Universidad Michoacana de San Nicolás de Hidalgo
Facultad de Ciencias Físico Matemáticas
"Mat. Luis Manuel Rivera Gutiérrez"

Sistema de Aprendizaje Personalizable Basado en el Juego del Memorama

Por

Leonardo David Saucedo León

Tesis para optar al grado de Licenciado en Ciencias Físico-Matemáticas

Asesora	:	**Karina Mariela Figueroa Mora**
Sinodales	:	José Gerardo Tinoco Ruiz
	:	Jesús Ortiz Bejar
	:	José Carlos Cortés Zavala
	:	Luis Valero Elizondo

Morelia, Michoacán, México
Enero del 2021

Agradecimientos

Primero que todo quiero dar las gracias a todas la personas que no confiaron en mí y me recomendaron abandonar la carrera ya que eso fue mi principal motor para no desistir de este maravilloso sueño.

Quiero dar las gracias a mi padre Juan Ramón Saucedo Pereyra por los grandes consejos y educación que me dio, a mi madre Martha Mónica León Cano por siempre confiar en mí y apoyarme incondicionalmente a lo largo de este trayecto. A mi padrino Julio Cesar Vargas García por haber sido mi guía y haber puesto en mí todas sus esperanzas.

A mis hermanos Mónica Beatriz Saucedo León, Juan Ramón Saucedo León y Nancy Saray Saucedo León, por todo el apoyo brindado a lo largo de estos años y el esfuerzo que hicieron para hacer esto realidad.

A Marco Antonio Esquivel Guillen, José Luis Rojas Castañeda, Uziel Silva Espino y sus respectivas familias por haber soñado conmigo, haberme acogido como un miembro más de sus familias y siempre brindarme su amparo.

A mi asesora Karina Mariela Figueroa Mora por dirigir esta tesis, por los grandes consejos, las enseñanzas que dejo en mí y por ser una espectacular maestra.

También quiero agradecer a los alumnos y profesores del Instituto Soul Maya que contribuyeron con su participación y disponibilidad para la realización del proyecto. En especial agradecimiento a la alumna Mayra Virginia Aguilar Guzmán quien sirvió como inspiración para la creación del proyecto, te deseo de todo corazón que ilumines e inspires a más gente con tu luz como lo hiciste conmigo.

Así mismo quiero dar gracias a Luis Manuel Tinoco Guerrero por haberme dado la idea para el nombre de la aplicación y realizar el logo, y a Ariana Geraldine Arreategui Tume quien ayudo con los diseños usados.

De igual manera quiero agradecer a todos aquellos que contribuyeron con su grano de arena para verme llegar un poco más lejos, quienes soñaron conmigo y tenían tantos anhelos como yo de verme culminar esta etapa. A todos les estaré eternamente agradecido y espero no haberlos defraudado. Infinitas GRACIAS.

A la memoria de Enrique Rodríguez Castillo, Maria de Lourdes Guerrero Magaña, Maria Sánchez García y Juan Carlos Rafael... Siddhartha.

Contenido

Resumen **1**

Abstract **1**

1 Introducción **2**

1.1 Motivación 2

1.2 Juegos Existentes 3

2 Marco Teórico **7**

2.1 Gamificación 7

2.2 Usabilidad 9

2.3 Base de datos 10

2.3.1 FireBase Database 10

2.4 Android Studio 11

3 Propuesta **13**

3.1 Diseño de la base de datos 13

3.2 Diseño de navegación 15

3.2.1 Opción: Mis juegos 15

3.2.2 Opción: Mis puntos 15

3.2.3 Opción: Perfil 16

4 Desarrollo **18**

4.1 Desarrollo de la pantalla de juego 18

4.1.1 Desarrollo de la pantalla de captura 20

4.2 Base de datos Firebase 28

4.3 Diseño final de la aplicación 36

4.4 Elementos finales 43
4.5 Vista de la puntuación 48
4.5.1 Opcion “Borrar” 48
4.5.2 Editar un juego 51
4.5.3 Opcion “Mis puntos” 55
4.5.4 Modificación de los datos de Perfil 55
4.6 Restricción en la navegación entre pantallas en la aplicación 58
4.7 Sistema Multijugador en tiempo real 60

5 Experimentación **75**

5.1 Analizando al Usuario con el rol de Docente 75
5.2 Analizando al Usuario Alumno 85

6 Conclusiones **94**

Índice de figuras

1.1 Juego de relacionar mosaicos (educaplay). 4

1.2 Juego de relacionar columnas (educaplay). 4

1.3 Juegos de relacionar (cerebriti). 5

1.4 Juegos de relacionar (dibujosparapintar). 6

3.1 Propuesta de diseño de la base de datos 14

3.2 Propuesta de diseño de navegación 17

4.1 Apariencia del frame. 19

4.2 Objeto datos. 20

4.3 Codigo del adaptador. 21

4.4 Parte del código donde se ejecuta el juego. 22

4.5 Contrucción de la base de datos. 23

4.6 Primera parte de los métodos de la base de datos. 23

4.7 Segunda parte de los métodos de la base de datos. 24

4.8 Vista para agregar preguntas y guardar un juego. 25

4.9 Código para agregar preguntas y guardar un juego. 26

4.10 Vista para agregar preguntas y guardar un juego. 27

4.11 Código del método “armado()“. 28

4.12 Primera parte para agregar Firebase a un proyecto Adroid. 29

4.13 Segunda parte para agregar Firebase a un proyecto Adroid. 29

4.14 Tercera parte para agregar Firebase a un proyecto Adroid. 29

4.15 Objeto “Persona“. 30

4.16 Objeto “Pregunta“. 31
4.17 Objeto “Relacion“. 31
4.18 Carpeta para crear menus. 31
4.19 Vista con el meú incluido. 32
4.20 Método para agregar preguntas/respuestas con validación de campos. 33
4.21 Método listaPreguntas. 34
4.22 Método agregarPreguntas. 34
4.23 Vista de inicio. 35
4.24 Vista para registrarse. 35
4.25 Método “Entrar”. 36
4.26 Vista una vez que el usuario entra. 37
4.27 Vista cuando seleccionamos un juego. 38
4.28 Rediseño de la pantalla de inicio. 39
4.29 Rediseño de la pantalla del usuario. 40
4.30 Método implementado para llenar la lista de puntuaciones. 42
4.31 Vista una vez terminado el juego. 44
4.32 Clase de la vista mostrada en la imagen anterior. 45
4.33 Vista para recuperar la contraseña. 46
4.34 Método para recuperar la contraseña y mandar el correo. 47
4.35 Vista de opciones para cada juego. 49
4.36 Programación de cada una de las opciones. 50
4.37 Método para borrar preguntas. 52
4.38 Opciones para editado. 53
4.39 Vista al momento de editar. 53
4.40 Métodos para borrar preguntas/respuestas. 54
4.41 Programación del botón de regreso. 55
4.42 Borrar una puntuación. 56
4.43 Vista de la configuración del perfil. 56
4.44 Métodos para la configuración del perfil. 57

4.45 Primera parte del proceso para borrar un perfil. 58
4.46 Segunda parte del proceso para borrar un perfil. 59
4.47 Programación del botón de regreso al momento de terminar un juego. 59
4.48 Objeto “Multijuego“. 62
4.49 Opciones de juego. 63
4.50 Confirmación de la opción. 63
4.51 Método “agregarPreguntas()“ adaptado. 63
4.52 Método “ResultadoConfirm()“adaptado. 64
4.53 condición agregada al metodo “Resultado2()“. 64
4.54 Vista de espera. 65
4.55 Métodos para terminar el turno de un usuario. 66
4.56 Código para hacer los movimiento y decidir turnos. 67
4.57 Método “fin()” adaptado. 68
4.58 Vita para un juego finalizado. 69
4.59 Restricción del botón de regreso. 70
4.60 Vita para salir del juego. 71
4.61 Primera parte del método “reduccion()“. 72
4.62 Segunda parte del método “reduccion()“. 72
4.63 Tercera parte del método “reduccion()“. 73
4.64 Logo. 73
4.65 Vista de inicio rediseñada. 74

5.1 Encuesta de diagnóstico para profesores. 76
5.2 Perfil de español. 77
5.3 Perfil de biología. 78
5.4 Examen 1 de química. 79
5.5 Examen 2 de química. 80
5.6 Examen 1 de biología. 81
5.7 Examen 2 de biología. 82
5.8 Examen de español. 83

5.9 Encuesta de satisfacción para profesores. 84
5.10 Gráfica de las calificaciones obtenidas en español antes de usar la app. 86
5.11 Gráfica de las calificaciones obtenidas en química antes de usar la app. 86
5.12 Gráfica de las calificaciones obtenidas en biología antes de usar la app. 87
5.13 Encuesta de diagnóstico para alumnos. 88
5.14 Gráfica de las calificaciones obtenidas en español después de usar la app. 89
5.15 Gráfica de las calificaciones obtenidas en química después de usar la app. 90
5.16 Gráfica de las calificaciones obtenidas en biología después de usar la app. 90
5.17 Encuesta de satisfacción para alumnos 91
5.18 Encuesta de satisfacción para alumnos. 92

Resumen

Los juegos siempre han sido un recurso para que los profesores atraigan la atención de sus estudiantes, una vez con ella es posible aprovechar ese entusiasmo y canalizarlo hacia el aprendizaje. También es por todos conocido que existen miles de juegos modernos de distintos tipos, niveles de avance y público objetivo. Sin embargo pocos juegos se pueden personalizar con un tipo de material académico muy específico. En este artículo describimos el diseño, adaptación y creación moderna de un juego clásico conocido por todos: el memorama.

Palabras clave: Vista, Memorama, AndrioidStudio, Usuario, Firebase

Abstract

Games have always been a resource for teachers to attract the attention of their students, once with it it is possible to take advantage of that enthusiasm and channel it towards learning. It is also well known that there are miles of modern games of different types, levels of advancement and target audience. However, few games can be customized with a very specific type of academic material. In this article he describes the modern design, adaptation and creation of a classic game known to all: memorama.

Keywords: View, Memorama, AdroidStudio, User, Firebase

Capítulo 1

Introducción

1.1 Motivación

Los mexicanos tenemos la creencia de que las matemáticas son algo complicado de entender, al menos en 2018, el 70% de los estudiantes habían reprobado dicha materia en algún momento de su vida escolar[Exc20], En matemáticas, así como en cualquier área, las definiciones son fundamentales para su entendimiento, pues estos nos dan las herramientas para poder llevar acabo la resolución de problemas. Algunas veces la falta de explicación de los mismos o el no resaltarlo lo suficiente es lo que puede llegar a provocar que un alumno tenga déficit de alguna materia en cuestión creando una especie de laguna que si no es atendida a tiempo le llega a repercutir con el tiempo durante su trayecto escolar [Gar08].

Mucho de esto se puede prevenir si desde que se detecta ese problema es atacado, hasta la fecha se sigue haciendo uso de los mismos métodos antiguos de aprendizaje, actualmente las nuevas generaciones han dejado de optar por estos métodos ya que la cultura ha ido evolucionando y viejas prácticas o costumbres se han quedado atrás.

Afortunadamente ese paso del tiempo también ha influido en una herramienta que cada día toma más poder como lo es el caso de la tecnología. Hemos ido adaptando nuestro estilo de vida a esta gran revolución que nos ha permitido hacer las mismas cosas de antes pero con una mayor facilidad y comodidad.

La tecnología permite que al combinar un par de ideas se pueda crear algo completamente nuevo e innovador que nos ayude a resolver nuestros problemas en la vida cotidiana, dentro de la gamificación existen ejemplos que combinan la enseñanza con las tecnologías haciendo llamativo el aprendizaje. Entonces podemos hacer uso de ello para crear una herramienta que nos facilite la memorización de conceptos no solo en matemáticas si no en cualquier área del ámbito escolar, algo que sea atractivo, innovador y que el alumnos puedan asociar de manera rápida por estar

familiarizado con ello.

El memorama es un juego muy popular y conocido dentro de la cultura mexicana, el cual consiste en una serie de cartas donde dos tarjetas tienen la misma imagen, éstas se encuentran con la imagen oculta con el objetivo es ir encontrando cada par, en cada intento se quiere memorizar la posición donde se encuentra cada tarjeta que ha sido destapada, solo es posible voltear 2 tarjetas en cada turno puede hacerse uso de esta dinámica para asociarlo con el aprendizaje, donde una de las tarjetas contenga el concepto y el par sea su respectiva definición, aumentando el grado de dificultad del juego y haciendolo más llamativo tanto como sistema de entretenimiento como sistema de aprendizaje aprovechando que hoy en día al menos 64.7 millones de jóvenes tienen smartphone en México [Exp] se puede tener un gran alcance desarrollando esta idea en forma de aplicación móvil.

1.2 Juegos Existentes

Hasta la fechan no hemos encontrado aplicaciones desarrolladas en la PlayStore (Tienda oficial de las aplicaciones para el sistema Andriod) que funcione de la manera aquí planteada o que cuente con algún juego interactivo de esta manera. Sin embargo como aplicaciones via web se pueden encontrar diversas páginas con ideas tangentes al objetivo de este trabajo. Por ejemplo se puede encontrar en educaplay [Edu17] diversos juegos, el más parecido a este trabajo es el llamado *relacionar mosaicos* [edua] en donde presentan tarjetas ocultas con imágenes de reverso las cuales se tienen que hacer click para revelar su contenido y hacer lo mismo con otra tarjeta, si existe alguna relación entre las imágenes estas desaparecen repitiendo este procedimiento hasta terminar con los mosaicos, vease la figura 1.1, además, tiene implementado un apartado donde se puede crear mosaicos con sus respectivas imágenes o texto creando la relación que indica que son la pareja buscada también, se puede consultar los juegos creados por otros usuarios y resolverlos, entre los temas más comunes se encuentran lecciones para fortalecer el aprendizaje del idioma inglés.

Otro de sus juegos con una temática similar a la planteada en este trabajo es el de *relacionar columnas* [edub] donde se muestran imágenes y al hacer al hacer click sobre la imagen de una columna, ésta se selecciona para posteriormente hacer lo mismo en la otra columna seleccionado la imagen que se cree es la respuesta. Si la respuesta es acertada se dibuja una línea entre las correspondientes imágenes, como se muestra en la figura 1.2. Este sitio permite interactuar con los juegos creados por otros usuarios y crear relaciones de manera similar al juego anteriormente mencionado [edu20].

Otra página donde también es posible encontrar este tipo de actividades es en cerebriti [Cer18], donde dentro de la variedad de formas de juego que poseen tienen una donde se relacionan

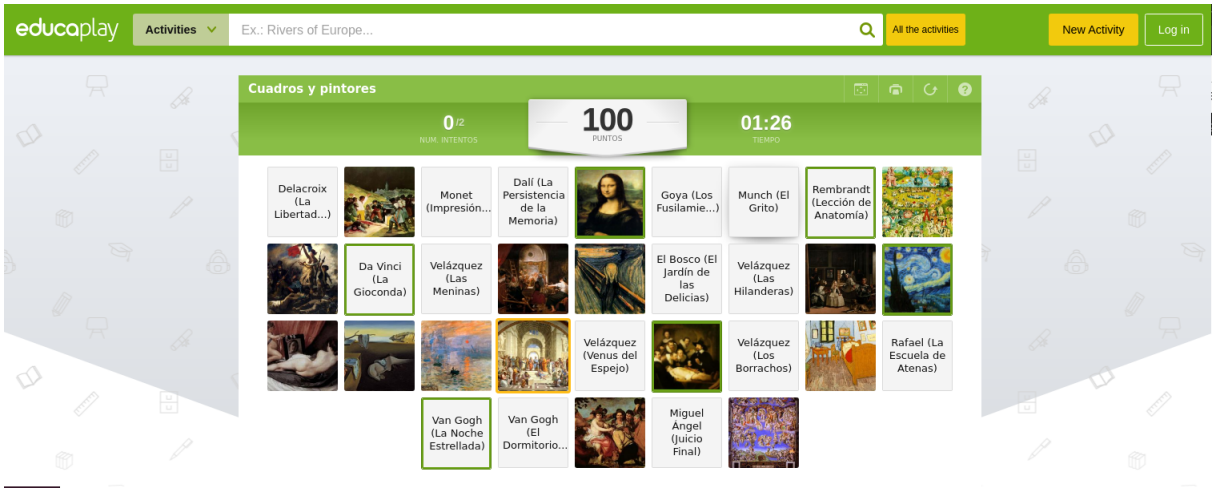


Figura 1.1: Juego de relacionar mosaicos (educaplay).

