad sucede en la clase **GameActivity**, y se organiza en tres métodos principales. Estos son:

- *movePlayer*. Este es el método encargado del movimiento del jugador y se utiliza cada vez que se pulsa una flecha de dirección. Se hace uso de distintas variables como son: **showSolution**, para impedir el movimiento o no si se esta mostrando la solución óptima o si los botones de movimiento están deshabilitados; distintas variables para controlar las coordenadas X e Y del jugador y su movimiento; **maze**, que se usa cuando se verifican las condiciones que permiten el movimiento; **lastOptimalCell**, donde se almacena el valor del ultimo punto óptimo descubierto; **wrongMovesCount**, que se usa como contador del numero de movimientos incorrectos consecutivos; **wrongPath**, que es el que almacenena el camino incorrecto; y dos variables que usan el cálculo de la distancia de manhattan para comprobar si el usuario se está alejando o no de la solución ótima y así contarle como movimiento incorrecto.

Para el procedimiento, primero se comprueba que se permite el movimiento, verificando que no se está mostrando la solución óptima y que los botones de movimiento están habilitados con el método encargado de ello **areMovementButtonsEnabled**. Se actualiza la posición del jugador con el movimiento si se cumplen distintas condiciones de movimiento, y se inicializan las variables con el cálculo de la distancia de Manhattan, como se acaba de mencionar, para comprobar si el usuario se está alejando o no de la solución óptima. Se pinta el laberinto con **drawMaze** y si la posición del jugador con el nuevo movimiento no es un punto de la solución óptima, se comprueba si se está alejando de la solución óptima para contarle o no como movimiento incorrecto, y además se llama al método **handleWrongMove**. Si el movimiento se ha realizado hacia un punto de la solución óptima, se actualiza **lastOptimalCell** con este valor, y se limpia el camino incorrecto y el contador de movimientos incorrectos.

- *handleWrongMove*. Este es el método encargado de manejar el pintado del camino incorrecto, así como arrancar la explicación de lo que ha sucedido o las preguntas dependiendo de las veces que se ha fallado. Se hace uso de distintas variables: **wrongPath**, que almacena el camino incorrecto; **wrongMovesCount**, que es el contador del número de movimientos incorrectos consecutivos; **wrongMoveAttempts**, que es el contador del número de intentos, para así mostrar la explicación, las preguntas de nivel medio, las preguntas de nivel difícil o terminar el juego al haber perdido. También se usa como variable **score**, que representa la puntuación y a la que se le van restando puntos dependiendo del intento en el que estés.

El procedimiento trata de: añadir a **wrongPath** la posición en la que se encuentra el jugador ya que es un movimiento incorrecto, comprobar si el número de movimientos incorrectos consecutivos es 3 o superior y si se cumple la condición, deshabilitar los botones de movimiento con el método **enableMovementButtons**, aumentar en uno el contador **wrongMoveAttempts** y comprobar cuantas veces ha sucedido esto, para así pintar, según el número de intentos, el camino incorrecto con **drawIncorrectPath** y la explicación inicializando la actividad **InfoActivity**, donde se encuentra, o las preguntas de nivel medio y difícil dependiendo de si es el segundo intento o el tercero. En el caso de llevar más de 3 intentos, se vuelve a la pantalla principal, **MainActivity**, con el mensaje de haber perdido.

- *drawIncorrectPath*. Este es el método encargado de pintar el camino incorrecto. Se usa una variable de la clase estándar **List**, la cual se llama **path**, y que equivale a **wrongPath** de los otros dos métodos explicados. Es una lista de “Cell” que se inicializa como **ArrayList**. Se usan distintas variables, que son: **mazeBitmap**, el “Bitmap” para la imagen explicado previamente en otras funcionalidades; **canvas**, que es el “canvas” de la imagen explicado previamente en otras funcionalidades; **path**, la cual se acaba de explicar; **paint**, para establecer un color al punto del “canvas” que se usan como puntos del camino incorrecto; dos variables para el ancho y el alto de los puntos que se van a usar como puntos del camino incorrecto; y **mazeImageView**, que es la imagen asociada al “Bitmap”, y que contiene el laberinto.

El procedimiento del método es: se verifica que las variables necesarias **mazeBitmap**, **canvas** y **path** no estén vacías, Se establece un color al **paint** y un ancho y alto para los puntos que se usarán como puntos del camino incorrecto, se dibujan

los puntos recorriendo **path**, se actualiza la imagen con el “Bitmap” y, pasado un tiempo se borra el camino incorrecto y se vuelve a permitir el movimiento con el método **enableMovementButtons**.

Es importante mencionar que este método es usado tanto en **handleWrongMove** cuando es el primer intento como en **onActivityResult**, en el cual se verifica si la respuesta a la pregunta es correcta con la variable booleana **isCorrect**, y si lo es, se llama al método **drawIncorrectPath** para pintar el camino incorrecto.

Figura 25

Laberinto in-game, con la reconducción por haber realizado 3 movimientos incorrectos consecutivos



Nota: Imagen de elaboración propia.

4.5.4. Preguntas

Esta parte del diseño se refiere a las preguntas que salen al hacer tres movimientos incorrectos consecutivos por segunda y tercera vez. Para esta implementación se ha usado la activity **GameActivity**, de donde arranca, y la activity **TrivialActivity**, donde se generan y maneja todo el control relativo a las preguntas.

Aunque esta parte del diseño no fue una de las partes que más problemas dio hacer, es la parte con más implementación. A continuación se explican los métodos relativos a **GameActivity** para esta implementación:

- *handleWrongMove*, (*Explicación muy parecida a la explicada anteriormente, pero con más detalles de la parte del trivial*). Esta parte de la explicación es la misma que el *handleWrongMove* explicado en el diseño del pseudo-backtracking.

Para la parte de este método correspondiente a las preguntas, es importante mencionar que en el segundo y tercer intento, cuando **wrongMoveAttempts** es 2 o 3, se utiliza un método para iniciar la actividad de las preguntas, el cual es *startTrivial*. Para ello, se le pasa al método un string con la dificultad de las preguntas, sabiendo así qué preguntas cargar.

- *startTrivial*. Este es el método que arranca la actividad de las preguntas. Recibe un string, **difficulty**, con la dificultad de las preguntas, para así cargar las preguntas de nivel medio o de nivel difícil. Con ello arranca la actividad **TrivialActivity**, pasándole a la actividad esa dificultad. También se le pasa una constante, **TRIVIAL_CODE**, que hace de clave para identificar si la actividad de la que estamos manejando la respuesta en el método **onActivityResult** es la actividad del trivial.

- *onActivityResult*. Este es el método encargado de manejar si la respuesta de las preguntas es correcta o no y qué se debe hacer en cada caso. Este método es propio de la extensión que se está usando de **AppCompactActivity**, y se usa esperando un resultado de una actividad iniciada con el método **startActivityForResult**, como es en el caso de **TrivialActivity**.

En este método se usan distintas variables propias del método: tres enteros **requestCode**, **resultCode** y **data**, los cuales se usan respectivamente para identificar que el resultado es de la actividad que esperabas, que el resultado es satisfactorio, es decir, que ha ido bien, y que hay datos. También se usan otras variables como: **isCorrect**, guarda el resultado de si la respuesta a la pregunta es correcto o no; **wrongPath**, para dibujar el camino incorrecto si se ha acertado la pregunta; **wrongMovesCount**, que se tiene que resetar cuando se hace la espera antes de mover al jugador; **lastOptimalCell**, que guarda el ultimo punto de la solución óptima descubierto; y dos variables, **playerX** y **playerY**, que almacenan la posición del jugador.

El procedimiento es el siguiente: se comprueba que es el resultado de la actividad que esperábamos, que da un resultado satisfactorio y que hay datos, se asigna el resultado de la respuesta a la pregunta al booleano **isCorrect**, y se comprueba si es correcto o no. Si es correcto, se deshabilitan los botones de movimiento con el método **enableMovementButtons**, mientras se hace el pintado del camino incorrecto y la recolocación, se actualiza el valor de la posición del jugador en X e Y con el ultimo punto de la solución óptima descubierto, se redibuja el laberinto, se pinta el camino incorrecto con el método **drawIncorrectPath** y se espera un tiempo antes de dibujar el laberinto con los cambios de la posición del jugador. Si la respuesta a la pregunta era incorrecta se inicia **MainActivity** bajo un string identificatorio de que se ha perdido.

A continuación se explican los métodos más importantes de **TrivialActivity** para esta implementación:

- *onDataChange*. Este es el método principal encargado de las preguntas y es un método propio de Firebase, además recibe los datos del nodo al que apunta la base de datos. En él se comprueba si hay datos relativos a las preguntas en la base de datos, y se selecciona una pregunta para mostrar. La dificultad de esta pregunta viene marcada desde la inicialización, mostrado en la Figura 26 y la creación de la base de datos mostrada en Figura 27, ya que se inicializa a través del nodo preguntas y su dificultad, que se muestra en la Figura 28.

En este método se utiliza la variable propia del método **snapshot**, la cual es de la clase propia de “Firebase” **DataSnapshot** y que recibe los datos de la base de datos. También se usan distintas variables creadas: **questionCount**, almacena el número de preguntas en la dificultad; **randomIndex**, usando la clase estándar de Java, **SecureRandom**, se genera un índice aleatorio seguro en función del número de preguntas en el nodo; **currentIndex**, se usa para ir iterando en las preguntas y así seleccionar una aleatoria; **question**, la cual es de la clase propia de “Firebase” **DataSnapshot**, y que recibe los datos de cada pregunta; y **selectedQuestion** la cual es también de la clase propia de “Firebase” **DataSnapshot**, y que almacenará los datos de la pregunta seleccionada. Además se hace uso del método

displayQuestion, que es el encargado de mostrar las preguntas y de manejarlas según su dificultad.