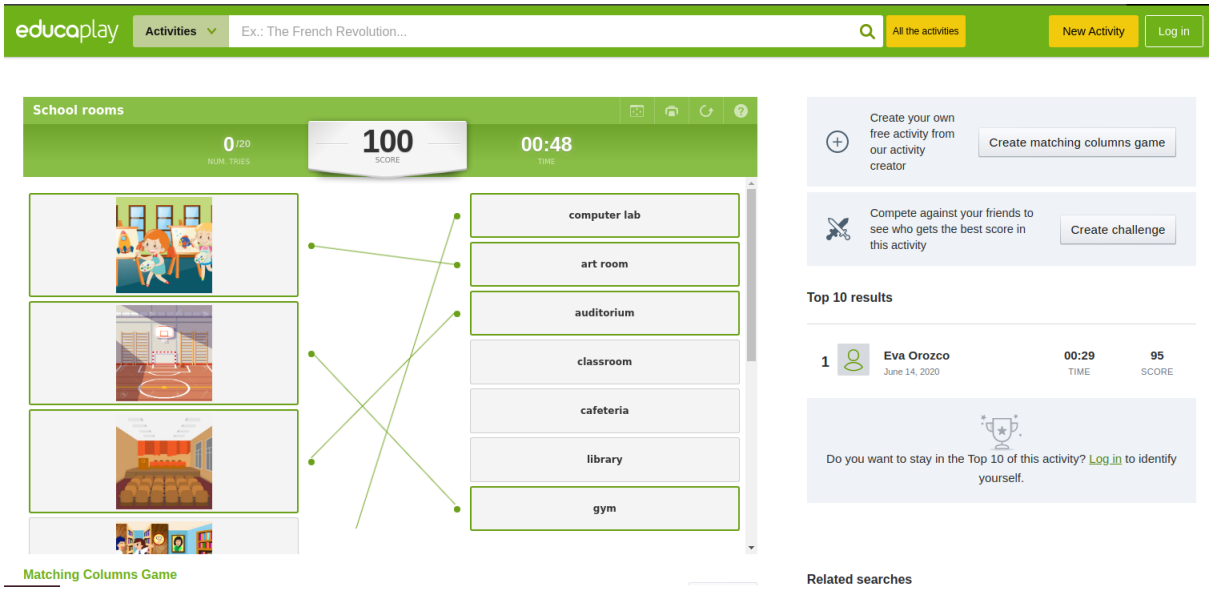


Figura 1.2: Juego de relacionar columnas (educaplay).

imágenes con textos [Cer20b] como se muestra en la figura 1.3 donde del lado izquierdo aparece una variedad de imágenes mientras que del lado derecho aparece un conjunto de palabras, para jugarlo se pone el cursor sobre alguna de las palabras, se hace click sobre ella y se arrastra hasta la imagen que se cree corresponde a la respuesta correcta, de ser acertada tanto el texto como la imagen desaparecen, en caso de no ser así el texto regresa a su respectiva posición. Se termina al desaparecer todas las imágenes, cuenta con un listado de juegos creados por la comunidad y también es permitido crear juegos [Cer20a].

Figura 1.3: Juegos de relacionar (cerebriti).

En la figura 1.4 se muetra otra aplicación con la misma forma de juego que la de relacionar columnas mencionada anteriormente en la página dibujosparapintar [Dib18] en su apartado *juegos de relacionar* [pp] de igual manera como se procedia en el juego anterior se selecciona una de las opciones dadas del lado derecho para después seleccionar otra del lado izquierdo de ser correcto se dibujaba la línea entre ellas y en caso no serlo no sucede tal evento, a diferencia del otro éste solo permite introducir texto en ambas columnas al momento de crear un juego. Existe una gran variedad creados por los usuarios.

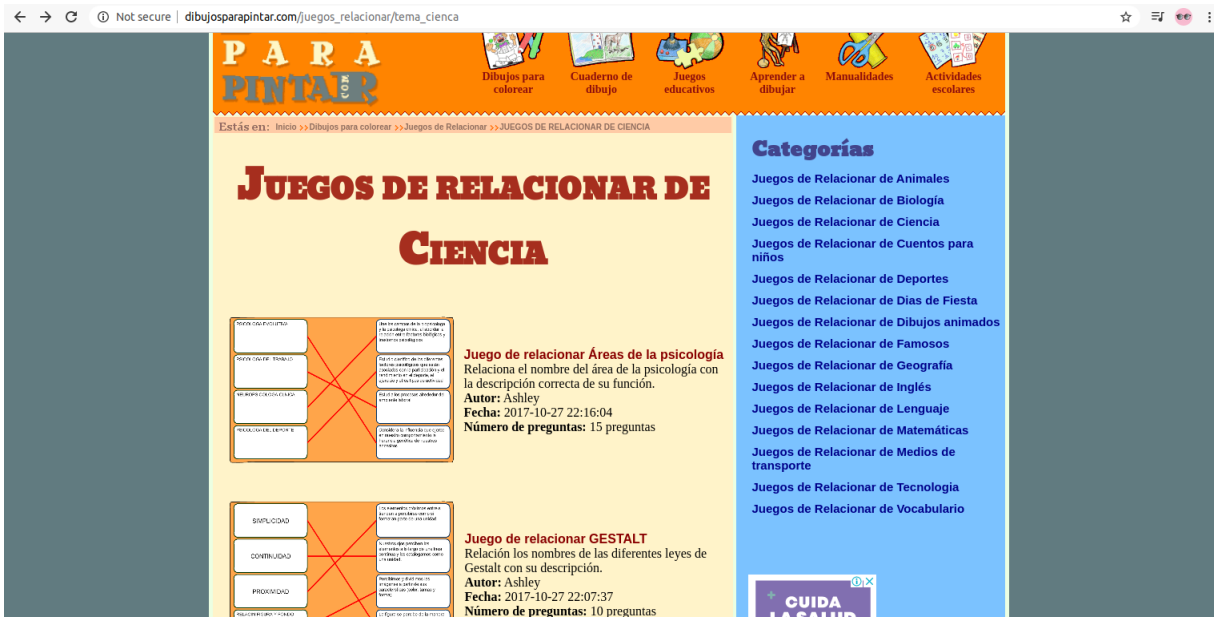


Figura 1.4: Juegos de relacionar (dibujosparapintar).

Capítulo 2

Marco Teórico

En este capitulo mostraremos algunos conceptos fundamentales de este trabajo. Primero se comenzará explicando el concepto de gamificación y usabilidad depués de explicará la metodología usada, así como la tecnología con que se desarrollará la aplicación.

2.1 Gamificación

La Gamificación es una técnica de aprendizaje que traslada la mecánica de los juegos al ámbito educativo-profesional con el fin de conseguir mejores resultados, ya sea para absorber mejor algunos conocimientos, mejorar alguna habilidad, o bien recompensar acciones concretas, entre otros muchos objetivos [Gai20].

El modelo de juego realmente funciona porque consigue motivar a los alumnos, desarrollando un mayor compromiso de las personas, e incentivando el ánimo de superación. Se utilizan una serie de técnicas mecánicas y dinámicas extrapoladas de los juegos [Gai20].

De acuerdo con [Gai20] la técnica mecánica es la forma de recompensar al usuario en función de los objetivos alcanzados. Algunas de las técnicas mecánicas más utilizadas son las siguientes:

- Acumulación de puntos: Se asigna un valor cuantitativo a determinadas acciones y se van acomulando a medida que se realizan.
- Escalado de nivel: Se definen una serie de niveles que el usuario debe ir superando para llegar al siguiente.
- Obtención de premios: A medida que se consiguien diferentes objetivos se van entreando

premios a modo de colección.

- Regalos: Bienes que se dan al jugador de forma gratuitua al conseguir un objetivo.
- Clasificaciones: Clasificar a los usuarios en función de puntos u objetivos logrados, destacando a los mejores.
- Desafíos: Competiciones entre los usuarios, el mejor obtiene los puntos del premio.
- Misiones o retos: Conseguir resolver o superar un reto u objetivo planteado.

También [Gai20] meciona que las técnicas dinámicas hacen referencia a la motivación del propio usuario para jugar y seguir adelante en la consecución de sus objetivos. Algunas de las técnicas dinámicas más utilizadas son las siguientes:

- Recompensa: Obtener un beneficio merecido.
- Estatus: Establecerse en un nivel jerárquico social valorado.
- Logro: Como superación o satisfacción personal.
- Competición: Por el simple afán de competir e intentar ser mejor que los demás.

De acuerdo con [Mol20] el proceso de gamificación es el siguiente:

- Viabilidad: En primer lugar hay que valorar si la gamificación es aplacable al contenido que se quiere enseñar en el aula.
- Objetivos: Hay que definir cuáles serán los objetivos de la gamificación.
- Motivación: Otro aspecto a valorar es la predisposición y el perfil de un grupo para llevar a cabo la gamificación en una actividad.
- Implementación: Se trata de sopesar qué relación existe entre la gamificación y el contenido que se enseña de una materia.
- Resultados: Es imprescindible realizar una evaluación de los resultados de la propuesta de gamificación que se haya llevado a cabo.

Los objetivos que persigue cualquier actividad en el ámbito de la gamificación según [Mol20] son:

- Fidelización: La gamificación establece un vínculo del alumno con el contenido que se está trabajando cambiando la perspectiva que tiene del mismo.

- Motivación: La gamificación quiere ser una herramienta contra el aburrimiento de determinados contenidos aplicados en el aula.
- Optimización: Por optimización se entiende el hecho de recompensar al alumno en aquellas tareas en las que no tienes previsto ningún incentivo.

2.2 Usabilidad

Usabilidad es un vocablo que no pertenece al diccionario oficial de la Real Academia Española (RAE) pero es algo muy frecuente dentro del ámbito de la informática y la tecnología, este concepto viene de el inglés usability, que hace referencia a la facilidad con la que un usuario es capaz de utilizar una herramienta diseñada [yMM20].

[Man20] menciona que la usabilidad es algo muy importante al momento del desarrollo de una tecnología. La usabilidad es la herramienta que nos permite hacer una aplicación o página sea sencilla para el usuario, intuitiva y cómoda, siendo el elemento mas importante el usuario debe poder interacutar con la aplicación aún cuando sea muy novedosa o revolucionaria.

Benevan [Man20] dice que la usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso.

La usabilidad es de uno de los principios más importantes dentro del desarrollo pues la intención de lo que es creado es que sea por y para el usuario, ésta se encuentra vinculada a la simplicidad, la facilidad, la comodidad y la practicidad. En otras palabras, se encuentra relacionada con la eficacia percibida de un objeto y la posibilidad de aprovechar todo su potencial.

Para que exista la usabilidad, la interacción hombre-máquina debe ser clara y sencilla, si existen deficiencias tanto en el aspecto material como en el lógico o virtual, la usabilidad se verá afectada. Sostiene que si el sistema no puede ser utilizado por el usuario en todo su potencial, no es útil y por lo tanto es deficiente, midiéndose todo esto a partir de pruebas empíricas.

La usabilidad exige revisar una serie de aspectos asociados con el uso y la forma en que las personas se relacionan con los sistemas que se le proporcionan. Las principales características de la usabilidad [Gar20] son las siguientes:

- Útil: Debemos preguntarnos si el producto y su sistema son útiles, si realmente estamos presentando una solución innovadora al problema en cuestión.
- Deseable: El producto debe ser eficiente con el empleo de sonidos, imágenes o animaciones los cuales deben estar equilibrados para darle constancia al proyecto.
- Usuable: Haciendo referencia a que este debe ser de fácil uso y que sirva como complemento con el resto de los aspectos.

- Accesible: Que sea lo más comprensible posible para cualquier persona sin importar si presentan una discapacidad.
- Valor: Que se ofrezca una herramienta valiosa para el usuario.

2.3 Base de datos

Una base de datos es un conjunto de datos relacionados, pertenecientes a un mismo contexto, agrupado y almacenados sistemáticamente para su posterior uso [Val20]. Podemos tomar como ejemplo particular las bibliotecas, pues estas están compuestas por documentos y textos impresos en papel y registrados para su consulta. Con el avance tecnológico que se ha alcanzado hoy en día la mayoría de las bases de datos se encuentran en formato digital permitiendo almacenar grandes cantidades de información en pequeños espacios físicos, siendo un componente muy importante y bastante útil al momento de tratar con almacenamiento de datos.

Existen programas denominados sistemas gestores de bases de datos o por su abreviatura SGBD (O conocuido en inglés cómo Database Management System o DBMS), que permiten almacenamiento para posteriormente acceder a ellos de forma rápida y estructurada, actualmente son automatizadas [Wik20]. Las bases de datos tienen una aplicación mayor en al ámbito de las instituciones públicas y de gestionamiento empresarial. éstas surgieron por la necesidad del ser humano de almacenar información, preservarla con el tiempo y su deterioro para poder acudir a ella posteriormente.

Cada base de datos se compone de una o más tablas que guarda un conjunto de datos. Cada tabla tiene una o más columnas y filas. Las columnas guardan una parte de la información sobre cada elemento que queramos guardar en la tabla, cada fila de la tabla conforma un registro.

Existen dos tipos de bases de datos basados en la variabilidad, la primera son las bases de datos estáticas, que son aquellas que contienen archivos para únicamente lectura cómo lo son los documentos o datos históricos, también utilizados para la inteligencia empresarial, y la segunda que son las bases de datos dinámicas que aquellas donde se guarda información que será en un futuro modificada, actualizada o borrada cómo por ejemplo las utilizadas en los centros comerciales.

2.3.1 FireBase Database

Según su propia definición, Firebase es un conjunto de herramientas orientadas a la creación de aplicaciones de alta calidad y al crecimiento de los usuarios. Creada por James Tamplin y Andrew Lee en 2012 fue adquirida por Google en 2014. Es una plataforma ubicada en la nube integrada

con Google Cloud Platform [Rui20].

Dentro de las ventajas que podemos encontrar es la fácil sincronización que se puede llegar a tener con ella sin tener que administrar las conexiones o escribir una lógica compleja para ello, se integra muy fácil a las aplicaciones móviles y a las grandes plataformas como IOS, Android o Unity, puede soportar ser usada desde plataformas pequeñas hasta muy grandes ya que usa infraestructura de Google permitiéndonos crear proyectos sin la necesidad de un servidor personal, ademas de brindar soporte técnico de manera gratuita.

Dentro de los servicios que podemos encontrar en ella tenemos una base de datos en Realtime la cual nos permite hacer modificaciones dentro de nuestras aplicaciones de manera inmediata, el servicio de autenticación lo cual nos simplifica el inicio de sesión y gestión, todo esto organizando nuestros datos dentro de un árbol JSON, otra de las ventajas que podemos encontrar es que si hacemos un cambio pero perdemos la conexión a internet este se almacena en el cache, para cuando volvamos a tener conexión el cambio sea realizado.

2.4 Android Studio

Android Studio es un entorno de desarrollo integrado (IDE) oficial para el desarrollo de apps para Android [Wik]. Además del potente editor de códigos y las herramientas para desarrolladores, Android Studio ofrece incluso más funciones que aumentan la productividad cuando se desarrollan apps para Android, como las siguientes:

- Un sistema de compilación flexible basado en Gradle.
- Un emulador rápido y cargado de funciones.
- Un entorno unificado donde puedes desarrollar para todos los dispositivos Android.
- Herramientas de Lint para identificar problemas de rendimiento, usabilidad y compatibilidad de la versión, entre otros.
- Compatibilidad con C++.

Además, permite realizar cambios de código y aplicar nuevos recursos a la aplicación sin necesidad de reiniciarla.

Este software fue diseñado para la creación de aplicaciones móviles con sistema operativo Android y, a día de hoy, es una herramienta imprescindible para cualquier desarrollador. Sin embargo, su uso se extiende mucho más allá de la creación, ya que permite realizar muchas otras acciones de suma importancia para el posterior desarrollo y actualización de la aplicación.Cada

proyecto de Android Studio incluye uno o más módulos con archivos de código fuente y archivos de recursos. Entre los tipos de módulos se incluyen los siguientes [Sou20]:

- Módulos de apps para Android.
- Módulos de biblioteca.
- Módulos de Google App Engine.
- Herramientas de Lint para identificar problemas de rendimiento, usabilidad y compatibilidad de la versión, entre otros.
- Compatibilidad con C++.

Además de las funciones y características que ya se han mencionado, cuenta con muchas ventajas.

Permite ejecutar las compilaciones de forma muy rápida, y así poder comprobar en el momento los fallos y mejoras de la aplicación. Realiza renderizados de layouts en tiempo real, y cuenta con la posibilidad de utilizar parámetros tools. Ejecuta la aplicación en tiempo real desde el propio teléfono móvil. Su potente emulador ayuda a comprobar el estado de la aplicación en el momento, sin necesidad de un ordenador.

También permite simular diferentes dispositivos y tabletas, pudiendo visualizarlas en un mismo entorno. De esta forma se puede trabajar varias aplicaciones simultáneamente y ver las partes de código necesarias de cada una.

Se puede crear elementos gráficos para la interfaz de la aplicación sin necesidad de utilizar el código. Esta función te ayudará a diseñar el aspecto visual de tu App de una forma muy sencilla.

Entre las desventajas podemos encontrar en este software es que requiere de una gran cantidad de recursos y gasta bastante batería. Así que, para que el emulador trabaje correctamente, se necesita un buen equipo de trabajo, con gran capacidad de RAM y espacio suficiente en tu disco duro [Est20].

Capítulo 3

Propuesta

3.1 Diseño de la base de datos

Para la creación de la base de datos se implementó el modelo relacional, peresentado en la figura 3.1 contando con 7 tablas las cuales fueron:

- Persona: Contiene el nombre del jugador, su apellido, correo, login (nickname o nombre de usuario) y un password; datos necesarios para el acceso a la aplicación, así como la identificación de los usuarios al momento de hacer una recopilación de datos durante el juego.

- Juego: Contiene los campos "titulo" donde se coloca un título al juego que hace referencia del tema que trataba, "nickname", "descripción" donde se coloca una pequeña reseña de la intención del juego, "fecha" campo en el que se almacena la fecha de creación, "pin" que es un número entero único que sirve para que los demás usuarios puedan encontrarlo dentro de la aplicación, seguido de los campos "nameMax" y "maxScore" que captura la máxima puntuación obtenida al jugar, así como el nickname de la persona que lo conseguio y por último la tabla "numVeces" donde se guarda el número de veces que ese juego era elegido. El propósito de los últimos 3 campos es recopilar la información necesaria para poder conocer las observaciones sobre el juego, ya fuese la dificultad y el interés de los jugadores, también sirviendo para saber qué temas se encontraban con menor comprensión.

- Preguntas: formada por los campos: "pregunta", "respuesta" e "imagen" las primeras dos tienen el contenido del juego y la parte más importante de este, ya que, ahí se encuentra todo el conocimiento que el alumno va a adquirir o en algunos casos reforzar, la intención del tercero es que el usuario pueda subir una foto o imagen donde se contiene la pregunta o respuesta para casos donde el teclado del teléfono no cuente con la capacidad de escribirla

(como en el caso de las formulas vistas en física, matemáticas o química por ejemplo) el cual esta en proceso de desarrollo en la aplicación.

- Tiene: Desempeña un papel muy importante como lo es la conexión entre las ́tablas “Preguntas” y “Juego” a través de sus campos “id_j” donde se almacena el id del juego creado y “id_p” que almacena el id de pregunta que pertenece al juego, ambos números generados por Firebase de manera automática, eso quiere decir que la tabla almacena que preguntas le corresponden a cada uno de los juegos.

- Jugar: Parte de la base de datos donde se almacena la información de cada partida que un jugador hace donde encontramos los campos “id_j” siendo el id del juego seleccionado, “nickName” el alias de la persona que lo jugó, “score” la puntuación que obtuvo y “tiempo” y “fecha” que guardan el tiempo que le tomó terminar el juego y la fecha en que lo hizo.

El díagrama mostrado en la Figura 3.1 corresponde al modelo relacional (ref a un libro dee bd) que puede almacenar la información que nos interesa como por ejemplo: permite crear a un usuario, saber quién a jugado qué juego, así como qué preguntas corresponden a cada juego. etc.

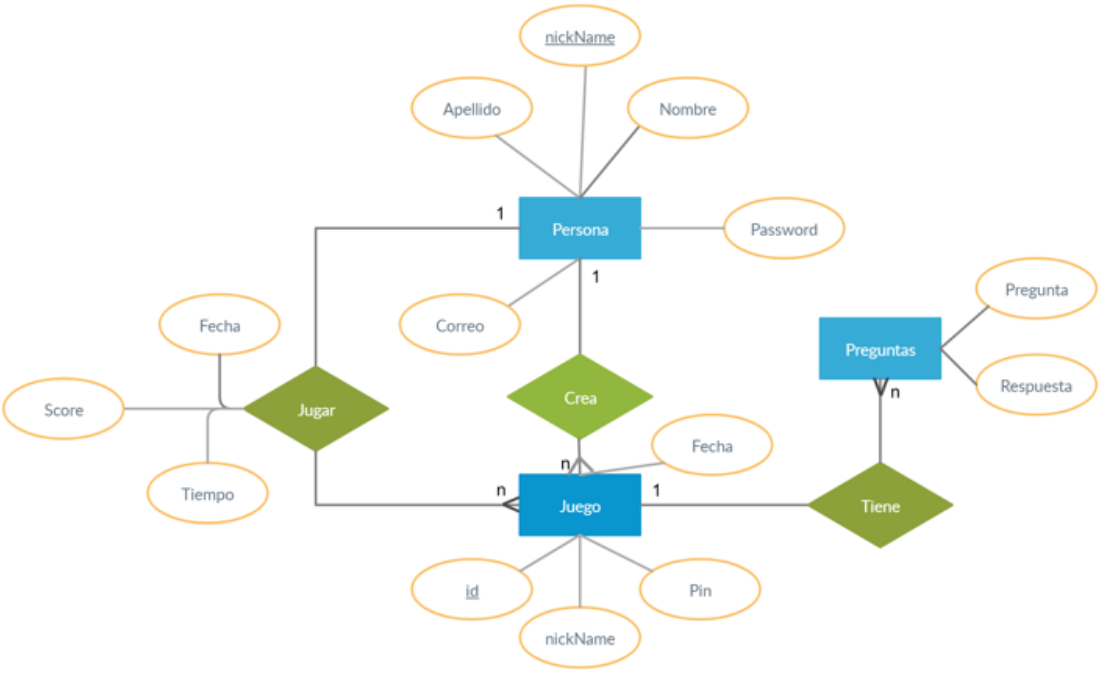


Figura 3.1: Propuesta de diseño de la base de datos

3.2 Diseño de navegación

La imagen 3.2 presenta la idea con la que fue pensada la interacción del usuario con la aplicación, contando como inicio la pantalla de "login" que es donde los usuarios ingresan su nickname y su contraseña para acceder a la aplicación, en caso de no estar registrados se cuenta con un botón que contiene una vista donde se pueden ingresar los datos para darse de alta y poder tener acceso. Una vez ingresados los datos de inicio de sesión podemos acceder a las siguientes 3 ventanas disponibles que son "Mis juegos", "Mis puntos" y "Perfil".

3.2.1 Opción: Mis juegos

Cuenta con una lista de los memoramas creados por el usuario formada por algunos detalles como lo son el nombre, pin, los datos del nickname que obtuvo la máxima puntuación y la fecha en que fue creado. Con esta vista a través de una serie de botones podemos acceder a diferentes opciones como lo son:

- Agregar un nuevo juego: Parte donde se añaden las preguntas con sus respectiva respuesta, se le asigna un nombre al juego y una ligera descripción de lo que trata.
- Ver juego: aquí el usuario puede ver listados los jugadores que han interactuado con su memorama, la fecha en que lo hicieron, la puntuación y el tiempo, también puede borrarlas, en esta misma vista se presentan las opciones de jugar ya sea en solitario u online con algún amigo,
- Editar un juego: Parte creada con la intención de que el usuario pueda modificar ya fuera alguna pregunta, respuesta o borrar ambas, también permitiendo la modificación del nombre y de la descripción.
- Eliminar un juego: Como el nombre lo describe es donde el usuario puede borrar sus cuestionarios creados.

3.2.2 Opción: Mis puntos

Como inicio muestra un listado de las puntuaciones obtenidas durante los diversos juegos con los que hemos interactuado, el tiempo que nos tomó completarlo y el nombre y puntuación de la persona que más alta calificación obtuvo; ya fuera por los memoramas que el mismo usuario ha creado o los creados por otros usuarios. Donde se incluyen las opciones de

- Borrar puntuación: Es utilizado para cuando el usuario quisiera quitar los datos de alguna de las partidas jugadas,

- Detalles: Mostrando los datos completos de la partida y un botón de búsqueda, esté botón permite nos lleva a 3 diferentes ventanas, ingresando el nickName de alguien podemos entrar a lo equivalente a la vista de "Mis juegos" de ese usuario (sin los permisos de editado o borrado), al ingresar el pin de un juego podemos entrar a lo mencionado como "Ver juego" que, como ya se comentó, nos permite jugar sólo o contra un rival. El tercer dato que podemos ingresar es un pin, una sala multijugador donde seremos redirigidos directamente a una vista donde comenzaremos a jugar con la persona que nos haya enviado el pin, dicho pin es proporcionado cuando una persona elige el modo multijugador dentro de la sección "Ver juego".

3.2.3 Opción: Perfil

Nos permite hacer un par de modificaciones a nuestra cuenta, la primera es cambiar la contraseña, la segunda es cerrar sesión y la tercera es borrar la cuenta, al momento de borrarla se eliminan automáticamente todo lo relacionado con nuestro perfil como lo son los juegos creados, las puntuaciones que hayamos hecho y nuestro usuario con todos los datos proporcionados al momento de hacer el registro.

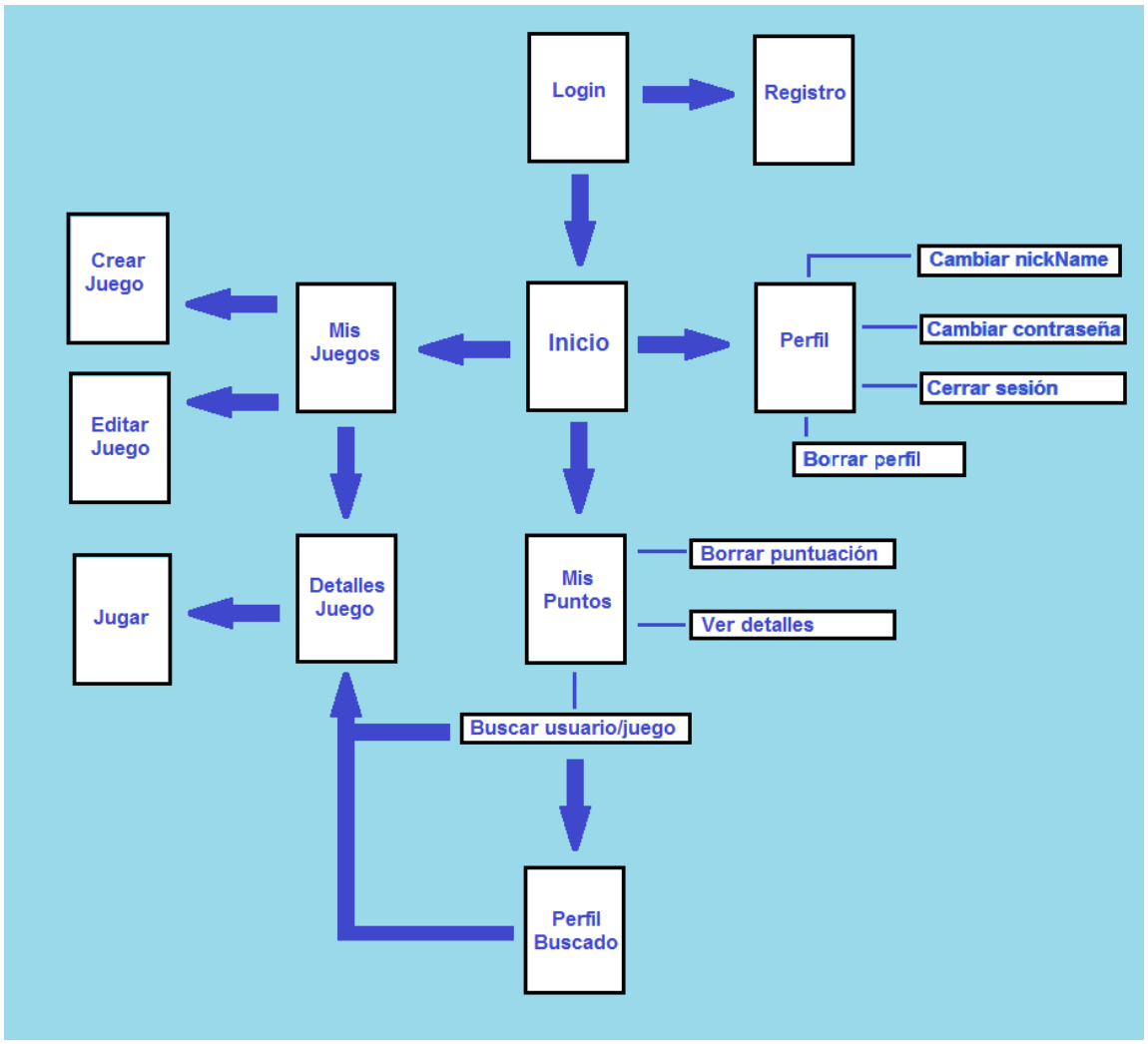


Figura 3.2: Propuesta de diseño de navegación

Capítulo 4

Desarrollo

En este capítulo mostraremos el desarrollo que se llevó a cabo para la construcción del juego de acuerdo a la ruta de navegación mostrada en el capítulo anterior.

4.1 Desarrollo de la pantalla de juego

Para el desarrollo de la app se usó la versión de android studio lo primero que se hizo fue definir una rejilla de trabajo. En dicha rejilla se colocarán objetpos. Así que el siguiente paso fue definir el objeto principal con XML de tiempo FrameLayout. El cual se definio como: imagen, texto e imagen, una sobrepuesta a la anterior, esto para simular que al desaparecer la primera imagen la respuesta se encontraba debajo de la "carta", cuando la pregunta o respuesta se presentaba en forma de texto, la segunda imagen desaparecía y en caso de contar con una imagen, ya fuese como pregunta o respuesta, el campo texto desaparecía permitiendo mostrar la imagen; esto fue realizado únicamente como la parte visual, ya que, se debe contar con un objeto llamado "adaptador" que es un archivo de tipo java encargado de colocar los datos dentro de cada uno de los *ítems* de la grilla que tiene métodos *"get"* y *"set"* estos datos vienen de un objeto creado de nombre "Datos" también de tipo java vease la figura 4.1.

Otro objeto muy importante es el llamado objeto "Datos" el cual cuenta con 6 variables de clase que son: *"id"* variable de tipo entero compartido con otro dato, esto quiere decir que la pregunta tiene un *id* y la repuesta comparte ese mismo *id* permitiéndonos identificar a qué pregunta le corresponde cuál respuesta; "ImagenP" que es un de tipo entero donde se almacena la imagen que se hace referencia de forma numérica, corresponde a la imagen que da el efecto de que las cartas están volteada, en otras palabras el reverso de la carta; "ImagenR" aquí se almacena la imagen (en caso de contar con alguna) que tiene el contenido de la carta al voltearse nuevamente. *"State"* es un dato de tipo booleano que nos permite

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/FrameCarta"
    android:layout_width="135dp"
    android:layout_height="175dp"
    android:layout_margin="20dp"
    android:padding="5dp">

    <ImageView
        android:id="@+id/imgR"
        android:layout_width="125dp"
        android:layout_height="164dp"
        android:scaleType="fitStart" />

    <TextView
        android:id="@+id/textR"
        android:layout_width="113dp"
        android:layout_height="152dp"
        android:gravity="center"
        android:text="TextView"
        android:textColor="@color/colorPrimaryDark" />


    <ImageView
        android:id="@+id/imgP"
        android:layout_width="125dp"
        android:layout_height="164dp"
        android:scaleType="fitStart" />

</FrameLayout>
```

Figura 4.1: Apariencia del frame.

saber cuándo una carta o *ítem* fue encontrada con su pareja y ya no participe dentro del juego, marcando como *false* en qué momento ya no debe participar en el juego y *true* en caso de que siga activa; después tenemos el dato *"Check"* también de tipo booleano este nos permite saber cuándo una carta se encuentra boca abajo o boca arriba, marcando *false* para la primera opción y *true* para la segunda, el código se muestra en la figura 4.2.

De esta manera el objeto "Datos" en forma de arreglo pasaba la información al objeto "Adaptador" (figura 4.3) y este, a su vez, se encargaba de inyectarla dentro de los *ítems* y el conjunto de *ítems* constituían a la "rejilla". Una vez personalizada la rejilla a nuestras necesidades de trabajo se comenzó llenando con datos de manera genérica para comprobar que funcionaba de forma correcta programando las animaciones que hacían que la imagen que representaba la carta volteada desapareciera al momento de tocarla y que nos permitiera ver lo que estaba debajo (ya fuese la otra imagen o el texto). Una vez logrado se procedió a programar lo que es el corazón de todo este proyecto, la funcionalidad y la forma de juego e interacción entre el memorama y el jugador, al momento de seleccionar algun *ítem* el primer paso era verificar que esa carta aún se encontrara en juego, ya que, a pesar de desaparecer visualmente las imágenes y textos sus *ítems* aun podían ser seleccionados. Si el *state* de ese *ítem* era false no ejecutaba ninguna acción pero si era *true* (que siguiera dentro del juego) se procedía a la siguiente parte que era acceder al frame correspondiente y desaparecer la imagen de la parte trasera de la imagen, se cambiaba el *check* a *true* (indicando que la carta se encontraba boca arriba). Pasado ese filtro se verifica que ambos tuvieran a misma *id*, si era así entonces se había encontrado la pregunta y su respectiva respuesta haciendo que el paso siguiente fuera con animación desaparecer toda la información (imágenes y texto) y cambiando el state de ambos a *false*, el caso contrario provocaba que la imagen desaparecida