El procedimiento trata de: se comprueba primero que el nodo al que apunta existe y contiene datos, se obtiene el número de hijos del nodo, se obtiene un índice aleatorio en función del número de hijos del nodo, se recorren las preguntas y se va comprobando si **currentIndex**, que almacena que la pregunta iterada es la seleccionada aleatoriamente. De ser así, la pregunta seleccionada, **selectedQuestion**, obtiene el valor de la pregunta bajo ese índice. Si no, se sigue iterando en función del **currentIndex**, y se muestran las preguntas a través del método **displayQuestion**.

Figura 26

Código de inicialización de la base de datos de preguntas

```
databaseReference = FirebaseDatabase.getInstance().getReference(path: "preguntas").child(difficulty);
```

Nota: Imagen de elaboración propia.

Figura 27

Código de creación de la base de datos de preguntas

```
if (savedInstanceState != null) {  
    restoreState(savedInstanceState);  
} else {  
    databaseReference.addListenerForSingleValueEvent(this);  
}
```

Nota: Imagen de elaboración propia.

Figura 28

Estructura de la base de datos de preguntas



Nota: Imagen de elaboración propia.

- *displayQuestion*. Este es el método encargado de establecer la configuración inicial de las preguntas, estableciendo los nodos simples como: el título, la imagen de la pregunta, cual es la respuesta correcta y el enunciado de la pregunta. En este método también se llama a los métodos de **mediumQuestions** y **difficultQuestions** que manejan el cargado y pintado de las opciones de las distintas preguntas en función de la dificultad.

Se utilizan las variables: **selectedQuestion**, la cual es de la clase propia de “Firebase” **DataSnapshot**, y que almacenara los datos de la pregunta seleccionada; **titleTextView**, es el título de la pregunta, **questionTextView**, es el enunciado de la pregunta; **correctAnswer**, que es un String que almacena la respuesta correcta de la pregunta, ya sea en las preguntas de nivel medio, el índice de la opción correcta o en las de nivel difícil, el nombre de la imagen correcta; **imageResourceName**, que es otro String que almacena el nombre del recurso de la imagen de la pregunta; **imageResourceId**, que es un entero que almacena el identificador del recurso de la imagen de la pregunta; **imageView**, que es la imagen de la pregunta; **optionsLayout**, es de la clase estándar de Android **GridLayout** y organiza y muestra las opciones de respuesta; **optionViews**, es de la clase estándar de Java “ArrayList” y es una lista que almacena las vistas de las opciones de respuesta; y **difficulty**, que es un string que almacena la dificultad de la pregunta. Además se utilizan los métodos **mediumQuestions** y **difficultQuestions** a los que se les llama en función de la dificultad de la pregunta.

El procedimiento es el siguiente: se comprueba que hay pregunta seleccionada, no siendo esta “null”. Si se cumple la condición, se establece el valor del **TextView** para el título y el enunciado de la pregunta, la respuesta correcta y el nombre de la imagen a través de la variable **selectedQuestion**, que contiene los datos relativos a la pregunta seleccionada. Se comprueba que existe un id para la imagen de la pregunta con ese nombre en los recursos y si es así se asigna a la **Imageview** de la pregunta, se limpian tanto **optionsLayout** como **optionsViews** y se inician los métodos **mediumQuestions** o **difficultQuestions** según la dificultad de la pregunta.

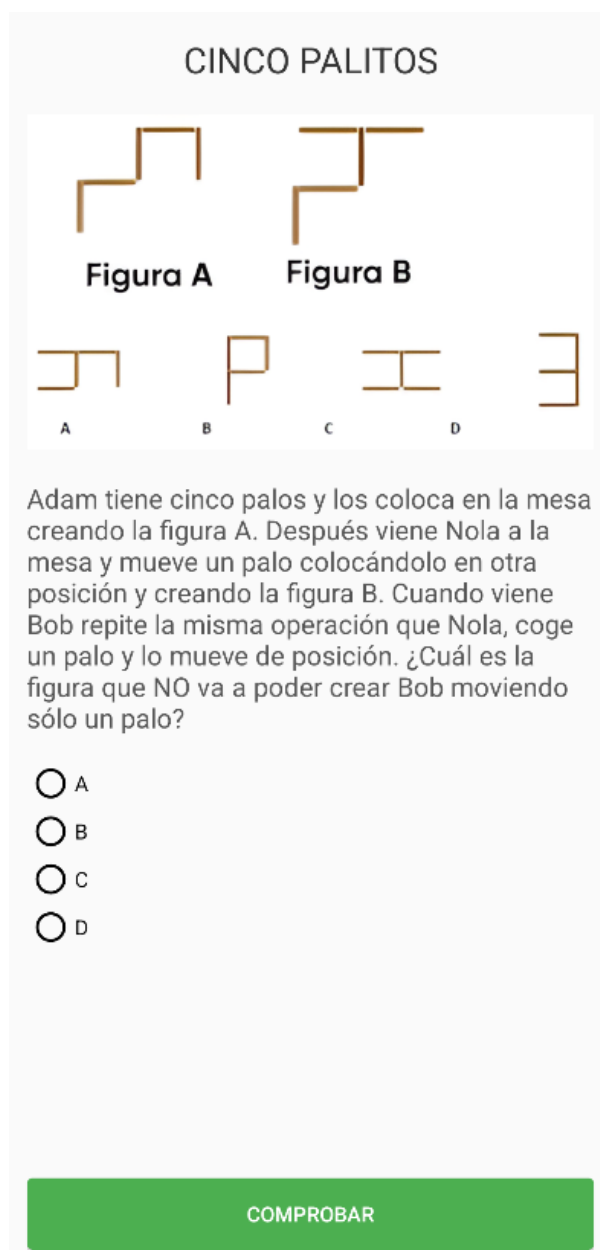
- *mediumQuestions*. Este método es el encargado de pintar las opciones de las preguntas de nivel medio, como se muestra en la Figura 29. Hace uso de distintas variables como son: **radioGroup**, que se usa para agrupar los botones y que así solo se pueda seleccionar uno a la vez; **optionSnapshot**, la cual es de la clase propia de “Firebase” **DataSnapshot**, y que se usara para ir iterando sobre el nodo “opciones” de las preguntas; **selectedQuestion**, la cual como se ha mencionado varias veces, es de la clase propia de “Firebase” **DataSnapshot**, y que almacenará en este caso los datos de las opciones de la pregunta seleccionada; **radioButton**, que es un botón “de radio” que se usará para seleccionar una opción; **selectedOption**, que es un **radioButton** que almacena la opción seleccionada; **optionViews**, que es la misma variable que la explicada en el método anterior, donde es de la clase estándar de Java “ArrayList”, y es una lista que almacena las vistas de las opciones de respuesta; y **optionsLayout**, que también es la misma variable que la explicada en el método anterior, donde es de la clase estándar de Android **GridLayout** y organiza y muestra las opciones de respuesta.

El procedimiento es: se crea la variable **radioGroup**, se recorre el **DataSnapshot** de nombre **optionSnapshot** donde se itera a través de los datos de las opciones

de la pregunta seleccionada, se crean botones de nombre **radioButton**, a los que se les establece el texto de las opciones, un color a este texto y un color al botón, se comprueba si se ha pulsado alguno de las botones de opción, si se ha pulsado alguno y ya había uno seleccionado se desmarca el anterior, se añaden cada uno de los botones al grupo **radioGroup** y también a la lista **optionViews**, y por último se añade el grupo **radioGroup** al **optionsLayout**.

Figura 29

Pantalla con una pregunta de nivel medio



Nota: Imagen de elaboración propia.

- *difficultQuestions*. Este método es el encargado de pintar las preguntas de nivel difícil, como se muestra en la Figura 30. Hace uso de distintas variables como son: **optionSnapshot**, la cual es de la clase propia de “Firebase” **DataSnapshot**, y

que se usa para ir iterando sobre el nodo “opciones” de las preguntas; **selected-Question**, la cual como se ha mencionado varias veces, es de la clase propia de “Firebase” **DataSnapshot**, y almacenará, igual que en el caso anterior los datos de las opciones de la pregunta seleccionada; **optionContainer**, que es de la clase estándar de Android **LinearLayout**, y que se usa para contener tanto las imágenes de las opciones como los botones de cada opción; **optionImageView**, que representa la imagen de cada opción; **optionImageResourceName**, que es un String que almacena el nombre del recurso de las imágenes de cada opción; **optionImageResourceId**, que es un entero que almacena el identificador del recurso de las imágenes de cada opción; **radioButton**, que es un botón “de radio” que se usara para seleccionar una opción; **selectedOption**, que es un **radioButton** que almacena la opción seleccionada; **imageSize**, que se usa para ajustar el tamaño de las imágenes de las opciones, y representa tanto el ancho como el largo de las imágenes; **optionViews**, que es la misma variable que la explicada en los métodos anteriores, donde es de la clase estándar de Java “ArrayList”, y es una lista que almacena las vistas de las opciones de respuesta; y **optionsLayout**, que también es la misma variable que la explicada en los métodos anteriores, donde es de la clase estándar de Android **GridLayout** y organiza y muestra las opciones de respuesta.