```
public class Datos {
    private Integer Id;
    private Integer ImagenP;
    private Integer ImagenR;
    private boolean Check;
    private boolean State;
    private String TextR;
    private Integer puntos;

    public Datos(){}

    public Datos(Integer id, Integer imagenP, Integer imagenR, String textR, boolean check, boolean state, Integer pts){
        Id = id;
        ImagenP = imagenP;
        ImagenR = imagenR;
        Check = check;
        State = state;
        TextR = textR;
        puntos = pts;
    }

    public Integer getId(){ return  Id; }
    public void setId(Integer id) { Id = id; }
    public Integer getImgP(){ return ImagenP; }
    public void setImagenP(Integer imagenP){ ImagenP = imagenP; }
    public Integer getImgR(){ return ImagenR; }
    public void setImagenR(Integer imagenR){ ImagenR = ImagenR; }
    public boolean getCheck(){ return Check; }
    public void setCheck(boolean check){ Check = check; }
    public boolean getState(){ return State; }
    public void setState(boolean state){ State = state; }
    public String getTextR(){ return TextR; }
    public void setTextR(String textR) { TextR = textR; }
    public Integer getPuntos() { return puntos; }
    public void setPuntos(Integer pts) { puntos = pts; }
}
```

Figura 4.2: Objeto datos.

volviera a ser visible indicando que no eran pareja y dando el efecto de que se volvían a poner boca abajo como se muestra en la figura 4.4.

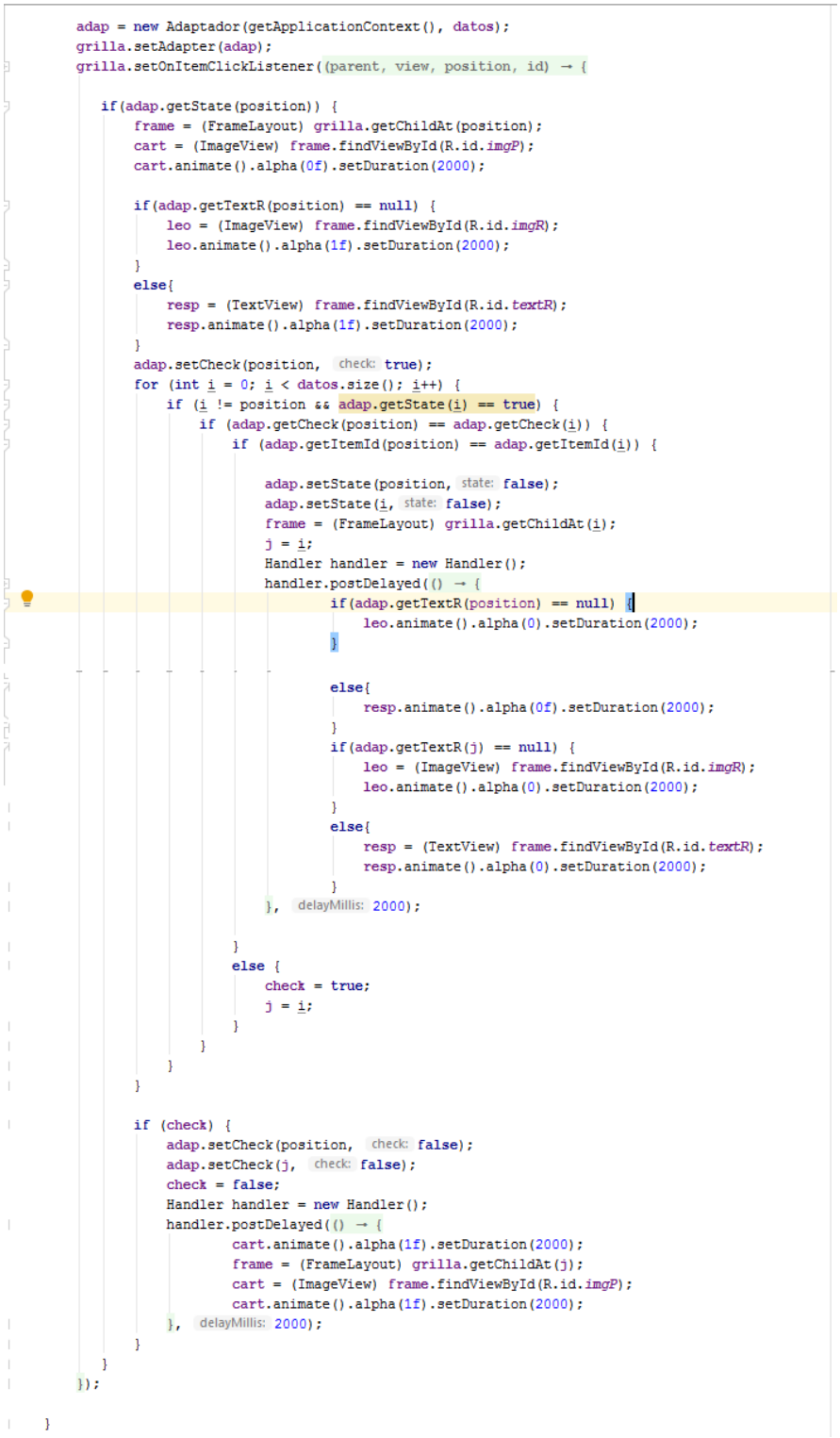
4.1.1 Desarrollo de la pantalla de captura

Una vez que se alcanzó el primer punto, el siguiente objetivo fue implementar una vista donde los datos del juego (preguntas, respuestas y nombre) fueran introducidos por el usuario, el primer paso fue crear una base de datos donde se guardaría la información, las tablas inicialmente fueron: “juego” conformado por los campos “id_game”, “fecha”, “nombre” y “numVeces”, la tabla “pregunta” con los campos “id_ques”, “preg”, “imagenR” y “textR” y por último la tabla “relacion” que solo contenía las tablas “d_j” y “id_p” como se puede ver en las figuras 4.5, 4.6 y 4.7.

Luego se procedió a crear una vista (figura 4.8) que contaba con dos campos de texto, uno para introducir la pregunta y otro para la respuesta, y tres 3 botones: el primero sirve para agregar otra pregunta; el segundo viene con el texto *`delete”*, cuya función era borrar la información de la base de datos a través del método “borrar”; y el último con el título “finalizar”, acción que daba por terminado el proceso de llenado de información. También está incluido un campo de texto donde se puede observar la información que se estaba agregando, esto para saber que se estaba procesando y/o relacionando de manera correcta.

```
public class Adaptador extends BaseAdapter {

    private Context contexto;
    private List<Datos> lista;

    public Adaptador(Context contexto, List<Datos> lista){
        this.contexto = contexto;
        this.lista = lista;
    }

    public int getCount() { return lista.size(); }
    public Object getItem(int position) { return lista.get(position); }
    public long getItemId(int position) { return  lista.get(position).getId(); }
    public Integer getImgR(int position) { return lista.get(position).getImgR(); }
    public Integer getImgP(int position) { return lista.get(position).getImgP(); }
    public boolean getCheck(int position) {return lista.get(position).getCheck();}
    public void setCheck (int position, boolean check) { lista.get(position).setCheck(check); }
    public boolean getState(int position){ return  lista.get(position).getState(); }
    public void setState(int position, boolean state) { lista.get(position).setState(state); }
    public String getTextR(int position) { return lista.get(position).getTextR(); }
    public void setTextR(int position, String textR) { lista.get(position).setTextR(textR); }
    public int getPuntos(int position) { return lista.get(position).getPuntos(); }
    public void setPuntos(int position, int pts) { lista.get(position).setPuntos(pts); }

    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflate = LayoutInflater.from(contexto);
        convertView = inflate.inflate(R.layout.griditem, root: null);

        ImageView imagenP = (ImageView) convertView.findViewById(R.id.imgP);
        imagenP.setImageResource(lista.get(position).getImgP());

        ImageView imagenR = (ImageView) convertView.findViewById(R.id.imgR);
        imagenR.setImageResource(lista.get(position).getImgR());

        TextView textR = (TextView) convertView.findViewById(R.id.textR);
        textR.setText(lista.get(position).getTextR());
        return convertView;
    }
}
```

Figura 4.3: Codigo del adaptador.



```
        adap = new Adaptador(getApplicationContext(), datos);
        grilla.setAdapter(adap);
        grilla.setOnItemClickListener((parent, view, position, id) -> {

            if(adap.getState(position)) {
                frame = (FrameLayout) grilla.getChildAt(position);
                cart = (ImageView) frame.findViewById(R.id.imgP);
                cart.animate().alpha(0f).setDuration(2000);

                if(adap.getTextR(position) == null) {
                    leo = (ImageView) frame.findViewById(R.id.imgR);
                    leo.animate().alpha(1f).setDuration(2000);
                }
                else{
                    resp = (TextView) frame.findViewById(R.id.textR);
                    resp.animate().alpha(1f).setDuration(2000);
                }
                adap.setCheck(position,  check: true);
                for (int i = 0; i < datos.size(); i++) {
                    if (i != position && adap.getState(i) == true) {
                        if (adap.getCheck(position) == adap.getCheck(i)) {
                            if (adap.getItemId(position) == adap.getItemId(i)) {

                                adap.setState(position, state: false);
                                adap.setState(i, state: false);
                                frame = (FrameLayout) grilla.getChildAt(i);
                                j = i;
                                Handler handler = new Handler();
                                handler.postDelayed(() -> {
                                        if(adap.getTextR(position) == null) {
                                            leo.animate().alpha(0).setDuration(2000);
                                        }

                                        else{
                                            resp.animate().alpha(0f).setDuration(2000);
                                        }
                                        if(adap.getTextR(j) == null) {
                                            leo = (ImageView) frame.findViewById(R.id.imgR);
                                            leo.animate().alpha(0).setDuration(2000);
                                        }
                                        else{
                                            resp = (TextView) frame.findViewById(R.id.textR);
                                            resp.animate().alpha(0).setDuration(2000);
                                        }
                                    }, delayMillis: 2000);

                                }
                                else {
                                    check = true;
                                    j = i;
                                }
                            }
                        }
                    }

                    if (check) {
                        adap.setCheck(position,  check: false);
                        adap.setCheck(j,  check: false);
                        check = false;
                        Handler handler = new Handler();
                        handler.postDelayed(() -> {
                                cart.animate().alpha(1f).setDuration(2000);
                                frame = (FrameLayout) grilla.getChildAt(j);
                                cart = (ImageView) frame.findViewById(R.id.imgP);
                                cart.animate().alpha(1f).setDuration(2000);
                            }, delayMillis: 2000);
                        }
                    }
                });

            }
```

Figura 4.4: Parte del código donde se ejecuta el juego.

```
public class DataBase extends SQLiteOpenHelper {

    Context ctx;
    public DataBase(Context context){
        super(context,  name: "data_base",  factory: null, version: 1);
        ctx = context;
    }

    public void onCreate(SQLiteDatabase db){
        db.execSQL("CREATE TABLE juego(id_game INTEGER PRIMARY KEY AUTOINCREMENT, fecha TEXT , nombre TEXT , numVeces TEXT )");
        db.execSQL("CREATE TABLE pregunta(id_ques INTEGER PRIMARY KEY AUTOINCREMENT,preg TEXT , imagenR INTEGER, textR TEXT )");
        db.execSQL("CREATE TABLE relacion(id_j INTEGER , id_p INTEGER )");
    }


    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
        db.execSQL("DROP TABLE IF EXIT juego");
        db.execSQL("DROP TABLE IF EXIT pregunta");
        db.execSQL("DROP TABLE IF EXIT relacion");
        onCreate(db);
    }

    public void deleteAll(){
        db.delete( table: "juego", whereClause: null, whereArgs: null);
        db.delete( table: "pregunta",  whereClause: null, whereArgs: null);
        db.delete( table: "relacion", whereClause: null, whereArgs: null);
    }

    public void deleteDataBase() { ctx.deleteDatabase( name: "data_base"); }
```

Figura 4.5: Contrucción de la base de datos.

```
    public long registrarJuego(String fecha, String nombre){
        ContentValues valores = new ContentValues();
        valores.put("fecha", fecha);
        valores.put("nombre", nombre);
        valores.put("numVeces", 0);
        return db.insert( table: "juego", nullColumnHack: null,valores);
    }

    public long registrarPregunta(String preg,Integer imagenR, String textR){
        ContentValues valores = new ContentValues();
        valores.put("preg",preg);
        valores.put("imagenR",imagenR);
        valores.put("textR",textR);
        return db.insert( table: "pregunta", nullColumnHack: null,valores);
    }

    public long conection(Integer id_j, Integer id_p){
        ContentValues valores = new ContentValues();
        valores.put("id_j",id_j);
        valores.put("id_p",id_p);
        return db.insert( table: "relacion", nullColumnHack: null,valores);
    }

    public Integer idP(String prg){
        Integer id = 0;
        String columnas [] = new String[]{};
        Cursor c = db.query( table: "pregunta",columnas, selection: null, selectionArgs: null, groupBy: null, having: null, orderBy: null);

        for(c.moveToFirst();!c.isAfterLast();c.moveToNext()){
            if(prg.equals(c.getString(c.getColumnIndex( columnName: "preg")))){
                id = c.getInt(c.getColumnIndex( columnName: "id_ques"));
            }
        }

        return id;

    }
```

Figura 4.6: Primera parte de los métodos de la base de datos.

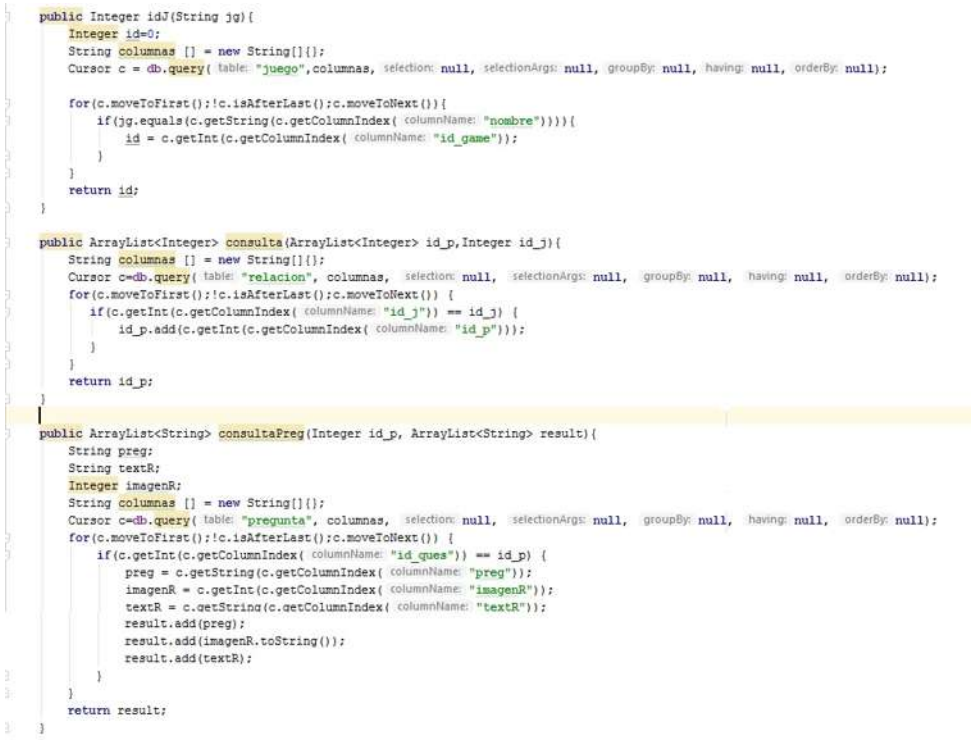


Figura 4.7: Segunda parte de los métodos de la base de datos.

Cada una de las acciones de los botones y otros métodos más necesitados se encuentran programados en su respectiva clase java; el primer método que encontramos es el “mas” utilizado por el botón que agrega nuevas preguntas en la vista, éste se encarga de llamar a la base de datos, obtener los datos que se encuentran dentro de los campos de texto de la vista, verificar que estén llenos (en caso de que alguno esté vacío lanza una alerta indicando el error) para agregarlos a la base de datos y por último vaciar los campos de texto de la vista. Su id correspondiente en la base de datos se guarda dentro de un arreglo llamado “ips” el número se obtiene con el método “idP()” implementado en la base de datos.

Posteriormente, tenemos el método “finalizar()”, utilizado por el botón con el mismo nombre, su función es llamar a una nueva vista donde se introduce el nombre y la fecha, datos que son atrapados por la función “Resultado()” que se encarga de cargar esta información en la base de datos con el método “registrarJuego()”; obtener el *id* dentro de la tabla y llamar al método “armado()” para después pasar el *id* del juego a la primera vista mencionada donde podremos interactuar con la información registrada, vease la figura 4.9. La vista se puede pareciar en la figura 4.10.

El método “armado()” (figura 4.11) se encarga de poner dentro de la tabla “conection” el *id* del juego con el *id* de la pregunta creando la relación. Por su parte el método borrar se

Figura 4.8: Vista para agregar preguntas y guardar un juego.

```
public void mas(View view){
    DataBase db = new DataBase( context: this);
    db.abrir();
    pregunta = (TextView) findViewById(R.id.pregunta);
    String nvP = pregunta.getText().toString();
    respuesta = (TextView) findViewById(R.id.respuesta);
    String nvR = respuesta.getText().toString();

    if(respuesta != null){
       try{
        long result= db.registrarPregunta(nvP, imagenR: 100000,nvR);
        ips.add(db.idP(nvP));
        db.cerrar();
           if(result>0){
               Toast.makeText(getApplicationContext(), text: "Pregunta guardada con exito", Toast.I
           }
      } catch (Exception e){
           Toast t = Toast.makeText( context: this,e.toString(), Toast.LENGTH_LONG);
           t.show();
      }
        lista.setText("");
    }

    pregunta.setText(null);
    respuesta.setText(null);
}

public void finalizar(View view) { new guardadoJuego(contexto,  actividad: Main2Activity.this); }

@Override
public void Resultado(String name, String fecha) {
    DataBase db = new DataBase( context: this);
    db.abrir();
    try{
    long result =db.registrarJuego(fecha,name);
        ipj = db.idJ(name);
    db.cerrar();
       if(result>0){
        Toast t=Toast.makeText( context: this, text: "Juego Guardado con exito",Toast.LENGTH_LONG);
        t.show();
           armado();
      }
    } catch (Exception e){
       Toast t = Toast.makeText( context: this,e.toString(), Toast.LENGTH_LONG);
       t.show();
    }

    Intent intent = new Intent( packageContext: Main2Activity.this,MainActivity.class);
    intent.putExtra( name: "id_j",ipj);
    startActivity(intent);
}

public void armado(){
    DataBase db = new DataBase( context: this);
    db.abrir();
    for(int i=0;i<ips.size();i++){
        db.conection(ipj,ips.get(i));
    }
    db.cerrar();
}
```

Figura 4.9: Código para agregar preguntas y guardar un juego.

Figura 4.10: Vista para agregar preguntas y guardar un juego.

encarga de eliminar la base de datos y toda su información (por si se llegara a requerir).

```
public void armado(){
    DataBase db = new DataBase( context: this);
    db.abrir();
    for(int i=0;i<ips.size();i++){
        db.conection(ipj,ips.get(i));
    }
    db.cerrar();
}
```

Figura 4.11: Código del método “armado()“.

La forma de interactuar correspondía a que el usuario escribiera una pregunta y su respectiva respuesta y con el botón de “más” agregar una nueva pregunta, cuando toda la información estuviese completa se presionaría el botón de “finalizar”, que nos lanza una nueva vista donde se coloca el nombre y la fecha, al presionar “guardar” el código realiza el ensamble entre preguntas y juegos para, finalmente, llamar a la vista al cuestionario creado.

4.2 Base de datos Firebase

Para agregar Firebase a nuestra aplicación hay que seguir una serie de pasos [Goo20]. El primero es crear un proyecto de Firebase para después registrar nuestra app dentro del proyecto, esto se logra colando la id de la aplicación que generalmente se encuentra dentro del archivo “bulid.gradle” en la carpeta *“app”* como se muestra en la figura 4.12. Después,se agrega el archivo de configuración de Firebase a la aplicación, para esto debemos descargar un archivo llamado “google-services.json” y se adjunta al módulo para, posteriormente añadir dentro del “build.gradle” del proyecto en la parte de “dependencies” la línea “classpath com.google.gms:google-services:4.2.0” vease la figura 4.13 y en el “build.gradle” de la aplicación (existen dos uno que pertenece a la aplicación y otro al proyecto) agregamos al final del archivo la línea “apply plugin: com.google.gms.google-services” y el parte de “dependencias” se coloca “implementation com.google.firebase:firebase-core:17.0.0” el propósito es incluir un complemento de Google Services para habilitar los productos de Firebase en la aplicación como aparece en la figura 4.14. Por último, se sincroniza la aplicación, lo que generará que se vincule con el proyecto de Firebase, y así se podrá empezar a usar la base de datos.

Firebase es una base de datos local muy distinta y única de trabajar por dos razones. Primordialmente es porque hace referencia a una tabla dentro de la base de datos, si existen se agregan los

```
buildscript {

  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
  }

  dependencies {
    // ...

    // Add the following line:
    classpath 'com.google.gms:google-services:4.2.0'  // Google Services plugin
  }
}

allprojects {
  // ...

  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
    // ...
  }
}
```

Figura 4.12: Primera parte para agregar Firebase a un proyecto Adroid.

```
apply plugin: 'com.android.application'

android {
  // ...
}

// Add the following line to the bottom of the file:
apply plugin: 'com.google.gms.google-services'  // Google Play services Gradle plugin
```

Figura 4.13: Segunda parte para agregar Firebase a un proyecto Adroid.

```
dependencies {
 // ...
 implementation 'com.google.firebase:firebase-core:17.0.0'

 // Getting a "Could not find" error? Make sure that you've added
 // Google's Maven repository to your root-level build.gradle file
}
```

Figura 4.14: Tercera parte para agregar Firebase a un proyecto Adroid.

datos y en caso de no existir se crea de manera automática, eso quiere decir que las tablas se van creando conforme la aplicación lo va necesitando y no dentro del proyecto que existe en Firebase. La segunda es que para poder agregar datos estos tienen que venir en forma de objetos, Firebase no recibe de parámetros los datos que deseamos agregar si no que recibe un único parámetro que es un objeto de tipo java que contiene los datos que se agregaran a la tabla.

Lo que lleva a crear 4 objetos que hasta ese momento era los requeridos. En primer lugar fue "Persona" (figura 4.15) con las variables "nombre", "apellido", "correo", *"password"* y *"login"*, el segundo fue "Pregunta" (figura 4.16) con las variables "pregunta", "imagen" y "respuesta", el tercer objeto fue "Relacion" que tiene por variables "id_j" y "id_p", y en último lugar "Juego" (figura 4.17) donde las variables son "fecha", "nombre", "numVeces", "pin" y *"nickname"*. Prácticamente el proceso fue convertir lo puesto la base de datos local a un objeto agregando "Persona" para poder crear una vista de entrada y registro, los objetos inician con mayúscula para indicar que son los que se mandan a la base de datos.

```
public class Persona {

    private String nombre;
    private String apellido;
    private String correo;
    private String password;
    private String login;

    public Persona(){

    }

    public Persona(String  nm, String ap, String cor, String pas, String log){
        nombre = nm;
        apellido = ap;
        correo = cor;
        password = pas;
        login = log;
    }

    public String getNombre() { return nombre; }
    public void setNombre(String nm) { nombre = nm; }
    public String getApellido() { return apellido; }
    public void setApellido(String ap) { apellido = ap; }
    public String getCorreo() { return correo; }
    public void setCorreo(String cor) { correo = cor; }
    public String getPassword() { return password; }
    public void setPassword(String pas) { password = pas; }
    public String getLogin() { return login; }
    public void setLogin(String log) { login = log; }
}
```

Figura 4.15: Objeto "Persona".

Con una nueva versión por delante se comenzó por adaptar las clases y sus métodos a la nueva herramienta además de darles un mejor diseño a las vistas, los primeros cambios se

```java
public class Pregunta {
    private String pregunta;
    private Integer imagen;
    private String respuesta;


    public Pregunta(String pg, int img, String rps){
        pregunta = pg;
        imagen = img;
        respuesta = rps;
    }

    public Pregunta(){

    }

    public String getPregunta() { return pregunta; }
    public void setPregunta(String pg) { pregunta = pg; }
    public Integer getImagen() { return imagen; }
    public void setImagen(Integer img) { imagen = img; }
    public String getRespuesta() { return respuesta; }
    public void setRespuesta(String rps) { respuesta = rps; }
}
```

Figura 4.16: Objeto “Pregunta“.

```java
public class Relacion {

    private int id_j;
    private int id_p;

    public Relacion(){

    }

    public Relacion(int nvidj, int nvip){
        id_j = nvidj;
        id_p = nvip;
    }
    public int getId_j() { return id_j; }
    public void setId_j(int nvidj) { id_j = nvidj; }
    public int getId_p() { return id_p; }
    public void setId_p(int nvir) { id_p = nvir; }
}
```

Figura 4.17: Objeto “Relacion“.

dieron en la vista donde se agregaron las preguntas y respuestas; los botones se sustituyeron con un menú en la parte superior de la pantalla; en la carpeta “res” de la aplicación se creó una nueva carpeta llamada “menu” como se ve en la figura 4.18, aquí se incluirían todos archivos correspondientes a las opciones de cada vista, los archivos ahí son de tipo “Menu resource file” con terminación “xml” (todo lo relacionado con la parte gráfica del proyecto tiene esta terminación). El menú de la vista contiene dos ítems uno con un símbolo de más y otro con el símbolo de guardado, el primero mencionado funcionaba para agregar una nueva pregunta y el segundo para guardar los datos dentro de la base de datos, una vista simple pero más limpia, vease la figura 4.19.

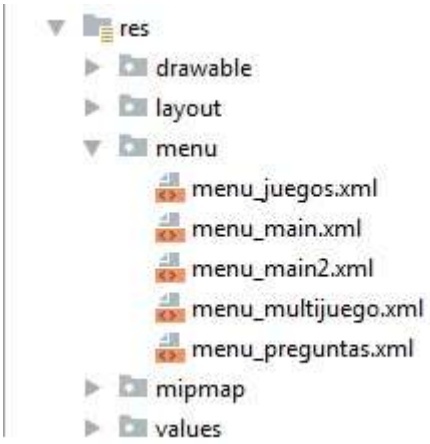


Figura 4.18: Carpeta para crear menus.

Los métodos del código seguían trabajando de la misma manera solo que adaptados a Firebase, al momento de querer agregar una nueva información se metía dentro de un objeto y el

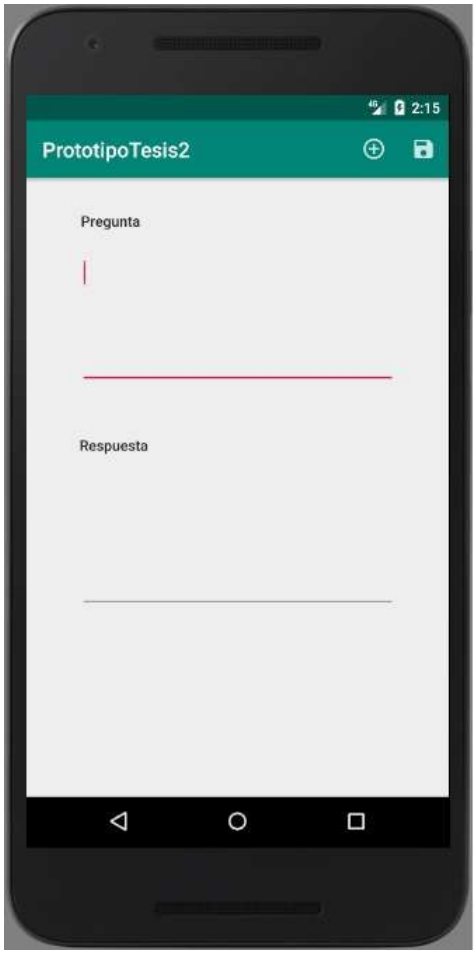


Figura 4.19: Vista con el meú incluido.

objeto se pasaba como parámetro en la referencia creada (figura 4.20), en esta ocasión se implementó una condición con la que los campos de pregunta y respuesta no podían quedar vacías y no se podía agregar una nueva pregunta hasta no llenar esos datos.

Otro cambio se dio en la clase que se ejecuta al momento de jugar con la id del juego que se recibía de la clase anterior se busca dentro de la tabla “Relación” con el método “listaPreguntas()” (figura 4.21), el id de la pregunta que correspondía a ese juego para pasarlo como parámetro al método “agregaPreguntas()” (figura 4.22), el cual tiene como función buscar en la tabla “Preguntas” la correspondiente información que se guarda dentro del arreglo “datos”.

```
public void Resultado(String name, String fecha){

    String nvP = pregunta.getText().toString();
    String nvR = respuesta.getText().toString();
    if(!nvP.equals("")&& !nvR.equals("")){
        Pregunta p = new Pregunta(nvP, img: 100000,nvR);
        databaseReference = FirebaseDatabase.getInstance().getReference().child("Preguntas");
        databaseReference.child(String.valueOf(maxIdP+1)).setValue(p);
        ips.add((int)maxIdP+1);
        Toast.makeText(getApplicationContext(), text: "Pregunta guardada con exito", Toast.LENGTH_SHORT).show();
    }
    databaseReference = FirebaseDatabase.getInstance().getReference().child("Juegos");
    Juego p = new Juego(fecha,name, nv: 0, pn: "7852-21",nickName);
    databaseReference.child(String.valueOf(maxIdJ+1)).setValue(p);
    armado();
    ips.clear();
    Intent intent = new Intent( packageContext: Main2Activity.this,Main4Activity.class);
    intent.putExtra( name: "nickName",nickName);
    startActivity(intent);

}

private void validacion(){
    String nvP = pregunta.getText().toString();
    String nvR = respuesta.getText().toString();
    if(nvP.equals("")){
        pregunta.setError("Required");
    }
    if(nvR.equals("")){
        respuesta.setError("Required");
    }
}
```

Figura 4.20: Método para agregar preguntas/respuestas con validación de campos.

Una vez que se terminó el proceso de adaptación se crearon nuevas vistas y funcionalidades para comenzar a acercarse a la versión final de la aplicación, implementando primero una vista de inicio que da la bienvenida al usuario y pide, para dar acceso, el *nickname* con el que se registró y su *password* como se puede ver en la figura 4.23. Si el usuario no cuenta con uno, dentro existe un botón que despliega una vista pop en la cual se piden los datos necesarios para darse de alta, vease la figura 4.24.

Su respectiva clase contaba con 3 métodos. Primera: “entrar()” figura 4.25, esta verifica que los datos proporcionados en el campo de *nickname* y *password* de la vista se encuentren en la base de datos y sean correctos, en caso de no ser así manda un mensaje informando que alguno de los dos es erróneo. Luego, está el método “registrar” el cual es llamado cuando el usuario presiona el botón de “registrar” que inicializa la vista “registrarUsuario”. Esta es la

```
public void listaPreguntas(){
    databaseReference.child("Relacion").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

            for(DataSnapshot objSnaptshot: dataSnapshot.getChildren()){
                Relacion p = objSnaptshot.getValue(Relacion.class);
                if(p.getId_j() == id_j){

                    agregaPreguntas(p.getId_p());
                }
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}
```

Figura 4.21: Método listaPreguntas.

```
public void agregaPreguntas(final int id_p){
        databaseReference.child("Preguntas").addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                datos.clear();
                for (DataSnapshot objSnaptshot : dataSnapshot.getChildren()) {
                    if ( Integer.parseInt(objSnaptshot.getKey()) == id_p) {
                        Pregunta p = objSnaptshot.getValue(Pregunta.class);
                        datos.add(new Datos(id_p,R.drawable.carta, imagenR: 10000,p.getPregunta(), check: false, state: true));
                        datos.add(new Datos(id_p,R.drawable.carta,p.getImagen(),p.getRespuesta(), check: false, state: true));
                    }
                }

                adap = new Adaptador(getApplicationContext(), datos);
                grilla.setAdapter(adap);
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });
    }
}
```

Figura 4.22: Método agregarPreguntas.

Figura 4.23: Vista de inicio.

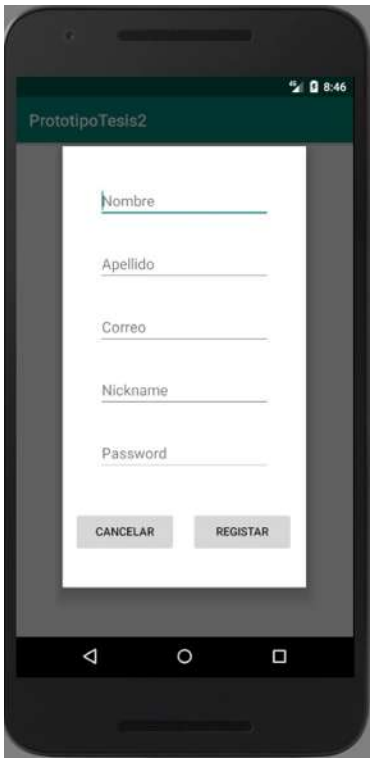


Figura 4.24: Vista para registrarse.

que contiene los campos donde el usuario los coloca (no pueden quedar vacíos). Por último, el método “Resultado()” cuando el usuario ha colocado todos sus datos y presiona el botón de aceptar se ejecuta haciendo referencia a la base de datos (previamente inicializada) en la tabla “Personas” y pasando a modo de objeto la información proporcionada.

Ya que los datos eran verificados y correctos se muestra una vista que contiene un listado con los juegos que el usuario ha creado como se ve en la figura 4.26.

En la parte superior se encuentran 3 iconos, el primero es un símbolo de “más” cuya función es crear un nuevo juego, el segundo son las teclas de un juego que sirve para buscar un juego creado por otro usuario por medio del *pin*, y, el último, un icono de perfil sirviendo para buscar el listado de juegos creados por otro usuario a través de su *nickname*.

Al presionar cualquiera de los elementos de la lista nos redirecciona a otra vista donde se muestra el título del juego, seguido de una lista con las puntuaciones y nombres de jugador de todos los usuarios que jugaron, y en la parte final un botón para iniciar el juego, esta misma vista era a la que se presentaba cuando un jugador buscaba un juego (figura 4.27).

Los métodos de las últimas dos vistas mencionadas no tienen tanta complejidad, ya que, solo son llamadas a la base de datos para obtener los datos que van dentro de las listas y los métodos de los botones que solo mandan información e inician la vista siguiente.