El procedimiento es el siguiente: se recorre el **DataSnapshot** de nombre **optionSnapshot** donde se itera a través de los datos de las opciones de la pregunta seleccionada, se crea un **LinearLayout** bajo el nombre **optionContainer** para contener tanto las imágenes como los botones, y se le establece la orientación y el padding entre sus distintas opciones. Se crea una imagen para las distintas opciones y se asigna el nombre de las opciones de las imágenes a **optionImageResourceName** a través de la variable **optionSnapshot**, que contiene los datos de las opciones de la pregunta seleccionada. Se comprueba que existe un id para la imagen de la opción con ese nombre en los recursos y si es así se asigna a la **Imageview** de la opción de la pregunta correspondiente **optionImageView**. Se crea un **radioButton**, al que se le asigna el nombre del recurso para la imagen, se cambia el color del **radioButton** y se comprueba si se ha pulsado alguno de los botones de opción. Si se ha pulsado alguno y ya había uno seleccionado se desmarca el anterior. Se establece un ancho y alto para las imágenes de las opciones, se le establece y se centra en cada posición, se añade la imagen de cada opción al **LinearLayout** de nombre **optionContainer**, se le añaden también los **radioButton**, se añade el **optionContainer** al **optionLayout** y también se le añade el **optioncontainer** al **optionViews**.

Figura 30

Pantalla con una pregunta de nivel difícil

TORNEO DE CASTORES

Se está celebrando un torneo de castores donde la clasificación de las distintas fases se va apuntando en una pizarra. Los corredores tienen un número asignado del 1 al 8 durante todo el torneo para poder ser identificados. Cuando se termina el torneo, las tarjetas de todos los participantes se mezclan y sólo se tienen colocadas las tarjetas de los participantes en la primera fase. ¿Podrías poner las demás tarjetas (lateral derecho) en el orden correcto?

COMPROBAR

Nota: Imagen de elaboración propia.

- *onClick*. Este método es el encargado de manejar la pulsación del botón bajo el texto “Comprobar”. Este es un método propio de Android que actúa como “listener” y cuando se pulsa el botón que controla, hace la acción establecida en él.

Se utilizan las variables: **view**, para obtener el id de la pulsación; **selectedOption**,

para comprobar si se ha escogido una opción; **isCorrect**, booleano ya mencionado que guarda si la respuesta a la pregunta es correcta; **difficulty**, para diferenciar entre los niveles de dificultad; **selectedOption**, que es el entero con la opción seleccionada; y un **Intent** el cual arranca el intent que arrancó la actividad **Trivia-IActivity**, ya que no se le pasan valores, y este es **GameActivity**. A este activity se le pasa una constante que actúa como clave para representar si todo ha ido bien, **RESULT_OK** y si la respuesta es correcta o no bajo la clave **trivial_correct**.

El procedimiento es muy sencillo: si se ha pulsado el botón de “Comprobar” y se ha seleccionado alguna opción en la pregunta se comprueba cuál es la dificultad. Si es “difícil”, se le asigna a **isCorrect** la respuesta correcta mediante el “tag” ya que se comprueba la respuesta correcta por el nombre del recurso. Si es “medio” se le asigna mediante “text” ya que se comprueba por texto. Se arranca la actividad de **GameActivity** pasándole si la respuesta es correcta y si ha ido bien. Si no se había seleccionado una opción antes de pulsar el botón, sale una advertencia que pide seleccionar una opción.

4.5.5. Puntuaciones y Tabla de Puntuaciones

Esta parte del diseño se refiere a la puntuación que se obtiene cuando ganas una partida del laberinto. Para esta implementación se ha usado la actividad **GameActivity**, donde sucede toda su implementación, **GamesActivity**, que controla la actividad que tiene la tabla de puntuaciones, y la clase **DdGames**, que es la base de datos donde se guardan las partidas.

Aunque esta parte del diseño no fue una de las partes que más problemas dio hacer, es una parte con bastante implementación y gran importancia en la aplicación. A continuación se explican los métodos importantes relativos a **GameActivity** para esta implementación:

- *scoreInit*. Este método es el encargado de inicializar los puntos según la dificultad. Es llamado al final del método **instancias**, cuando lo demás ha sido instanciado. En él se usan dos variables, **difficulty**, para navegar por el switch en función de la dificultad seleccionada para el laberinto, y **score**, al cual se le asignan unos puntos u otros en función de la dificultad correspondiente.

El procedimiento es muy simple: se itera sobre **difficulty** con un switch y según el case al que corresponda, se le asigna a **score** los puntos correspondientes a la dificultad: 1550 para “difícil”, 1150 para “Normal” y 800 para “Fácil”

- *saveGame*. Este método es el encargado de guardar las partidas en la base de datos una vez el jugador ha perdido o ganado. En él se usan las variables: **dbGames**, que es una variable de la clase propia **DbGames**, y que sirve para manejar la base de datos de las partidas; **game**, que es una variable de la clase propia **Game**, la cual es una clase POJO (Plain Old Java Object) que guarda los datos relativos a un juego o partida; y **allGames**, que es un **List** de datos de tipo **Game** que se usa para guardar las partidas.

El procedimiento es también muy simple: se crea una variable **dbGames** de la clase de la base de datos de partidas, se crea una variable de la clase **Game**, la cual se llama también **game**, y a la que se le asigna el nombre recibido desde **MainActivity**, la dificultad con la que se ha jugado la partida (la última establecida antes

de ganar, ya que se puede cambiar en la partida) y la puntuación, asegurándose de que al menos sea 0. Se crea una variable que hace de lista para guardar las partidas, se le asignan los datos de las partidas, se añade la nueva partida y se ordena la puntuación. También se ordenan desde la base de datos para manejar que pueda haber empates y se guarda el juego, ya con el rango actualizado.

- *handleWrongMove* (Explicación muy parecida a la explicada para las preguntas, pero con más detalles del manejo de los puntos en esta). Esta parte de la explicación es la misma que el `handleWrongMove` explicado en el diseño del pseudo-backtracking.

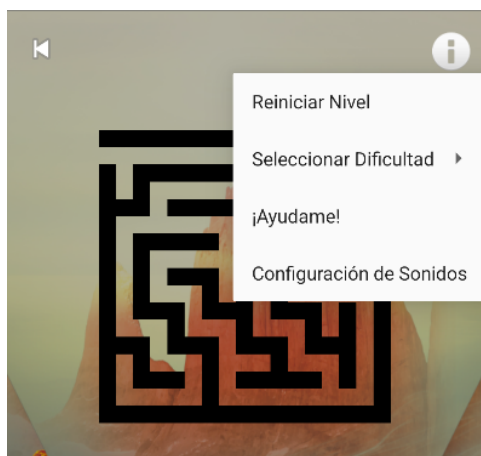
Para la parte de este método correspondiente a la puntuación, es importante mencionar que en cualquiera de los tres primeros intentos, lo primero que se hace es restar 200 puntos a la puntuación base de la dificultad seleccionada. Esto sucede cuando `wrongMoveAttempts` es 1, 2 o 3, ya que si es mayor que 3, directamente se pierde y se arranca el método `gameLost`, el cual establece la puntuación en 0, ya que no se ha sido posible completar el laberinto.

- *onMenuItemClick*. Este método es el encargado del manejo de las pulsaciones del menú encontrado en la esquina superior derecha, bajo un símbolo de información. Este es un método propio de la clase estándar de Android y maneja las interacciones con los distintos items del menú, los cuáles se pueden observar en la Figura 31 y Figura 32. Se usan las variables: `menuItem`, que representa los distintos items del menú; `score`, el entero que representa la puntuación y al cual se le reasigna el valor cuando se selecciona una dificultad; y `wrongMoveAttempts` el cual se resetea también cuando se selecciona una dificultad.

El procedimiento es muy simple: se comprueban los ids. Si el id es el del botón de “Reiniciar Nivel” se llama al método `restartLevel` y se reinicia el nivel. Si se ha pulsado “Seleccionar Dificultad”, al pulsar en cualquiera de las opciones, se le asigna la dificultad con el método `setDifficulty`, y el `score` correspondiente a la opción seleccionada. Si se ha pulsado el botón de “¡Ayúdame!” se llama al método de `showSolutionDialog`, el cual muestra un diálogo para mostrar la solución o no. Y si se ha pulsado el botón de “Configuración de Sonidos”, se llama al método `toggleSoundDialog`, para manejar la música reproduciéndola, parándola, subiéndola o bajándola.

Figura 31

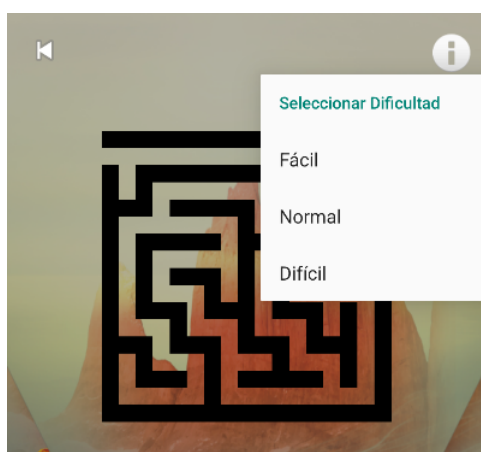
Pantalla con el menú de opciones



Nota: Imagen de elaboración propia.

Figura 32

Pantalla con la opción de Seleccionar Dificultad dentro del menú de opciones



Nota: Imagen de elaboración propia.