```
public void entrar(View view){
    db = firebaseDatabase.getReference();
    rgs = false;
    db.child("Personas").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            for(DataSnapshot objSnaptshot: dataSnapshot.getChildren()){
                Persona p = objSnaptshot.getValue(Persona.class);
                if((usuario.getText().toString().equals(p.getCorreo()) || usuario.getText().toString().equals(p.getLogin())) && contraseña.getText().toString().equals(p.getPassword())){
                    rgs = true;
                    nickName = p.getLogin();
                }
            }

            if(rgs){
                Intent intent = new Intent( packageContext: Main3Activity.this,Main4Activity.class);
                 intent.putExtra( name: "nickName",nickName);
                 startActivity(intent);
            }
            else{
                Toast.makeText(getApplicationContext(), text: "El usuario o la contraseña son invalidos", Toast.LENGTH_SHORT).show();
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}

public void registrar(View view) { new registarUsuario(contexto, actividad: Main3Activity.this); }


public void Resultado(String nombre, String apellido, String correo, String password, String login){
     p = new Persona(nombre,apellido,correo,password,login);
    db = FirebaseDatabase.getInstance().getReference().child("Personas");
    db.child(String.valueOf(maxIdP+1)).setValue(p);
}
```

Figura 4.25: Método “Entrar”.

4.3 Diseño final de la aplicación

Para este punto la aplicación se encontraba en el inicio de lo que sería el resultado final, las funciones básicas estaban implementadas y el siguiente objetivo en mente era hacer un diseño mucho más amigable al usuario así como comenzar a explotar cualquier elemento de la aplicación, se rediseñaron algunas vistas y se agregaron herramientas que complementaron la aplicación, este fue el paso revolucionario y con más cambios dentro del desarrollo, como se puede observar en la figura.

Siguiendo el orden de navegación la primera vista en donde los cambios empezaron a ser notorios fue la pantalla de inicio (figura 4.28); se agregó una imagen de fondo, el tipo de letra, así como botones personalizados. Para cambiar el tipo de letra se debe colocar la tipografía a usar en formato .ttf (el implementado fue futuranormal) dentro de la carpeta assents/Font para que desde la clase se pueda hacer referencia mediante “Typeface tf = Typeface.createFromAsset(getAssets(), font/futuranormal.ttf);” y pasando el resultado como parámetro al texto usando la propiedad “.setTypeaface()”. Para los botones personalizados dentro de la carpeta *“drawable”* se crea un nuevo documento xml donde se agregan las características deseadas como el color, la forma y el tipo de texto que utilizara, hecho esto al momento que creamos un botón debemos agregar “android:background=@drawable/nombre del diseño”, en lo que respecta a su clase esta no sufrió ningún cambio.

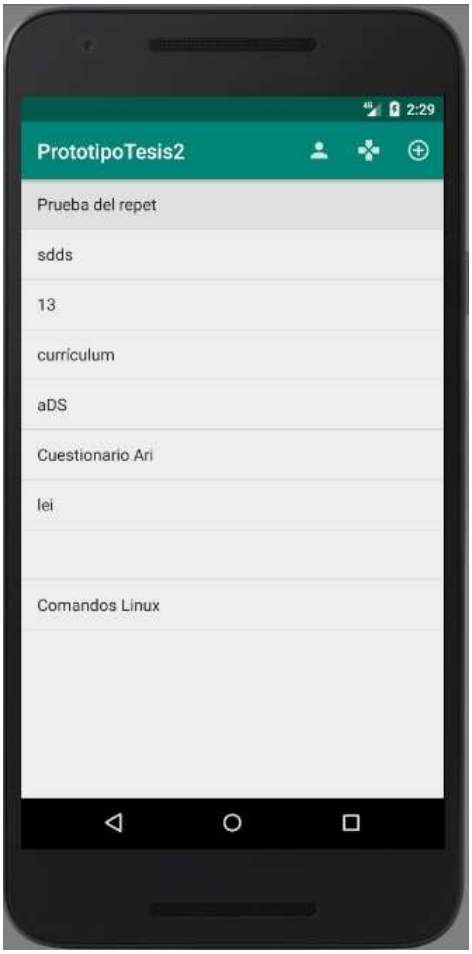


Figura 4.26: Vista una vez que el usuario entra.

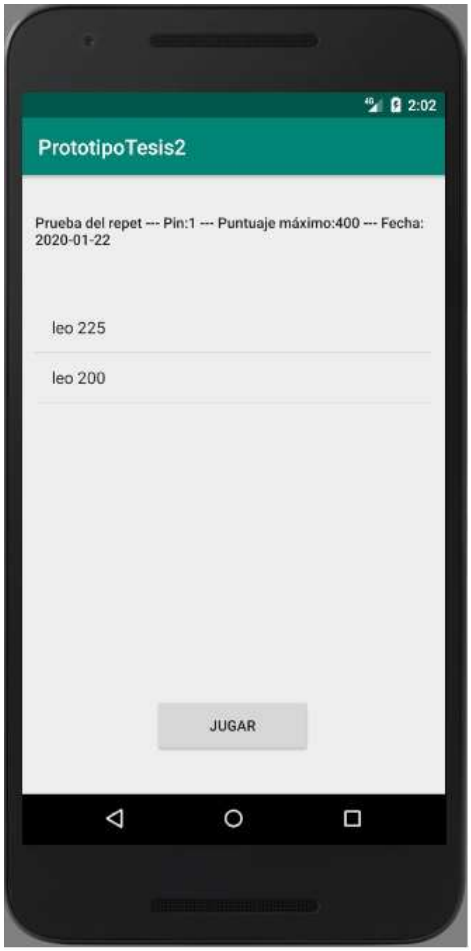


Figura 4.27: Vista cuando seleccionamos un juego.

Figura 4.28: Rediseño de la pantalla de inicio.

Figura 4.29: Rediseño de la pantalla del usuario.

La vista donde se mostraba una el listado de juegos creados ahora llamada “Mis juegos” pasó por un gran cambio de diseño, ya que, actualmente con pestañas deslizables muestra una nueva sección llamada “Mis puntos”, “Mis juegos” ahora presentaba el pin del juego, la máxima puntuación obtenida y la fecha en que fue creado en cada uno de sus ítems y contaba con un botón en la parte inferior derecha un botón con el icono de “más” que servía crear un nuevo juego. La sección de “Mis puntos” expone en forma de listado todos los juegos que se han realizado, mostrando en cada ítem el nombre del juego, el pin, la puntuación máxima, la fecha en que se creó, la puntuación del usuario y el tiempo que le había tomado realizarlo; además de contar con un botón con el ícono de una lupa que nos permite buscar ya fuese a un jugador o un juego, los botones sustituyeron al menú que se encontraba en la parte superior de la aplicación, vease la figura 2.29.

Para realizar estos cambios se creó una nueva actividad (cabe destacar que para crear una nueva vista Android Studio lo llama “actividad” que hasta ese momento todas habían sido vacías) de tipo “Navigation Drawer Activity” que es la que permite la navegación ventanas a través del deslizamiento llamadas *“fragment”*, cada *fragment* debe tener su propia vista que es donde se coloca el contenido.

Su respectiva clase es una de las más complejas dentro del proyecto. En esta parte se manipulan los objetos que tienen en común los fragments, pero no las acciones de cada uno; se debe primero pasar los fragments que se encuentran previamente definidos en un adaptador y haciendo referencia a él se inyecta dentro de la clase, dependiendo del fragment, la vista que se le va dar al botón, lo que quiere decir que el botón en ambos es el mismo solo que al deslizarse cambia de apariencia y funcionalidad. Cuando el botón es presionado llama a su respectivo método “setOnClickListener()” (los botones tienen dos formas de operar, la primera es llamando a un método de la clase o llamando a un método predeterminado todo depende de en qué nivel se requiera su funcionalidad) que la verificar en que pestaña nos encontramos define que acción se va a ejecutar. Si el apartado es “Mis juegos” manda a la vista donde se creó un nuevo juego; si es “Mis puntos” muestra una pequeña ventana que pide ingresar ya sea el *nickname* de un usuario o el pin de un juego para redireccionar a su respectiva vista. El botón implementado fue el pionero en una herramienta implementada en todo el proyecto en sus versiones siguientes, al dejar presionado el botón manda una alerta donde presenta cuál es su función.

Para llenar los datos de “Mis puntos” se creó una nueva tabla en la base de datos llamada “Jugar” con los campos *“nickname”*, “id_J”, “tiempo”, “fecha”, *“score”* y “nombre”, al momento de que el usuario termina un juego se recogen datos como el nombre del usuario, el identificador del juego, el tiempo que le tomó resolverlo, la puntuación que obtuvo, la fecha en que lo hizo y el nombre del juego. Entonces dentro de la clase se hace referencia a ella y se obtienen los datos que son inyectados dentro de la lista, para obtenerlo se busca

dentro de la tabla aquellos datos cuyo campo *"nickname"* coincidiera con el nombre del usuario como en la figura 4.30.

```
public void llenarLista(){
    db.child("Jugar").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            juegos.clear();
            for(DataSnapshot objSnapshot: dataSnapshot.getChildren()){
                j = objSnapshot.getValue(Jugar.class);
                id = j.getId_J();
                if(nickName.equals(j.getNickName())){
                    segundos = (int) (j.getTiempo() *  0.001);
                    minutos = segundos /60;
                    segundos = segundos % 60;
                    id_J.add(objSnapshot.getKey());
                    juegos.add(j.getNombre() + "   Score: " + j.getScore() + "   Tiempo: " + minutos + ":" + segundos);
                }
            }

            adaptador = new ArrayAdapter<String>(view.getContext(),android.R.layout.simple_list_item_1,juegos);
            listaPuntos.setAdapter(adaptador);
            listaPuntos.setOnItemClickListener((parent, view, position, id) → {
                    new detallesJuego(view.getContext(), actividad: puntosFragment.this ,id_J.get(position));
            });
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}
```

Figura 4.30: Método implementado para llenar la lista de puntuaciones.

Al presionar alguna de las puntuaciones se desplegaba una pequeña vista donde se mostraba de manera más ordenada los detalles de la parte seleccionada.

La vista donde se creaban los juegos sufrió ligeros cambios, al momento de guardarlo se implementa un icono que sirve como botón para desplegar un calendario en donde se selecciona la fecha de creación, también se ejecutan un par de condiciones que mandan un mensaje al usuario si algunos de los campos se encuentra vacío.

Otro cambio importante es el de presionar el botón de buscar en la pantalla principal, ya que en el campo de texto aceptaba tanto un pin como nickname, por lo cual se programa una función que primero busca en la tabla "Jugadores" alguna coincidencia, en caso de no encontrarse buscaba en la tabla "Juegos" la coincidencia, dependiendo de lo introducido era la vista a la que seriamos mandados, en caso de que el campo se encontrara vacío indicaba que para completar el proceso no podía debía contener algo escrito a través de una alerta.

Conteo de puntos

El último cambio es dentro de la clase que se ejecuta al momento de jugar, primeramente se asignó un sistema de puntuaciones, para esto el objeto "Dato" se modificó agregándosele una nueva variable de clase llamada "puntos", esta se inicia con un valor de 100 al momento de comenzar el juego, cada *ítem* contaba su respectiva puntuación, si se elige una pareja incorrecta cada ítem pierde el 50% de su valor, en caso contrario se suman las puntuaciones de ambos *ítems* y se le agrega al valor que se obtiene anteriormente como puntuación (empezando en cero). Después se agrega un método que se ejecuta cuando el juego termina llamado "fin", para saber que ha terminado, cada que se encuentra un par, incrementa, el segundo el valor de una variable llamada "jts" y cuando esa variable es igual al número de datos entonces se ha terminado. El trabajo de dicha función es calcular el tiempo que ha tomado resolverlo (se empieza a medir una vez que el usuario hace su primer movimiento) y llama a una nueva vista pasándole los datos del tiempo, la fecha, el nombre del jugador, el *id* del juego, la puntuación y el nombre del juego.

Como se mencionó con anteriomente se desarrolló una nueva vista muy simple con el propósito de hacerle saber al usuario su desempeño durante el juego, empezando por mostrar una pequeña felicitación, su puntuación y su tiempo, con un botón que presenta el título de "Continuar", vease la figura 4.31. Cuando el botón era presionado se hace una referencia a la tabla "Jugar" de la base de datos agregando el nuevo objeto con los datos correspondientes (que fueron los obtenidos de la vista anterior), luego se busca dentro de la tabla "Juegos" y se localiza el juego correspondiente para actualizar la puntuación máxima en caso de que el usuario logre una mayor a la registrada, para que esto sea posible se modificó la tabla "Juegos" agregando un nuevo campo llamado "maxScore" (figura 4.32).

Para este punto la aplicación ya era la base de lo que es la versión final, entonces es de preocupar la experiencia del usuario y agregar más funciones a la aplicación con el objetivo de que se presenta más completa. En cuestión de trasformaciones pasadas a lo largo del desarrollo de la aplicación se encuentran la fase 3. Para comentar el nuevo conjunto de cambios realizados se describirá cada una conforme a la navegación.

4.4 Elementos finales

Comenzando nuevamente por la pantalla de inicio se agregó una nueva opción que consiste en un método para que el usuario pueda recuperar su contraseña si se ha olvidado de ella (figura 4.33), para la vista solo se agregó un texto en la parte inferior que decía "Olvide mi contraseña" al seleccionarlo se abría una pequeña pestaña donde el usuario debe introducir su correo y su nickName para, posteriormente, seleccionar el botón de "Enviar".

Figura 4.31: Vista una vez terminado el juego.

```
        Intent intent = getIntent();
        Bundle extra = intent.getExtras();
        fecha = extra.getString( key: "fecha");
        tiempo = extra.getLong( key: "tiempo");
        nickName = extra.getString( key: "nickName");
        id_j = extra.getInt( key: "id_J");
        total = extra.getInt( key: "score");
        nombre = extra.getString( key: "nombre");
        btn = (Button) findViewById(R.id.buttonZ);

        segundos = (int) (tiempo *  0.001);
        minutos = segundos /60;
        segundos = segundos % 60;

        puntuacion = (TextView) findViewById(R.id.puntuacion);
        puntuacion.setText("Tú puntuación fue de: "+total);

        tmp = (TextView) findViewById(R.id.tiempo);
        tmp.setText("Tu tiempo fue de: "+minutos+":"+segundos);

        inicializarFirebase();
        databaseReference = FirebaseDatabase.getInstance().getReference().child("Jugar");
        databaseReference.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if(dataSnapshot.exists()){
                    maxIdP=(dataSnapshot.getChildrenCount());
                }
                else{
                    maxIdP=0;
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });
    }
    public void agregar(View view){
        Jugar p = new Jugar(nickName,id_j,tiempo,fecha,total,nombre);
        databaseReference = FirebaseDatabase.getInstance().getReference().child("Jugar");
        databaseReference.child(String.valueOf(maxIdP+1)).setValue(p);

        databaseReference = firebaseDatabase.getReference();
        databaseReference.child("Juegos").addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                Juego j;
                for(DataSnapshot objSnapshot: dataSnapshot.getChildren()){
                    if(id_j == Integer.parseInt(objSnapshot.getKey())){
                        j = objSnapshot.getValue(Juego.class);
                    //      Toast.makeText(getApplicationContext(),"Hola Leo", Toast.LENGTH_SHORT).show();
                        if(total>j.getMaxScore()) {
                            j.setMaxScore(total);
                            databaseReference = FirebaseDatabase.getInstance().getReference().child("Juegos");
                            databaseReference.child(objSnapshot.getKey()).setValue(j);
                        }
                    }
                }

            }
            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });

        Intent intent = new Intent( packageContext: Main7Activity.this, Main8Activity.class);
        intent.putExtra( name: "nickName", nickName);
        startActivity(intent);

    }
```

Figura 4.32: Clase de la vista mostrada en la imagen anterior.

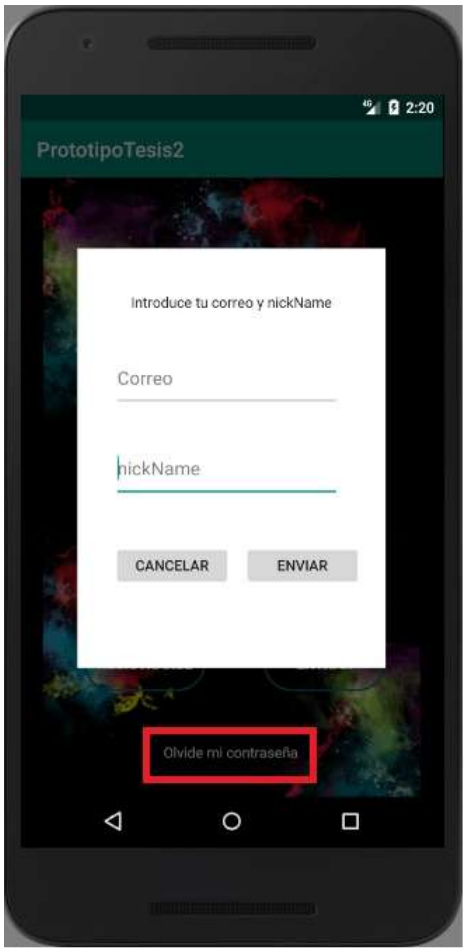


Figura 4.33: Vista para recuperar la contraseña.

En la clase se creó un método llamado “resultadoCorreo()” que se ejecuta con el botón de “Enviar” lo primero que hace es buscar dentro de la tabla “Personas“ un usuario que coincida con los datos proporcionados, en caso de no encontrarlo manda un mensaje diciendo que alguno de los datos es erróneo, caso contario se da permiso a la aplicación de mandar correos a través de un “StricMode”, después se crea una variable de tipo *“Properties”* donde se pasan los parámetros como el host al que nos queremos comunicar, el puerto que se va a utilizar y un boleano como autorización. Luego con una variable de tipo *“Session”* se coloca de parámetros el correo y contraseña que serán utilizados para mandarle la contraseña al usuario (se creó un correo especialmente para esto, en las configuraciones se le debe permitir que la aplicación pueda iniciar sesión) si el valor retornado es *true* entonces se inicia una variable de tipo *“Mesagge”* que recibe como parámetro la confirmación enviada por la *“Session”*, ya inicializado se usan sus métodos *“set”* para llevar los datos del mensaje como el correo de donde se envía la contraseña, el correo del usuario, el asunto y el mensaje, como se muesta en la figura 4.34. Por último se manda el correo y el usuario recibe una altera de que la contraseña se envió a su correo.

```
public void resultadoCorreo(final String correoUser, final String nickNameUser){
    db = firebaseDatabase.getReference();
    db.child("Personas").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            for(DataSnapshot objSnaptshot: dataSnapshot.getChildren()){
                Persona p = objSnaptshot.getValue(Persona.class);
                if(p.getCorreo().equals(correoUser) && p.getLogin().equals(nickNameUser)){
                    password = p.getPassword();
                }
            }

            if (password != "") {
                StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();
                StrictMode.setThreadPolicy(policy);
                Properties props = new Properties();
                props.put("mail.smtp.host", "smtp.googlemail.com");
                props.put("mail.smtp.socketFactory.port", "465");
                props.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
                props.put("mail.smtp.auth", "true");
                props.put("mail.smtp.port", "465");

                try {
                    session = Session.getDefaultInstance(props, getPasswordAuthentication() → {
                                return new PasswordAuthentication( userName: "companyplox@gmail.com",  password: [redacted]; //correo y contraseña del proved
                            });
                            if (session != null) {
                                Message message = new MimeMessage(session);
                                message.setFrom(new InternetAddress("companyplox@gmail.com")); //correo del provedor
                                message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(correoUser)); //Correo del receptor
                                message.setSubject("Recuperación de contraseña"); // Asunto
                                message.setContent( o: "Tú contraseña es: " + password,  s: "text/html; charset=utf-8");
                                Transport.send(message);
                                Toast.makeText(getApplicationContext(),  text: "La contraseña se a enviado al correo: " + correoUser, Toast.LENGTH_SHORT).show();

                            }
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
                else{
                    Toast.makeText(getApplicationContext(),  text: "El correo o el nickName es invalido", Toast.LENGTH_SHORT).show();
                }
            }
            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {
            }
        });
    }
```

Figura 4.34: Método para recuperar la contraseña y mandar el correo.

4.5 Vista de la puntuación

La siguiente vista es donde el usuario visualiza sus puntos y sus juegos creados, se agregó un botón en la parte superior derecha con un ícono, el cual tiene como propósito que el usuario pueda configurar información de su cuenta. También se agregaron nuevas opciones a los elementos de la listas tanto en mis "Mis juegos" como en "Mis puntos". En Android Studio hay diferentes tipos de click, el click largo y el click corto. El largo se refiere a cuando dejamos el dedo sobre algo por mucho tiempo, y el corto se refiere a cuando presionamos y soltamos inmediatamente llamados "setOnLongClickListener()" para el largo y "setOnClickListener()" para el corto.

Para "Mis juegos" al hacer un click corto se muestra un mensaje diciendo que para ver las opciones se requiere dar un click largo, por su parte el click largo (figura 4.35) despliega una serie de opciones "Ver" que al seleccionarlo manda al usuario a la vista donde podemos comenzar el juego. "Editar" esta opción permite al usuario modificar las preguntas y respuestas de algún juego, y "Borrar" cuya función era borrar un juego de los creados.

Para programar todos estos cambios existen varias operaciones complejas, el menú que sale al dar un presionar no se crea dentro de la vista, sino en la carpeta de "menus" se crea uno nuevo que luego es inyectado en el código para crear la relación, a través de la función "onCreateContextMenu()" es que sucede esto, luego la función "onContextItemSelected()" permite mencionar en qué parte queremos que se muestre el menú y que debe hacer cada una de las opciones, las primeras dos nos mandan a una vista, las cuales se retomarán más adelante (figura 4.36).

4.5.1 Opcion "Borrar"

La tercera opción es más interesante al ser un proceso nuevo, se habían agregado elementos pero no borrado hasta el momento, se pueden borrar los datos del juego de la tabla "Juegos" simplemente es decir, se debe implementar un borrado en cascada, ya que eso solo eliminaría los datos como el nombre, la fecha de creación y quien lo creo las preguntas y la interacción se encontraban en otras tablas entonces el proceso era el siguiente, primero se deben buscar todas las preguntas relacionadas con ese juego para borrarlas y borrar la relación, luego se deben borrar el registro de los puntos que se han obtenido al elegir ese juego para finalmente borrar los datos del juego de la base de datos, con eso todo lo relacionado al juego desaparece. Cuando el usuario selecciona esta opción se le pide una confirmación, una vez aceptada se hace referencia a la tabla "Relacion" en la base de datos la cual con el id del juego que el usuario quiere borrar busca el id de las preguntas pertenecientes a ese juego para pasarlo como parámetro a la función "borraPreguntas()" que hace el mismo proceso pero con la tabla

Figura 4.35: Vista de opciones para cada juego.

```java
@Override
public void onCreateContextMenu(@NonNull ContextMenu menu, @NonNull View v, @Nullable ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    int id = v.getId();
    MenuInflater inflater = new MenuInflater(getContext());
    if(id == R.id.listaJuegos){
        inflater.inflate(R.menu.menu_juegos,menu);
    }
}

@Override
public  boolean onContextItemSelected(@NonNull MenuItem item) {
    AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();
    switch (item.getItemId()){
        case R.id.action_view_game:
            Intent intent = new Intent(view.getContext(),Main5Activity.class);
            intent.putExtra( name: "nombre",juegos.get(info.position));
            intent.putExtra( name: "id_J",id_J.get(info.position));
            intent.putExtra( name: "nickName", nickName);
            startActivity(intent);
            return true;
        case R.id.action_edit_game:
            Intent intent2 = new Intent(view.getContext(),editarJuegoActivity.class);
            intent2.putExtra( name: "nombre",juegos.get(info.position));
            intent2.putExtra( name: "id_J",id_J.get(info.position));
            intent2.putExtra( name: "nickName", nickName);
            startActivity(intent2);
           return true;
        case R.id.action_delete_game:
             new confirmDelete(getContext(), actividad: juegosFragment.this,info.position);
            return true;
        default:
            return super.onContextItemSelected(item);
    }

}
```

Figura 4.36: Programación de cada una de las opciones.

“Preguntas” ya localizados los elementos se borran con la función “db.child.().removeValue()”, “db” es la referencia en la base de datos a la tabla, a “child()” se le debe pasar como parámetro el número del elemento que queremos borrar, en Firebase cada elemento tiene un número que corresponde a su posición en la tabla el cual es único, después se llamaba a la función “borrarPuntos()” para buscaba dentro de la tabla “Jugar” los datos donde el id de los juegos (el que se iba a eliminar y el de la base de datos) coincidieran para borrarlo, por último se borraba los datos de la tabla “Juegos” y con eso el proceso de borrado quedaba completado, vease la figura 4.37.

4.5.2 Editar un juego

Para la segunda para la segunda opción que era la de “Editar” se creó una nueva vista donde que recibía como datos el id del juego seleccionado y sus detalles, esta nueva vista contaba en la parte superior derecha con dos íconos el primero para guardar el juego una vez hechas las modificaciones y el segundo un ícono para agregar más preguntas seguido de los detalles del juego y en lo de más una lista con las preguntas y su respectiva respuesta, al momento de dar un click largo sobre alguna pregunta se desplegaban opciones dos opciones “Editar” y “Borrar”. Estas vistas pueden verse en las figuras 4.38 y 4.39.

Para la parte de la clase los detalles del juego se metieron dentro del cuadro de texto, luego con el id del juego se llamaba a una función llamada “listaPreguntas()” que buscaba dentro de la tabla “Relación()” la id de las preguntas que correspondían siendo pasadas como parámetro a la función “agregaPreguntas()” para con ese buscar la información correspondiente que era puesta en la lista de que el usuario veía, además de agregarse por partes a 3 arreglos “preguntas”, “respuestas” y *“ips”*, los primeros dos servían para editar una pregunta y el tercero para borrarla.

Los menús se agregaron como se explicó anteriormente (tanto el superior como el de cada item) para los ítems dependía de que se seleccionaba, si se seleccionaba la opción de “Editar” se llamaba una ventana donde se pasaban los datos de los arreglos en la posición donde fue seleccionado, el primer texto presentaba la pregunta y el segundo la respuesta en modo que se podía editar el contenido dentro de ellas, si el usuario daba en cancelar la ventana se cerraba sin ocasionar un cambio, si le daba guardar se llamaba al método “Resultado” que hacía referencia a la base de datos en la tabla “Preguntas” para agregar en la misma posición el nuevo elemento que sustituía al viejo.

Si en vez de eso seleccionaba “Borrar” se le pedía una confirmación que al aceptarla llamaba a la función “ResultadoConfirm()” que recibía como parámetro la posición de la lista que fue seleccionada y haciendo referencia a la tabla “Preguntas” eliminaba la información que contenida en el id (Esa id estaba guardada en el arreglo *“ips”* y con la posición de retornaba

```
public void ResultadoConfirm(final int position){
    db.child("Relacion").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

            for(DataSnapshot objSnaptshot: dataSnapshot.getChildren()){
                Relacion p = objSnaptshot.getValue(Relacion.class);
                if(p.getId_j() == id_J.get(position)){
                    db = FirebaseDatabase.getInstance().getReference().child("Relacion");
                    db.child(objSnaptshot.getKey()).removeValue();
                    borrarPreguntas(p.getId_p());

                }
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });

    borrarPuntos(id_J.get(position));
    db = FirebaseDatabase.getInstance().getReference().child("Juegos");
    db.child(String.valueOf(id_J.get(position))).removeValue();
}


public void borrarPreguntas(final int id_p){

    inicializarFirebase();
    db.child("Preguntas").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            for (DataSnapshot objSnaptshot : dataSnapshot.getChildren()) {
                if (Integer.parseInt(objSnaptshot.getKey()) == id_p) {
                    db = FirebaseDatabase.getInstance().getReference().child("Preguntas");
                    db.child(String.valueOf(id_p)).removeValue();
                }
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}

public void borrarPuntos(final int id_J){
    inicializarFirebase();
    db.child("Jugar").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            for (DataSnapshot objSnaptshot : dataSnapshot.getChildren()) {
                Jugar p = objSnaptshot.getValue(Jugar.class);
                if (p.getId_J() == id_J) {
                    db = FirebaseDatabase.getInstance().getReference().child("Juegos");
                    db.child(objSnaptshot.getKey()).removeValue();
                }
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
```

Figura 4.37: Método para borrar preguntas.

Figura 4.38: Opciones para editado.

Figura 4.39: Vista al momento de editar.

la que correspondía) luego se llamaba a la función “deleteRelation()” que buscaba dentro de la tabla “Relacion” el correspondiente elemento que contenía esa id de pregunta para ser borrado, vease la figura 4.40.

```
public void ResultadoConfirm(int position){
    databaseReference = FirebaseDatabase.getInstance().getReference().child("Preguntas");
    databaseReference.child(String.valueOf(ips.get(position))).removeValue();
    list.clear();
    deleteRelation(ips.get(position));
}


public void Resultado(String pregEdit, String respuesEdit, int posi){...}

public void check(){...}

public void deleteRelation(final int ipPregunta){
    inicializarFirebase();
    databaseReference.child("Relacion").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            for(DataSnapshot objSnaptshot: dataSnapshot.getChildren()){
                Relacion p = objSnaptshot.getValue(Relacion.class);
                if(p.getId_p() == ipPregunta){
                    databaseReference = FirebaseDatabase.getInstance().getReference().child("Relacion");
                    databaseReference.child(objSnaptshot.getKey()).removeValue();
                }
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}
```

Figura 4.40: Métodos para borrar preguntas/respuestas.

También otro detalle que se cuidó fue al momento de que el usuario presionara la tecla regreso, ya que podía dejar cambios inconclusos así que se programó ese botón para al ser presionado mandara un mensaje donde el usuario confirmaba que querí regresar a la vista anterior advirtiendo que los cambios se perderían, si el usuario confirmaba la elección entonces los cambios que había realizado eran borrados, esto era programado con el método “onKeyDown()” que detectaba cuando ese botón era presionado, como se muestra en la figura 4.41.

Otra de las opciones a elegir dentro de la vista de “Mis juegos“ es la opción para agregar un nuevo cuestionario, el cual cambio un poco en la forma de funcionar, ahora cuando el usuario presiona el botón despliega una ventana que pidiéndonos colocar el nombre y la descripción del juego a crear, dichos campos no pueden quedar vacíos si se quiere continuar con el proceso.

```
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_BACK && event.getRepeatCount() == 0) {
        new confirmBack( contexto: this, actividad: editarJuegoActivity.this);
        return true;
    }
    return super.onKeyDown(keyCode, event);
}

public void ResultadoBack() {
    for(int i=0;i<newPreg.size();i++) {
        databaseReference = FirebaseDatabase.getInstance().getReference().child("Preguntas");
        databaseReference.child(String.valueOf(newPreg.get(i))).removeValue();
        deleteRelation(Integer.parseInt(newPreg.get(i)));
    }
    preguntas.clear();
    respuestas.clear();
    list.clear();
    Intent intent = new Intent( packageContext: editarJuegoActivity.this, Main8Activity.class);
    intent.putExtra( name: "nickName",nickName);
    finish();
    startActivity(intent);
}
```

Figura 4.41: Programación del botón de regreso.

4.5.3 Opcion “Mis puntos”

Para “Mis puntos” se implementó un click largo, al momento de hacer esta acción al usuario se le mostraba una ventana preguntando si deseaba borrar esa pregunta al hacer la confirmación se llamaba a la función “ResultadoConfirm()” el cual recibía como parámetro la posición que se quería borrar, se buscaba dentro del arreglo que contenía las ids y se hacía referencia a la tabla “Jugar” para buscar el campo que tuviera la id y borrarlo(figura 4.42).

4.5.4 Modificación de los datos de Perfil

Como se mencionó antes existe ícono que nos manda a una parte donde el usuario puede cambiar su información (figura 4.43), es una vista muy simple pero en cuestión de funcionalidad cumple con todos los requisitos, primero tiene dos botones con forma de engranes a lado de la información como el correo y el *nickname*, al presionar alguno nos mandaba una ventana donde se le pedía al usuario escribir la nueva información ya fuese correo o el *nickname* y la contraseña campos que no podían quedar vacíos, también estan 3 botones, el primero para cambiar la contraseña, el segundo para cerrar la sesión y el tercero para borrar nuestra cuenta, todo cambio pide la contraseña como confirmación.

Para la clase encargada de hacer los cambios las funciones fueron muy parecerías ya que el proceso era el mismo, las cuales fueron “ResultadoCambio()” y “ResultadoCambioPass()”, al momento de hacer el cambio se corroboraba que el *nickname* (que se encuentra en todas

Figura 4.42: Borrar una puntuación.

Figura 4.43: Vista de la configuración del perfil.

las vistas al momento de que el usuario inicia sesión) y la contraseña proporcionada por el usuario fueran correctas, de no ser así se mandaba una altera de que el cambio no se había hecho ya que la contraseña era incorrecta, en caso de ser correcta se modificaba el objeto "user" de tipo "Persona" en la propiedad requerida, ya fuese con "user.setLogin()" para el *nickname* o "user.setCorreo()" para el correo o "user.setPassword()" para la contraseña, luego se hacía referencia a la tabla "Personas" y se le pasaba el objeto modificado para guardar los cambios, para el botón de "Cerrar sesión" solo se programó un método llamado "cerrarSesion()" el cual lo único que hacía era mandar al usuario a la vista inicial, vease figura 4.44.

```
public void cerrarSesion(View view){
    Intent intent = new Intent( packageContext: perfilActivity.this,Main3Activity.class);
    finish();
    startActivity(intent);
}

public void ResultadoCambio(String cambio, final Persona user, final  String idUser,final int pos){
    if(pos==0){
        user.setLogin(cambio);
    }
    else{
        user.setCorreo(cambio);
    }
    db = FirebaseDatabase.getInstance().getReference().child("Personas");
    db.child(idUser).setValue(user);
}

public void iniciarFirebase(){
    FirebaseApp.initializeApp(this);
    firebaseDatabase = FirebaseDatabase.getInstance();
    db = firebaseDatabase.getReference();
}

public void ResultadoCambioPass(String cambio, Persona p, String id){
    p.setPassword(cambio);
    db = FirebaseDatabase.getInstance().getReference().child("Personas");
    db.child(id).setValue(p);
}
```

Figura 4.44: Métodos para la configuración del perfil.

Eliminar una cuenta

Para el botón de "Borrar cuenta" las cosas eran más complejas ya que se tenía que borrar toda información dada o creada por el usuario (un proceso muy similar al de borrar juegos) El usuario debe ingresar su contraseña, una vez que ésta es verificada se le mandaba una confirmación donde podía aun retractarse, en caso de dar aceptar y asegurar que estaba seguro de su decisión se llamaba a la función "resultadoDelete()" las tablas que resultan afectadas son "Juegos" todos los juegos creados por el usuario, se aplica un borrado en cascada con la función "borrarRelacion" encargado de buscar el id las preguntas que pertenecían a un juego en la tabla "Relacion", ya que se obtenían las ids estas pasaban como parámetro

a la función “borrarPreguntas()” que a través de la tabla “Preguntas” y el id buscaba las preguntas del juego, con todos estos datos identificados se comenzaban a borrar uno por uno, se borraba el juego, se borraba la relación y todas las preguntas relacionadas al juego, después se procedía a borrar los puntos que el usuario había conseguido a lo largo de la interacción de la aplicación mediante la función “borrarPuntos()” que recibía como parámetro el id del juego, paplicada en cascada en la tabla “Jugar” de la base de datos todos los datos donde el id del juego coincidiera con el pasado recibido como parámetro para proceder a borrarlo, como se muestra en la figuras figuras 4.45 y 4.46.