Aunque no tienen casi implementación, también se debe mencionar **gameWon** y **gameLost**, en los cuales se llama al método **saveGame** cuando estos se ejecutan, y que son ejecutados cuando se gana una partida o se pierde. En el caso de **gameLost** se establece la variable **score** a 0. A continuación se explican los métodos importantes relativos a **GamesActivity**:

- *instancias*. Este método es el encargado de instanciar las variables utilizadas en toda la “activity” y es importante para la implementación de las puntuaciones ya que se insertan datos a la base de datos si esta está vacía.

Se usan distintas variables, como son: **gamesListView**, el cual es de la clase estándar de Android **listView**, y que almacena las distintas partidas bajo una interfaz personalizada; **buttonBack**, que es un botón para volver a **MainActivity**; y **dbGames** que es una variable de la clase propia **DbGames**, y la cual sirve para manejar la base de datos de las partidas.

El procedimiento es: se instancian las tres variables y si la lista de juegos de **dbGames** esta vacía se insertan datos con el método **insertInitialData** de la variable **dbGames**.

- *loadGames*. Este método es el encargado de cargar los juegos en la lista de juegos **gamesListView**. Este método se llama en el **onCreate** de la “activity”, una vez se han creado las instancias y aplicado el “listener” al botón de retroceso. Este método usa las variables: **games**, que es un **List** de datos de tipo **Game** que se usa para guardar las partidas; **dbGames**, la cual se ha explicado previamente varias veces, y que es una variable de la clase propia **DbGames**, y que sirve para manejar la base de datos de las partidas; **adapter**, el cual es el adaptador de la clase propia **GamesAdapter**, y que es un “Adapter” personalizado para la lista de partidas; y **gamesListView**, que es de la clase estándar de Android **listView** y que almacena las distintas partidas bajo una interfaz personalizada.

El procedimiento de este método es: se crea una variable con el nombre de **games**, que es una lista a la cual se le asignan las partidas de la base de datos, se ordenan los juegos por puntuación, se le asigna a **adapter** estas partidas y se le establece a **gamesListView** este **adapter**.

Es importante mencionar que la interfaz de esta pantalla esta basada en otra interfaz creada para una práctica que fue realizó por mí mismo y dos compañeros en este curso [45]. A continuación se explican los métodos importantes relativos a **DbGames**:

- *saveGame*. Este método es el encargado de guardar los juegos en la base de datos. Se llama dentro del método **saveGame** de **GameActivity** y en los insert que se hacen cuando la base de datos esta vacía con el método **insertInitialData**. Se usan las variables: **dbHelper**, la cual es de la clase propia de “SQLite” **DbHelper** y que representa la base de datos; **db**, la cual es de la clase propia de “SQLite” **SQLiteDatabase** y que permite manipular la base de datos; **values**, que es de la clase estándar de Android **ContentValues**, y que almacena los datos de la partida para insertarlos en la base de datos; y **game**, que es una variable de la clase propia **Game**, la cual es una clase POJO (Plain Old Java Object) y que guarda los datos relativos a un juego o partida.

El procedimiento es: se crea la variable **db** de tipo **SQLiteDatabase** para la escritura en la base de datos, se crea la variable **values** para almacenar los datos de la partida y se guardan en cada campo de la base de datos valores en función de la variable **game**. Por último, se inserta en la base de datos este **values** con la nueva fila.

- *getAllGames*. Este método es el encargado de devolver todos los juegos y es usado tanto en **GameActivity** como en **GamesActivity** para recibir todos los valores de la base de datos e insertar nuevos a partir de esto. Se usan distintas variables como son: **games**, que es un **List** de datos de tipo **Game** que se usa para guardar las partidas y que se inicializa como “ArrayList”; **dbHelper**, la cual es de la clase propia de “SQLite” **DbHelper** y que representa la base de datos, como ya se ha mencionado anteriormente; **db**, la cual es de la clase propia de “SQLite” **SQLiteDatabase** y que permite manipular la base de datos, como ya se ha mencionado también anteriormente; **cursor**, la cual es de la clase estándar de Android **Cursor** y que se usa para iterar sobre los resultados de la consulta a la tabla “games”; y **game**,

que es una variable de la clase propia **Game**, la cual como se ha mencionado varias veces previamente, es una clase POJO (Plain Old Java Object) que guarda los datos relativos a un juego o partida.


El procedimiento es: se crea la variable **games**, para almacenar los juegos o partidas, se crea la variable **db** de tipo **SQLiteDatabase** para la lectura de la base de datos, se crea un **cursor** para ir recorriendo la tabla de juegos iterando y añadiendo a la lista de juegos, **games**, los juegos creados a partir de recibir lo que hay en la base de datos bajo el nombre “id”, “playerName”, “difficulty”, “score” y “rank”, y se devuelve la lista de juegos.

- *insertInitialData*. Este método es el encargado de insertar datos iniciales en la base de datos si está vacía. Estos datos se pueden observar en la Figura 33. Se utiliza en **GamesActivity**, dentro del método *instancias* como se ha explicado anteriormente. Se utilizan las variables ya mencionadas con anterioridad, llamadas: **db**, **cursor**, **dbHelper** y **game**.

El procedimiento es: se crea la variable **db** de tipo **SQLiteDatabase** para la escritura en la base de datos, se crea un **cursor**, y se comprueba si este tiene datos. Si no los tiene, se crean tres partidas o juegos distintos con distintas dificultades y el máximo de puntos posibles para esas dificultades, estableciéndose los valores usando los métodos de **game** y luego usando **saveGame** para insertarlos en la base de datos.

Figura 33

Pantalla que muestra la tabla de Puntuaciones, con los datos iniciales



Dificultad	Nombre	Pts	Rank
Difícil	Javier	1500	1
Normal	Javier	1150	2
Fácil	Javier	800	3

Nota: Imagen de elaboración propia.

- *updateRanks*. Este método se encarga de actualizar los rangos de la tabla de puntuaciones una vez se inserta una nueva partida. Este método maneja la posibilidad

de empates entre puntuaciones. Se utilizan distintas variables, las cuales son: **games**, que es un **List** de datos de tipo **Game** que contiene las partidas; **dbHelper**, la cual es de la clase propia de “SQLite” **DbHelper** y que representa la base de datos, como ya se ha mencionado anteriormente; **db**, la cual es de la clase propia de “SQLite” **SQLiteDatabase** y que permite manipular la base de datos, como también se ha mencionado anteriormente; **currentRank**, que es un entero que almacena el rango inicial y que se utiliza para asignar rangos a las partidas; **game**, que es una variable de la clase propia **Game**, la cual como se ha mencionado varias veces previamente, es una clase POJO (Plain Old Java Object) que guarda los datos relativos a un juego o partida; y **values**, que es de la clase estándar de Android **ContentValues** y que se usa para actualizar la base de datos con el nuevo rango de las partidas.

El procedimiento es: se crea la variable **db** de tipo **SQLiteDatabase** para la escritura en la base de datos, se inicializa la variable **currentRank** a 1, la cual se utilizará para asignar los rangos a las partidas, se va recorriendo la lista de juegos, se obtiene el juego actual, y se compara su puntuación con la del juego anterior. Si las puntuaciones son iguales, se asigna el mismo rango, si no, se asigna **currentRank**. Se crea un **ContentValues** para actualizar el rango en la base de datos y se ajusta **currentRank** si la siguiente puntuación es diferente. Se itera sobre toda la lista de juegos, actualizando así todos los valores de la tabla si es necesario.

4.5.6. Música

Esta parte del diseño se refiere a la música que se puede poner de fondo en la aplicación. Para esto puedes activar o desactivar la música en la actividad **MainActivity** o en el menú de arriba a la derecha de **GameActivity**. Estas dos activities son las dos activities que se han ocupado del diseño de esta funcionalidad, junto con la clase **MusicManager**, que es la encargada de manejar la música.

Aunque esta parte del diseño no fue una de las más problemáticas, es necesario mencionarla ya que tiene un gran peso en pantalla. A continuación, se explican los métodos importantes relativos a **MainActivity** para esta implementación:

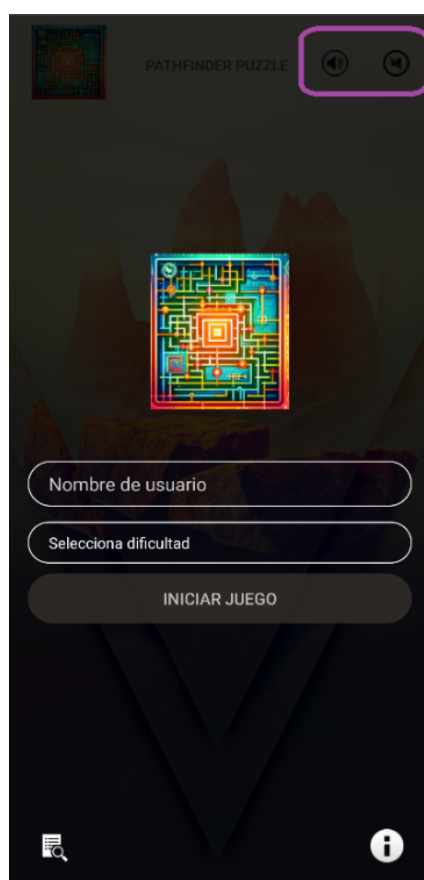
- *onClick*. Este método controla la pulsación de los distintos botones de la actividad **MainActivity**. En este caso, los dos botones que nos interesan son los de id **sound** y **nosound**, que son los que activan y desactivan la música, y estos se pueden observar en la Figura 34. Este método es un método propio de Android que actúa como “listener” y cuando se pulsa el botón correspondiente, realiza la acción establecida en él. Contiene las variables: **view**, para obtener el id de la pulsación y comprobar el botón pulsado; **intent** que se utiliza distintas veces para arrancar las actividades de **GamesActivity** en el caso de haberse pulsado la tabla de puntuaciones, o arrancar **GameActivity** en el caso de haber rellenado bien los campos y haberle dado al botón de “Iniciar Juego”; **playButton** y **pauseButton**, que son los botones que nos interesan ya que activan o desactivan la música; **userName**, que contiene el nombre de usuario introducido; y **selectedDifficulty**, que contiene la dificultad seleccionada.

El procedimiento es muy simple: se comprueba el id de la pulsación. Si es **gamesList**, se arranca **GamesActivity** a través de un Intent. Si es el id **helpAppButton** se arranca el método **launchHelpActivity**. Si es el id **sound**, se arranca la música

usando la clase **MusicManager**, y llamando a su método **playMusic**, activando el botón de pausa y desactivando el botón de poner música. Si es el id **nosound**, se pausa la música usando la clase **MusicManager** y llamando a su método **pauseMusic**, desactivando el botón de pausa y activando el botón de poner música. Si por último el id de la pulsación es **buton_start**, se comprueba si se ha introducido nombre y seleccionado dificultad; si es así se arranca la actividad **GameActivity** para el juego y se le pasa tanto la dificultad como el nombre. Si no se había introducido nombre o seleccionado dificultad, pide hacerlo.

Figura 34

Pantalla que muestra el control de la musica en MainActivity, marcado con recuadro rosa



Nota: Imagen de elaboración propia.

A continuación se explican los métodos importantes relativos a **GameActivity**:

- *toggleSoundDialog*. Este es el método encargado del manejo de la música en **GameActivity**. En este método se maneja el volumen de la música, pudiendo subirlo o bajarlo, así como pausarla o reproducirla. Para ello, se muestra una modal donde aparece una barra de sonido para ajustar el volumen y dos textos que actúan como botones: uno con el texto “Cerrar” para cerrar la modal, y otro con el texto “Pausar Música” o “Reproducir Música” dependiendo de si ya se esta reproduciendo música o no.

Se utilizan distintas variables: **builder**, el cual es de la clase estándar de “Android” **AlertDialog.Builder** y que se utiliza para crear la modal para la “Configuración de Sonidos”; **volumeControl**, que es de la clase estándar de “Android” **SeekBar** y que representa la barra de volumen con la que subir y bajar el volumen; **savedVolume**, que es un float que almacena el volumen; y **dialog**, que es de la clase estándar de “Android” **AlertDialog** y que se utiliza para representar la modal.

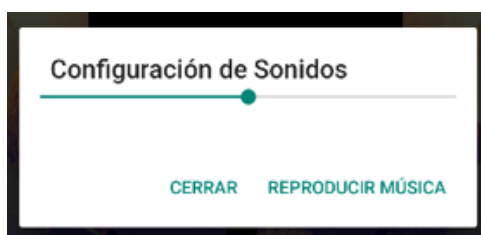
El procedimiento es: se crea el **builder** para la modal y se le establece un título. Si la barra de volumen **volumeControl** ya estaba asignada a alguna vista, se elimina. Se recibe el **savedVolume**, se establece el máximo de volumen mediante la constante **MAX_VOLUME** y se ajusta el progreso de la barra de volumen en función del máximo posible y el guardado. Se establece un “listener” a la barra para que, si se cambia el volumen de la barra, este se actualice en el método propio de la clase “SeekBar” **onProgressChanged**. Por último, se le asigna la barra de sonido al **builder**, se configura la funcionalidad para pausar o reproducir la música según el estado actual y se muestra el **dialog**.

- *onMenuItemClick*. Este método es el encargado del manejo de las pulsaciones del menú encontrado en la esquina superior derecha, bajo un símbolo de información. Este es un método propio de la clase estándar de Android y maneja las interacciones con los distintos items del menú. Se usan las variables: **menuItem**, que representa los distintos items del menú; **score**, que es un entero que representa la puntuación y al cual se le reasigna el valor cuando se selecciona una dificultad; y **wrongMoveAttempts** que se resetea también cuando se selecciona una dificultad.

El procedimiento es muy simple: se comprueban los ids. Si el id es el del botón de “Reiniciar Nivel” se llama al método **restartLevel** y se reinicia el nivel. Si se ha pulsado “Seleccionar Dificultad”, al pulsar en cualquiera de las opciones, se le asigna la dificultad con el método **setDifficulty**, y el **score** correspondiente a la opción seleccionada. Si se ha pulsado el botón de “¡Ayúdame!”, se llama al método de **showSolutionDialog**, el cual muestra un diálogo para mostrar la solución o no; Si se ha pulsado el botón de “Configuración de Sonidos”, se llama al método **toggleSoundDialog**, para manejar la música reproduciéndola, parándola, subiéndola o bajándola. Esta opción abre el menú de la Figura 35, para que se realice lo ya mencionado.

Figura 35

Pantalla que muestra el control de la música en GameActivity



Nota: Imagen de elaboración propia.

A continuación se explican los métodos importantes de **MusicManager**:

- *playMusic*. Este método es el encargado de manejar la reproducción de la música, y se utiliza cada vez que se quiere hacer esto. Se utilizan distintas variables como son: **context**, es de la clase estándar de “Android” **Context**, para dar un contexto al método de donde se esta ejecutando; **resId**, que es un entero identificativo de la música que se va a reproducir; **mediaPlayer**, que es de la clase estándar de “Android” **MediaPlayer**, y que se utiliza para la reproducción del archivo de la música; **preferences**, que es de la clase estándar de Android **SharedPreferences**, y que usa la constante que hace de clave **PREFS_NAME**, para guardar las preferencias relacionadas con la música como el volumen y si es la primera vez que se reproduce; **isFirstPlay**, que es un booleano al que se le asigna la constante que hace de clave **PREF_FIRST_PLAY**, para saber si es la primera vez que se ejecuta la música; **volume**, que es un float que almacena el volumen con la constante que hace de clave **PREF_VOLUME** y **resumePosition**, que es una constante de tipo entero para guardar el momento exacto donde se encontraba la música antes de pausarla. El procedimiento es: si **mediaPlayer** esta vacío, se le asigna la música. Se establecen las preferencias para la música y se configura un booleano para saber si es la primera vez que se reproduce la música. Si lo es, se le establece un volumen, si no, se recoge el volumen previamente guardado. Se reanuda la música desde el último punto y se inicia.
- *pauseMusic*. Este método es el encargado de pausar la música y se usa siempre para este fin. Se utilizan las variables explicadas en el anterior método de: **mediaPlayer** y **resumePosition**. El procedimiento es: si hay música en **mediaPlayer** y esta está sonando, se guarda el momento en el que esta se encuentra en **resumePosition** y se pausa.
- *isPlaying*. Este método es el encargado de devolver si la música se esta reproduciendo y que se usa siempre para ello. Se usa la variable ya explicada de **mediaPlayer** y el procedimiento es comprobar que esta no está vacía y si se está reproduciendo, devolviendo entonces “true”; si no, devuelve “false”.

Aunque no son métodos con mucho contenido o utilización, cabe mencionar: **releaseMusic**, en el cual se liberan los recursos asociados a **mediaPlayer** si este tenía recursos asociados; **setVolume**, que tiene dos versiones, el primero recibe un volumen y, si existe música asociada, se establece ese volumen, y el segundo usa este primer **setVolume** y se guarda en las preferencias el volumen; y **getVolume**, que devuelve el volumen almacenado en las preferencias. En ambos casos, el manejo se hace a través de la constante que hace de clave **PREFS_NAME** para identificar que es la preferencia de la música y **PREF_VOLUME** para guardar el volumen.

Es importante mencionar más en detalle las clases **MediaPlayer** y **SharedPreferences**.

- *MediaPlayer*, permite la reproducción de audio y video y proporciona una gran cantidad de métodos para manejar el control de estos. [46]
- *SharedPreferences*, permite almacenar datos de manera persistente usando un par clave-valor. Además, este se usa para almacenar configuraciones. [47]

Se han usado ambas juntas, y para ello se ha consultado como se debía hacerlo en esta web, [48].

También es importante mencionar que la música que utilizada se ha sacado de este video de “Youtube” [49], y es del Album 1 de “Brian Eno”.

4.5.7. Pantalla principal, Explicación del juego y Otros

Esta parte del diseño se refiere a la pantalla principal con las distintas funcionalidades que tiene, como la explicación de como jugar y otras partes del diseño que son muy pequeñas o no son muy importantes de explicar. Por ello, se explicaran brevemente las que más merecen mención, aunque de una manera simple.

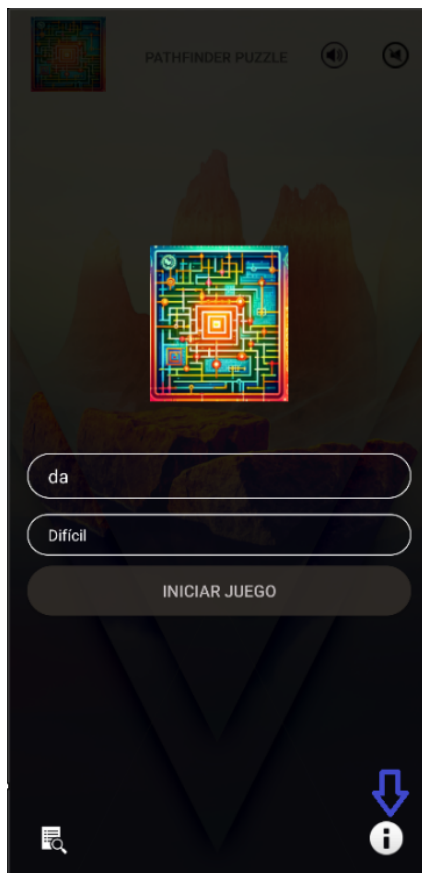
Es importante mencionar que la interfaz de la activity **MainActivity** esta basada en otra interfaz creada para una trabajo previo [50].

De la pantalla principal, es importante mencionar el método ya explicado de **MainActivity** que controla las pulsaciones. La parte más importante de explicar que aún no se ha detallado es la siguiente:

La pulsación del botón con id **helpAppButton**, la cual lanza el método **launchHelpActivity**, que inicia la activity **HelpActivity** con la explicación de cómo jugar, la cual se muestra en la Figura 36. Para esto, también se utiliza el método **checkFirstRun**, el cual, gracias a una variable **settings** de la clase estándar de “Android” ya explicada **SharedPreferences**, asignándole de preferencia con una constante que hace de clave para identificar si es la primera vez que se arranca la aplicación, muestre también la explicación de como jugar y cambie la preferencia de que es la primera vez que se ejecuta la aplicación.

Figura 36

Pantalla Pricipal, se marca el botón para la explicación de como jugar



Nota: Imagen de elaboración propia.

También es importante mencionar la parte del diseño que muestra y oculta la solución óptima en los laberintos. Esto sucede cuando se pulsa el botón con id **show_solution**, el cual arranca un método llamado **showsolutionDialog**, que lanza una modal para mostrar la solución o ocultarla en el caso de estar mostrándose. Si se pulsa el botón para mostrar la solución, **showSolution**, que es la variable booleana que contiene si se muestra o no, se pone a “true”. En todos los demás casos en los que se utiliza y se le asigna valor, se pone a “false”, como cuando se mueve, ya que se quiere que en cualquier caso que no sea el haber pedido mostrar la solución, esta se oculte, incluyendo si el usuario se mueve.

5. Estudio de caso

5.1. Descripción del estudio de caso

Se han usado dos pequeños encuestas de satisfacción distintas, una para los niños que han probado la aplicación y otra para los docentes. Se ha utilizado la “escala de Likert” en la creación del formulario creado en ambos casos. [51]. Este formulario fue creado mediante Google Forms [37].

Para este estudio de caso, se ha buscado que niños de entre 8 y 10 años probaran la aplicación, estando estos niños entre el segundo y cuarto año de primaria, y siendo este el “target” al que iba dirigido la aplicación. También se ha buscado que docentes que imparten en los cursos de estas edades probaran la aplicación. Finalmente se ha podido contar con dos niñas de 9 y 10 años respectivamente, y con un docente que imparte clase en estas edades, lo cual ha servido para hacerse una pequeña idea.

Ambos formularios requerían introducir un nombre, que podía ser simple. Las preguntas para el **formulario de los niños** eran las siguientes:

- *¿Como de entretenida te ha parecido la aplicación?*. Esta pregunta es del tipo de la “escala de Likert” **De valor**, se utiliza para medir la experiencia que ha tenido el niño al usar la aplicación.
- *¿Con que frecuencia ha necesitado las ayudas proporcionadas en el juego?*. Esta pregunta es del tipo de la “escala de Likert” **De frecuencia o repetición**, se utiliza para medir si el niño ha necesitado las ayudas proporcionadas por el juego debido a la dificultad del laberinto.
- *¿Con que frecuencia ha necesitado ayuda de un adulto para las preguntas, en caso de tener que responderlas?*. Esta pregunta es del tipo de la “escala de Likert” **De frecuencia o repetición**, se utiliza para medir si el niño ha necesitado de ayudas de un adulto en las preguntas, en caso de haber llegado a ver las preguntas, ya que estas salen al fallar 2 y 3 veces el laberinto.
- *Evalúa tu grado de satisfacción con la aplicación en términos generales (Interfaz, Mecánicas, etc)*. Esta pregunta es del tipo de la “escala de Likert” **De satisfacción**, se utiliza para evaluar la satisfacción del niño con los distintos aspectos de la aplicación, como son la interfaz, la música, el funcionamiento del juego, etc.
- *¿Qué es lo que más te ha gustado de la aplicación? (Selecciona un máximo de 2)*. Esta pregunta no es de ningún tipo de la “escala de Likert” pero se ha propuesto para obtener un feedback de cuáles han sido los aspectos que mas han gustado, en este caso al niño que ha probado la aplicación.

Las preguntas para el **formulario de docentes** eran muy parecidas, salvo que a estos se les preguntaba con vista a lo que podía ofrecer a los niños. La única pregunta distinta era esta:

- *¿En qué medida estás de acuerdo con que esta aplicación puede ser útil en la educación de los niños?*. Esta pregunta es del tipo de la “escala de Likert” **De acuerdo**, se utiliza para saber si el docente esta de acuerdo en considerar útil esta pregunta.

Esta pregunta en el formulario de los docentes sustituye a las preguntas de: ¿Con que frecuencia ha necesitado las ayudas proporcionadas en el juego? y ¿Con que frecuencia ha necesitado ayuda de un adulto para las preguntas, en caso de tener que responderlas?, que se encuentran en el formulario de los niños.

Para el fomrulario de los niños se ha creado esta encuesta de satisfacción [52] y para el formulario de satisfacción de los docentes esta otra [53].

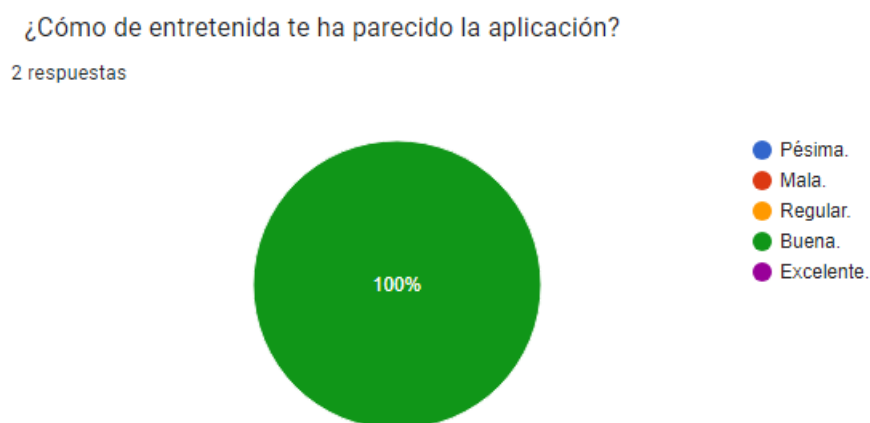
5.2. Análisis de los resultados

Es importante resaltar que los resultados no pueden ser del todo concluyentes, ya que no se ha contado con suficientes niños de la edad a la que iba dirigido, que pudieran probar la aplicación, al igual que no se ha contado con suficientes docentes.

Los resultados obtenidos de la encuesta de satisfacción de los niños son los correspondientes a las figuras: Figura 37, Figura 38, Figura 39, Figura 40 y Figura 41.

Figura 37

Resultado de la primera pregunta a niños



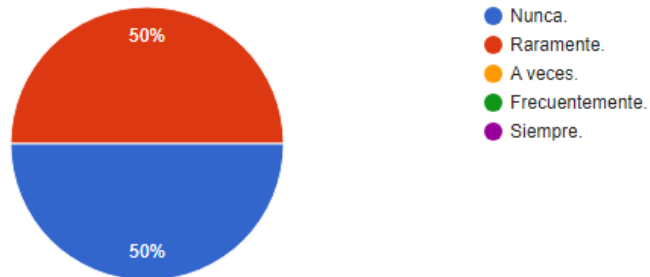
Nota: Imagen de elaboración propia.

Figura 38

Resultado de la segunda pregunta a niños

¿Con que frecuencia ha necesitado ayuda de un adulto para las preguntas, en caso de tener que responderlas?

2 respuestas



Nota: Imagen de elaboración propia.

Figura 39

Resultado de la tercera pregunta a niños

¿Con que frecuencia ha necesitado ayuda de un adulto para las preguntas, en caso de tener que responderlas?

2 respuestas



Nota: Imagen de elaboración propia.

Figura 40

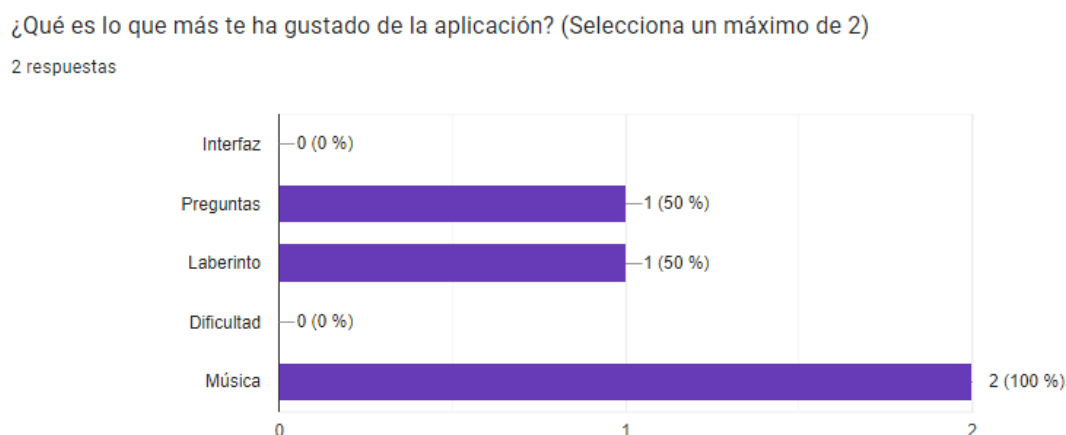
Resultado de la cuarta pregunta a niños



Nota: Imagen de elaboración propia.

Figura 41

Resultado de la quinta pregunta a niños



Nota: Imagen de elaboración propia.

De estos resultados se puede apreciar que a los niños les ha gustado la aplicación, sobre todo la ambientación que se le da al acivar la música de fondo, los laberintos aleatorios y las preguntas. La respuesta que se pudo obtener de la segunda y la tercera pregunta, viene en relación con la última, y es que a una de las niñas, la de 9 años, le pareció que los laberintos no eran difíciles pero eran entretenidos, pero las preguntas en algún caso no las entendía bien. La otra niña, la de 10 años, encontró fácil el laberinto y las preguntas le parecieron entretenidas pero no difíciles.

La idea de dirigir la aplicación a niños de entre 8 y 10 años surge porque las preguntas pueden ser difíciles en algunos casos, ofreciendo como recompensa un movimiento adicional al acertar. Esto busca equilibrar la dificultad, de modo que si el laberinto no presenta un gran desafío, las preguntas lo hagan.

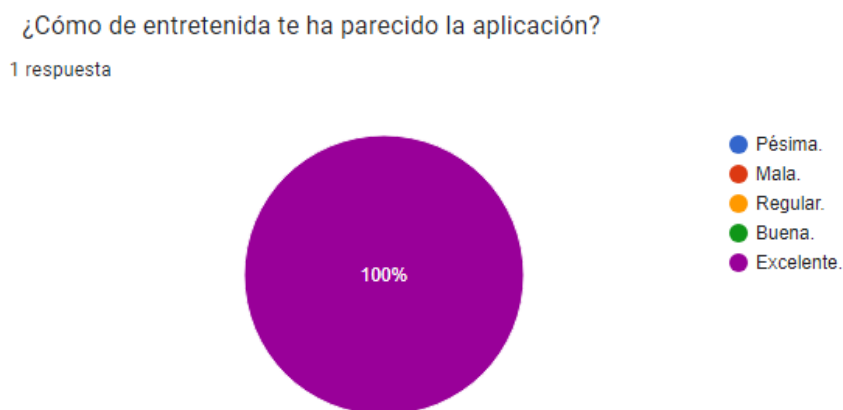
Para ajustar la dificultad y asegurar que la aplicación sea desafiante, se podría ampliar el rango de edad hacia niños de entre 6 y 10 años, incrementar la complejidad de los

laberintos y las preguntas, y ofrecer pistas para aquellas preguntas más difíciles. Además, sería útil probar la aplicación con un mayor número de niños para obtener resultados más concluyentes, ya que la muestra actual es insuficiente.

Los resultados obtenidos de la encuesta de satisfacción de los docentes son los correspondientes a las figuras: Figura 42, Figura 43, Figura 44 y Figura 45.

Figura 42

Resultado de la primera pregunta a docentes



Nota: Imagen de elaboración propia.

Figura 43

Resultado de la segunda pregunta a docentes



Nota: Imagen de elaboración propia.

Figura 44

Resultado de la tercera pregunta a docentes

Evalúa tu grado de satisfacción con la aplicación en términos generales (Interfaz, Mecánicas, etc)

1 respuesta



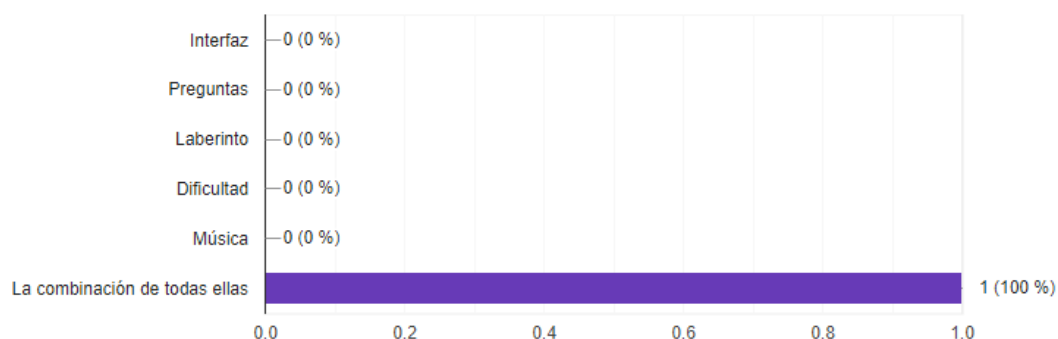
Nota: Imagen de elaboración propia.

Figura 45

Resultado de la cuarta pregunta a docentes

¿Qué es lo que más te ha gustado de la aplicación? (Selecciona un máximo de 2)

1 respuesta



Nota: Imagen de elaboración propia.

En cuanto a los resultados de la encuesta de satisfacción para los docentes se puede apreciar que la aplicación les ha gustado mucho y consideran que puede ser útil en la educación de los niños. En relación con la segunda pregunta, el docente que respondió a la encuesta indicó que la aplicación es útil para la educación de los niños porque los conceptos se explican de una manera entretenida y cómoda, facilitando su aprendizaje.

6. Conclusión y Lineas futuras

En este apartado se explican las conclusiones finales sacadas con el trabajo, así como si se han cumplido los objetivos y cómo se podría mejorar la aplicación.

6.1. Objetivos cumplidos

En cuanto al objetivo principal se ha llegado a la conclusión de:

- *Primer objetivo. Potenciar el desarrollo de competencias asociadas al pensamiento computacional.* Este objetivo se considera cumplido, ya que como se menciona, en el apartado del “Contexto” correspondiente 2.3 de esta memoria, se trabajan distintas habilidades del pensamiento computacional. Además, el personal docente que probó la aplicación considero que esta aplicación podía ser útil en la educación de los niños al considerar que trabajaba de una forma adecuada estos aspectos.

Y sobre los objetivos secundarios se ha llegado a la conclusión de:

- *Segundo objetivo. Crear una aplicación para dispositivos móviles que permite trabajar conceptos de pensamiento computacional en edades tempranas.* También se considera que este objetivo ha sido cumplido por las mismas razones que el objetivo principal.
- *Tercer objetivo. Aplicar mecánicas del juego en el diseño de la aplicación.* Se considera que este objetivo ha sido cumplido ya que se explica la relación con las mecánicas del juego y su implementación en el apartado del “Contexto” correspondiente 2.4 de esta memoria.
- *Cuarto objetivo. Propiciar un ambiente de interacción cómodo para el niño, con el fin de evitar el estrés.* Se considera que este objetivo ha sido cumplido, ya que durante el estudio, por observación directa se constato que los niños ponían la música y en ningún momento se mostraron estresados.
- *Quinto objetivo. Fomentar la resolución de laberintos como una herramienta para la comprensión de algoritmos típicos, como es el caso de “backtracking”.* Se considera que este objetivo ha sido cumplido ya que se explica el funcionamiento de backtracking de una manera cercana al usuario. Además de que el personal docente que probó la aplicación consideró que esta aplicación podía ser útil en la educación de los niños al considerar que esta explicaba bien los conceptos dentro de la aplicación.

6.2. Conclusiones

El pensamiento computacional es un concepto que mucha gente no conoce, o al menos bajo ese término. Este trabaja las habilidades ya mencionadas en el apartado del “Contexto” 2.3, y como se menciona en el punto 2.1 del mismo apartado, el pensamiento computacional trabaja una parte de STEM. En parte, es por esto por lo que el pensamiento computacional puede ser de gran utilidad en distintos aspectos de la vida, no solo para ayudar con la programación.

Al igual que las matemáticas o la ciencia se usan fuera de la tecnología, el pensamiento computacional, el cual trabaja parte de STEM (Science, Technology, Engineering

and Mathematics), también puede servir más allá del mundo tecnológico o incluso de la educación al cual se está aplicando en este caso. Su utilidad se extiende a cualquier área donde sea importante la resolución de problemas, el análisis sistemático o la capacidad de pensamiento crítico, como en la medicina, ayudando a descubrir patrones en un análisis; en los negocios, ayudando a analizar gran cantidad de datos de manera óptima; y en otras muchas áreas.

Antes de realizar este trabajo, no estaba completamente seguro de qué implicaba el pensamiento computacional. Ahora entiendo que es una habilidad crítica que debería ser integrada en la educación desde temprana edad. El pensamiento computacional no solo mejora la capacidad de resolver problemas y de pensar de manera lógica y estructurada, sino que también prepara a los estudiantes para enfrentar desafíos en diversos ámbitos de la vida, desde planificar un presupuesto familiar hasta diseñar un proyecto científico. Estas habilidades son fundamentales y aplicables en muchos aspectos de la vida diaria, por lo que su enseñanza en las escuelas debería ser esencial.

Este trabajo ha sido de gran ayuda para comprender lo que es verdaderamente el pensamiento computacional y cómo potencia diversas habilidades que muchas personas ya emplean. También ha facilitado la familiarización con la programación en Android, el uso de LaTeX para la creación de documentos, el trabajo con metodologías y el manejo de distintas bases de datos.

6.3. Líneas de trabajo futuras

Como mejoras que se podrían aplicar a la aplicación en un futuro, se proponen distintos aspectos como:

- Añadir más preguntas de cada dificultad a la base de datos, para que si juegas muchas partidas estas no se repitan.
- Añadir preguntas más difíciles o una nueva dificultad, pero añadiendo también la posibilidad de solicitar una pista para la pregunta, por si fuera muy complicada.
- Hacer que la explicación de lo ocurrido al fallar por primera vez en cada partida ocurra solo la primera vez que se hace en la partida o las primeras veces, para evitar que si se juegan muchas partidas se pueda hacer repetitivo.
- Añadir Realidad Aumentada (RA) a la aplicación, pudiendo además cambiar la vista desde la que se ve el laberinto, haciendolo más difícil, Esta mejora es una mejora muy grande, y llevaría muchísimo tiempo.
- Probar la aplicación con más usuarios, para así conseguir unos resultados concluyentes.

Bibliografía

- [1] CSIRO. *Bebras 2018 Solution Guide*. 2018. URL: <https://www.csiro.au/-/media/Digital-Careers/Files/Bebras-Files/Bebras-2018-Solution-Guide.pdf>.
- [2] ROBOTIX. *Pensamiento Computacional en el Aula: ¿Por qué es Importante?* S.f. URL: <https://www.robotix.es/es/blog/pensamiento-computacional-n284>.
- [3] Jeannette M. Wing. *Computational Thinking – Communications of the ACM*. 2006. URL: <https://cacm.acm.org/opinion/computational-thinking/>.
- [4] Mordechai Ben-Ari. *Teaching Computer Science: A Concise Approach*. London: Springer, 2001. ISBN: 978-1447102094.
- [5] JJ Peleato. *2.4 Backtracking — Programación, refactoriza tu mente*. S.f. URL: <https://docs.jjpeleato.com/algoritmia/backtracking>.
- [6] A Freeman, Samantha Adams Becker y M Cummins. *NMC/COSN Horizon Report: 2017 K-12 Edition*. 2017. URL: <https://www.learntechlib.org/p/182003/>.
- [7] Sevda Kucuk y Burak Sisman. “Students’ attitudes towards robotics and STEM: Differences based on gender and robotics experience”. En: *International journal of child-computer interaction* 23-24 (1 de jun. de 2020), pág. 100167. DOI: 10.1016/j.ijcci.2020.100167. URL: <https://www.sciencedirect.com/science/article/abs/pii/S2212868920300039?via%3Dihub>.
- [8] Jordi Adell Segura y Linda Castañeda Quintero. *El docente ante la innovación educativa: la tipología del profesorado universitario*. 2019. URL: https://repositori.uji.es/xmlui/bitstream/handle/10234/189983/adell_2019_Eld.pdf?sequence=1&isAllowed=y.
- [9] Educador. *Juegos de lógica para potenciar el pensamiento crítico en niños - PEREDA*. 24 de ene. de 2024. URL: https://iespereda.es/juegos-de-logica-para-potenciar-el-pensamiento-critico-en-ninos/?expand_article=1.
- [10] Julia Iriarte. *Pensamiento computacional y juegos de mesa - Bebé a Mordor*. 31 de ene. de 2023. URL: <https://bebeamordor.com/pensamiento-computacional-y-juegos-de-mesa/>.
- [11] *Scratch - about*. S.f. URL: <https://scratch.mit.edu/about>.
- [12] Adrián Rodríguez Mira. *Mecánicas de juego más habituales en los videojuegos*. 26 de feb. de 2024. URL: <https://www.tokioschool.com/noticias/mecanicas-de-juego-habituales-en-videojuegos/>.
- [13] *Manhattan Distance*. S.f. URL: <https://xlinux.nist.gov/dads/HTML/manhattanDistance.html>.
- [14] *Distancia Manhattan — Interactive Chaos*. 2024. URL: <https://interactivechaos.com/es/manual/tutorial-de-machine-learning/distancia-manhattan>.
- [15] *Euclidean distance*. S.f. URL: <https://xlinux.nist.gov/dads/HTML/euclidndstnc.html>.
- [16] Thomas H. Cormen et al. *Introduction to Algorithms*. 2.^a ed. Depth-first Search. MIT Press y McGraw-Hill, 2001. Cap. 22.3, págs. 540-549. ISBN: 0-262-03293-7.

- [17] *Metodologías de desarrollo de software: ¿qué son?* 4 de jun. de 2024. URL: <https://www.santanderopenacademy.com/es/blog/metodologias-desarrollo-software.html>.
- [18] B Boehm. “A spiral model of software development and enhancement”. En: *Software engineering notes* 11.4 (1 de ago. de 1986), págs. 14-24. DOI: 10.1145/12944.12948. URL: <https://doi.org/10.1145/12944.12948>.
- [19] *Qué es Trello: descubre sus funciones, usos y todo lo que ofrece* — Trello. S.f. URL: <https://trello.com/es/tour>.
- [20] Pablo Vizcaíno-Alcantud. *Miro: Pizarra Colaborativa*. 2021. URL: <https://economicas.ua.es/es/documentos/miro.-pizarra-colaborativa.pdf>.
- [21] Javier Gaspariño Muñoz. *Tablero Kanban*. 2024. URL: <https://trello.com/b/t09Xh7oc/tablero-kanban>.
- [22] *Android Developer Documentation*. 2024. URL: <https://developer.android.com/studio?hl=es-419>.
- [23] *About GitHub and Git*. S.f. URL: <https://docs.github.com/es/get-started/start-your-journey/about-github-and-git>.
- [24] *Git*. S.f. URL: <https://git-scm.com/>.
- [25] *Firestore*. S.f. URL: <https://firebase.google.com/?hl=es>.
- [26] D. Richard Hipp. *SQLite Home page*. 2024. URL: <https://sqlite.org/>.
- [27] *Sobre nosotros: la historia, los logotipos y los clientes de Trello* — Trello. S.f. URL: <https://trello.com/es/about>.
- [28] Produle. *MockFlow - Wireframe tools, prototyping tools, UI mockups, UX Suite*. S.f. URL: <https://mockflow.com/>.
- [29] *draw.io - free flowchart maker and diagrams online*. S.f. URL: <https://app.diagrams.net/>.
- [30] Sonar. *SonarCloud Online Code Review as a Service Tool*. S.f. URL: <https://www.sonarsource.com/products/sonarcloud/>.
- [31] *DeepL Translate - El mejor traductor del mundo*. S.f. URL: <https://www.deepl.com/es/translator>.
- [32] iLovePDF. *iLoveIMG — Herramientas online para editar imágenes*. S.f. URL: <https://www.iloveimg.com/es>.
- [33] iLovePDF. *Desde el 2010 iLoveIMG ofrece herramientas online para gestionar imágenes*. S.f. URL: https://www.iloveimg.com/es/ayuda/acerca_de.
- [34] *Pixelcut — Editor de fotos con IA gratuito*. S.f. URL: <https://www.pixelcut.ai/es>.
- [35] *Pixelcut*. S.f. URL: <https://create.pixelcut.ai/upscaler>.
- [36] Kaleido. *Quitar fondo a imagenes online y consigue fondos transparentes - remove.bg*. S.f. URL: <https://www.remove.bg/es>.
- [37] *Google Forms: Online Form Creator* — Google Workspace. S.f. URL: <https://www.google.com/forms/about/>.

- [38] *Plataforma de almacenamiento personal en la nube y uso compartido de archivos - Google*. S.f. URL: https://www.google.com/intl/es_es/drive/.
- [39] *QR Code Generator*. S.f. URL: <https://app.qr-code-generator.com/>.
- [40] *Benefits of using LaTeX editing software — Overleaf*. S.f. URL: <https://es.overleaf.com/about/why-latex>.
- [41] *LaTeX - A document preparation system*. S.f. URL: <https://www.latex-project.org/>.
- [42] Android Developers. *Bitmap*. 2024. URL: <https://developer.android.com/reference/android/graphics/Bitmap>.
- [43] Android Developers. *Canvas*. 2024. URL: <https://developer.android.com/reference/android/graphics/Canvas>.
- [44] Android Developers. *Paint*. 2024. URL: <https://developer.android.com/reference/android/graphics/Paint>.
- [45] latinyloco. *Moviles*. 2024. URL: <https://github.com/latinyloco/Moviles>.
- [46] Android Developers. *MediaPlayer*. 2024. URL: <https://developer.android.com/reference/android/media/MediaPlayer>.
- [47] Android Developers. *SharedPreferences*. 2024. URL: <https://developer.android.com/reference/android/content/SharedPreferences>.
- [48] *How to add sharedPreferences in an audio player mainactivity code*. 2020. URL: <https://stackoverflow.com/questions/62463907/how-to-add-sharedpreferences-in-an-audio-player-mainactivity-code>.
- [49] Brian Eno. *Ambient 1: Music for Airports [Full Album]*. 1978. URL: <https://www.youtube.com/watch?v=vNwYtllyt3Q>.
- [50] JavierGasparinoMunoz. *GitHub - JavierGasparinoMunoz/TFG3*. 2020. URL: <https://github.com/JavierGasparinoMunoz/TFG3>.
- [51] Melissa Hammond. *Escala de Likert: qué es y cómo utilizarla (incluye ejemplos)*. 22 de feb. de 2023. URL: <https://blog.hubspot.es/service/escala-likert>.
- [52] Javier Gaspariño Muñoz. *Encuesta de satisfacción Pathfinder Puzzle*. 2024. URL: <https://forms.gle/XY3eK6XXg5ukth5XA>.
- [53] Javier Gaspariño Muñoz. *Encuesta de satisfacción Docentes*. 2024. URL: <https://forms.gle/zbMxgRSusfHZXKjr5>.