```
public void resultadoDelete(final Persona userDelete, final String idUser){
    db.child("Juegos").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            for(DataSnapshot objSnaptshot: dataSnapshot.getChildren()){
                Juego p = objSnaptshot.getValue(Juego.class);
                if(p.getNickName().equals(userDelete.getLogin())){
                    db = FirebaseDatabase.getInstance().getReference().child("Juegos");
                    db.child(objSnaptshot.getKey()).removeValue();
                    borrarRelacion(p.getPin());
                    borrarPuntos(Integer.parseInt(p.getPin()));
                }

            }
            db = FirebaseDatabase.getInstance().getReference().child("Personas");
            db.child(idUser).removeValue();
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}

public void borrarRelacion(final String id_J){
    iniciarFirebase();
    db.child("Relacion").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            for(DataSnapshot objSnaptshot: dataSnapshot.getChildren()){
                Relacion p = objSnaptshot.getValue(Relacion.class);
                if(p.getId_j() == Integer.parseInt(id_J)){
                    db = FirebaseDatabase.getInstance().getReference().child("Relacion");
                    db.child(objSnaptshot.getKey()).removeValue();
                    borrarPreguntas(p.getId_p());
                }
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}
```

Figura 4.45: Primera parte del proceso para borrar un perfil.

4.6 Restricción en la navegación entre pantallas en la aplicación

Anteriormente se habló de la preocupación de que el usuario dejara procesos incompletos al momento de navegar en la aplicación, esto sucede cuando se presiona la tecla de navegación hacia atrás para evitar que esto pasara en dos vistas se condiciono el uso de ese botón, la primera fue cuando el jugador se encontraba jugando y media presionaba el botón, mediante el método “onKeyDown()” se lanzaba una ventana al usuario donde se le preguntaba si estaba seguro de querer abandonar la partida, si el respondía que sí entonces es mandado a la vista donde se encontraban sus juegos y puntuaciones, borrando los datos de la partida inconclusa,

```
    public void borrarPreguntas(final int id_p){
        iniciarFirebase();
        db.child("Preguntas").addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                // datos.clear();
                for (DataSnapshot objSnaptshot : dataSnapshot.getChildren()) {
                    //       Toast.makeText(getContext(),objSnaptshot.getKey(), Toast.LENGTH_SHORT).show();
                    if (Integer.parseInt(objSnaptshot.getKey()) == id_p) {
                        db = FirebaseDatabase.getInstance().getReference().child("Preguntas");
                        db.child(String.valueOf(id_p)).removeValue();
                    }
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });
    }

    public void borrarPuntos(final int id_Pin){
        iniciarFirebase();
        db.child("Jugar").addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                // datos.clear();
                for (DataSnapshot objSnaptshot : dataSnapshot.getChildren()) {
                    //       Toast.makeText(getContext(),objSnaptshot.getKey(), Toast.LENGTH_SHORT).show();
                    Jugar puntos = objSnaptshot.getValue(Jugar.class);
                    if (puntos.getId_J() == id_Pin) {
                        db = FirebaseDatabase.getInstance().getReference().child("Jugar");
                        db.child(objSnaptshot.getKey()).removeValue();
                    }
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });
    }
}
```

Figura 4.46: Segunda parte del proceso para borrar un perfil.

la otra parte donde se utilizó esté método fue en la vista que se mostraba una vez que el usuario terminaba un juego, esto para evitar que al regresarse comenzara el juego de nuevo y se mandaran datos incoherentes a la base de datos, en vez de esto era mandado a la vista donde se visualizan los juegos y puntuaciones, como se muestra en la figura 4.47.

```
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_BACK && event.getRepeatCount() == 0) {
        new confirmBack( contexto: this, actividad: MainActivity.this);
        return true;
    }
    return super.onKeyDown(keyCode, event);
}

public void ResultadoBack() {
    Intent intent = new Intent( packageContext: MainActivity.this, Main8Activity.class);
    intent.putExtra( name: "nickName",nickName);
    finish();
    startActivity(intent);
}
```

Figura 4.47: Programación del botón de regreso al momento de terminar un juego.

4.7 Sistema Multijugador en tiempo real

La aplicación para este entonces ya estaba muy completa y cumplía de manera muy satisfactoria toda la funcionalidad, era una aplicación sencilla pero cumplía con su objetivo pero ¿qué es de un juego sin un sistema online?. Hoy en día el avance de la internet ha permitido que todo tipo de consolas puedan contar con una versión online de sus juegos, un sistema donde el usuario pueda jugar con otros usuarios sin la necesidad de estar juntos o en la misma consola, ahí el usuario pone a prueba sus habilidades con otro grupo de personas haciendo de los juegos algo más competitivo e interesante, esto motiva al usuario a practicar para mejorar sus habilidades. Entonces comenzó la parte más ambiciosa de todo el proyecto, crear un sistema de multijugador *online*.

Existían todas las herramientas para hacerlo, Firebase actualizaba las vistas automáticamente cuando sucedía un cambio en la base de datos y el juego ya funcionaba en un modo solitario, solo era unir ambas cosas dentro de una misma herramienta.

En esta parte se explicará el proceso para conseguir una conexión multijugador, los detalles se guardarán para la explicación del código final. Comenzamos creando la conexión entre dos celulares, la idea inicial era hacer que Firebase detectara primero que había usuarios conectados a una misma "sala" (éste fue el nombre que recibió la tabla en la que se almacenarían los datos y que sería la encargada de crear la interacción entre jugadores) a través de un "*pin* de sala" era como se accesaría a los datos de la sala. Para esto debía existir un usuario que fuera el que creara la sala al cual se le daba una clave única para compartir con los usuarios que desearan jugar, los invitados debían colocar el *"pin"* dentro del buscador para ser redirigidos a una vista muy similar a la del jugador solitario.

Una vez que se detectaba cuantos usuarios se conectaban el siguiente paso fue que todos tuvieran los mismos datos en el mismo orden, este fue un paso muy complejo ya que hasta ese momento no se había cargado/descargado arreglos de datos a la base de datos, luego se hizo la parte más importante la cual era que un jugador hiciera un movimiento y esté se viera reflejado en la pantalla de los otros usuarios (que fue el ¡Boom! de la aplicación). Para este punto teníamos todo lo necesario, múltiples usuarios estaban conectados a una misma tabla de información la cual al descargarse en su celular era la misma para todos en el mismo orden y cuando uno de ellos hiciera un movimiento el celular de los demás los verían en tiempo real. Ya solo faltaban agregar detalles como comenzar cuando decidieran, saltar el turno de un jugador después de cierto tiempo, que solo el usuario del cual era turno pudiera hacer un movimiento, que la sala no admitiera más usuarios una vez comenzada la partida, identificar cuando el juego había terminado y dar un ganador se sufrió un inconveniente al final del proceso pues la conexión era inestable ocasionando problemas al momento de jugar, así que se decidió para hacer una versión más estable que la forma de juego sería 1 vs 1 y no

“n” jugadores como se había decidido de un inicio, de esta manera los problemas se reducían bastante dando una experiencia al usuario mucho más agradable. Este texto relata de manera breve semanas de experimentación e investigación.

Tabla Multijuego

Se tuvo que crear una nueva tabla y un nuevo objeto para la base de datos el cual se nombró “Multijuegos” y “Multijuego” (figura 4.48) (Recordando el plurar hace referencia a la tabla de la base de datos y el singular al objeto dentro de la aplicación) como la lista de ítems “posicion” un interto que nos permitiría saber cuál fue la carta seleecionada, “pinSala” que es un string donde se almacena la clave única para poder acceder a los datos, “pinJuego” entero donde se almacenaba el “pin” del juego que sé había seleccionado, “numConect” dato de tipo entero que donde se guardaba la cantidad de personas conectadas en la sala, “turno” entero el cual iba incrementando conforme un jugador hacía su jugada que nos permitía saber a qué jugador le tocaba hacer un movimiento, “maxPuntuacion” entero que guardaba la puntuación del ganador, “ganador” datos de tipo string que guarda al usuario que resultó con mayor puntuación, “datos” arreglo de tipo “Datos” donde se guardaban los datos de juego que se había seleccionado (preguntas y respuestas) cada uno contaba con su respectivo método “*get*()” y “*set*()”.

A las vistas donde se podía iniciar un juego se le agregó la opción de multijugador en forma de menú en la parte superior inferior se creaba el “pin de sala” con una combinación de números y letras, la cadena estaba compuesta por el “pin” del juego seleccionado concatenado con su posición dentro de la tabla seguido de la palabra “-pty” está ultima cadena era lo que diferenciaba a una sala de un juego como puede verse en las figuras 4.49 y 4.50.

Una vez que el usuario daba en “aceptar” se extraían los datos del juego que se había seleccionado y se guardaban dentro de un arreglo llamado “datos” a través de la función “agregarPreguntas()” (figura 4.51), para después pasarlo como parámetro al objeto de tipo “Multijuego” llamado “sala” juntos con otros datos y haciendo referencia a la tabla “Multijuegos”, se creaba un nuevo campo el cual era a lo que llamábamos “sala de juego”, se llamaba a una nueva vista en donde se pasaba el “id” del jugador, el “id” de la sala, el *“nickname”* del jugador y un “numJug” que era el número de jugador, la adaptación de “ResultadoConfirm()” se puede ver en la figura 4.52.

Como se mencionó anteriormente a cada sala se le asignaba un pin único para ser compartido con el usuario que se deseaba competir. Para que el otro usuario pudiese conectarse a la sala se modificó la opción de búsqueda dentro de la vista de “Mis puntos” para que pudiera reconocer el pin.

Dentro del método “Resultado2()” (figura 4.52) se agregó una nueva condición, si el string

```
public class Multijuego {
    int posicion;
    String pinSala;
    int pinJuego;
    int numConect;
    int turno;
    int maxPuntuacion;
    String ganador;
    boolean start;
    ArrayList<Datos> datos;
    boolean surrender;
    public Multijuego(){

    }

    public Multijuego(int pos,String pS,int pJ, int nC,int trn, int maxPun, String gan, boolean st, boolean sr){
        posicion = pos;
        pinSala = pS;
        pinJuego = pJ;
        numConect = nC;
        turno = trn;
        maxPuntuacion = maxPun;
        ganador = gan;
        start = st;
        surrender = sr;
    }

    public int getPosicion() { return posicion; }
    public void setPosicion(int pos) { posicion = pos; }
    public String getPinSala() { return  pinSala; }
    public void setPinSala(String pS) { pinSala  = pS; }
    public int getPinJuego() { return pinJuego; }
    public void setPinJuego(int pJ) { pinJuego = pJ; }
    public int getNumConect() { return numConect; }
    public void setNumConect(int nC) { numConect = nC; }
    public int getTurno() { return turno; }
    public void setTurno(int trn) { turno = trn; }
    public int getMaxPuntuacion() { return maxPuntuacion; }
    public void setMaxPuntuacion(int max) { maxPuntuacion = max; }
    public String getGanador() { return ganador; }
    public void setGanador(String gan) { ganador = gan; }
    public boolean getStart() { return  start; }
    public void setStart(boolean st) { start = st; }
    public boolean getSurrender(){ return surrender; }
    public void setSurrender(boolean sr){ surrender = sr; }
}
```

Figura 4.48: Objeto “Multijuego“.

Figura 4.49: Opciones de juego.

Figura 4.50: Confirmación de la opción.

```
public void agregaPreguntas(final int id_p,final int idSala){
    inicializarFirebase();
    db.child("Preguntas").addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            for (DataSnapshot objSnaptshot : dataSnapshot.getChildren()) {
                if ( Integer.parseInt(objSnaptshot.getKey()) == id_p) {
                    Pregunta p = objSnaptshot.getValue(Pregunta.class);
                    datos.add(new Datos(id_p,R.drawable.carta, imagenR: 10000,p.getPregunta(), check: false, state: true, pts: 100));
                    datos.add(new Datos(id_p,R.drawable.carta,p.getImagen(),p.getRespuesta(), check: false, state: true, pts: 100));
                }
            }

            Random rgen =  new  Random ( ) ;   // Generador de números aleatorios
            int randomPosition;
            Datos temp;
            for  ( int i = 0 ; i < datos.size(); i ++ )  {
                randomPosition = rgen. nextInt (datos.size()) ;
                temp = datos.get(i);
                datos.set(i,datos.get(randomPosition));
                datos.set(randomPosition,temp);
            }
            datosMultijuego salaDatos = new datosMultijuego(datos, pS: String.valueOf(idSala)+String.valueOf(maxIdM)+"-pty");
            db = FirebaseDatabase.getInstance().getReference().child("DatosMultijuegos");
            db.child(String.valueOf(maxIdJM+1)).setValue(salaDatos);

        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}
```

Figura 4.51: Método “agregarPreguntas()“ adaptado.

```
public void ResultadoConfirm(Integer idSala){
    listaPreguntas(idSala);

    Multijuego sala  = new Multijuego( pos: -1, pS: String.valueOf(idSala)+String.valueOf(maxIdM+1)+"-pty",idSala, nC: 1, trn: 0, maxPun: 0, gan: " ", st: false, in: false);
    db = FirebaseDatabase.getInstance().getReference().child("Multijuegos");
    db.child(String.valueOf(maxIdM+1)).setValue(sala);
    Intent intent = new Intent( packageContext: this,MainMultijuego.class);
    intent.putExtra( name: "id_J",idSala);
    intent.putExtra( name: "idSala", value: String.valueOf(idSala)+String.valueOf(maxIdM+1)+"-pty");
    intent.putExtra( name: "nickName",nickName);
    intent.putExtra( name: "nombre",titulo);
    intent.putExtra( name: "numJug", value: 1);
    startActivity(intent);
}
```

Figura 4.52: Método “ResultadoConfirm()“adaptado.

ingresado no correspondía a la tabla “Juegos” dentro de la base de datos se llamaba a la tabla “Multijuegos”.

```
db.child("Multijuegos").addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        String idSala = "";
        Multijuego j;
        int numJug = 0;

        if(!fg) {
            for (DataSnapshot objSnapshot : dataSnapshot.getChildren()) {
                j = objSnapshot.getValue(Multijuego.class);
                if (pinGame.equals(j.getPinSala())) {
                    if (j.getNumConect() <= 2){
                        fg = true;
                        idSala = j.getPinSala();
                        numJug = (j.getNumConect() + 1);
                         j.setNumConect(numJug);
                        db = FirebaseDatabase.getInstance().getReference().child("Multijuegos");
                        db.child(String.valueOf(objSnapshot.getKey())).setValue(j);
                    }
                    else{
                        fg = false;
                        Toast.makeText( context: Main8Activity.this,  text: "Esta sala se encuentra en juego", Toast.LENGTH_SHORT).show();
                    }
                }
            }
            if (fg) {
                Intent intent = new Intent( packageContext: Main8Activity.this,MainMultijuego.class);
                intent.putExtra( name: "idSala",idSala);
                intent.putExtra( name: "nickName",nickName);
                intent.putExtra( name: "numJug",numJug);
                finish();
                startActivity(intent);
            }
        }
    }

    @Override
```

Figura 4.53: condición agregada al metodo “Resultado2()“.

Ya que ambos usuarios se encontraban conectados a la sala se ejecutaba el archivo “MainMultijuego.java” se declaraban las variables para extraer los datos de los parámetros y se inicializaban algunos con valor de cero o *true*. El primer paso era llamar a la base de datos y hacer referencia dentro de la tabla “Multijuegos” para después revisar el que el juego no hubiese terminado mediante la variable booleana (esto para evitar que se hicieran llamadas recursivas a la base de datos al terminar el juego), una vez encontrado los dato de la sala se procedía a extraer los datos del juego (Preguntas y respuestas) mediante el método “llenarLista()”.

Luego en caso de que solo un jugador se encontrara conectado se le mostraba un letrero rojo indicándole que esperara a su contrincante y no se dejaba hacer ningún movimiento (figura 4.53). Con los dos usuarios en juego se procedía a iniciar el método “tiempoAgotado()” si

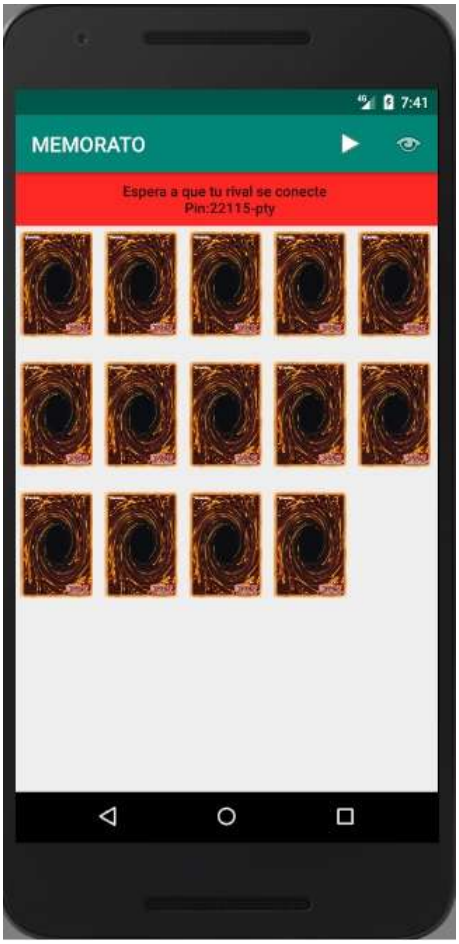


Figura 4.54: Vista de espera.

la variable *"start"* se encontraba en *true* esto nos indicaba que el tiempo se había acabado y un jugador no había realizado sus movimientos dentro de lapso de tiempo determinado (8 segundos) entonces se recorrían todas las cartas y se buscaba alguna con el *"check"* y el *"state"* en *true* (Una carta que se encontrara volteada y sin pareja encontrada) esto para cuando el usuario solo volteara una carta. Luego se llamaba al método "modificacion()" el cual iniciaba el contador de 8 segundos, si ese contador no era detenido se mandaba un mensaje diciendo que el tiempo se había acabado y se cambia el valor de las variables "posicion" a -1, "turno" se incrementaba en uno y *"start"* en true como se puede muestra en la figura 4.54.

```
public void tiempoAgotado(Multijuego m){
    if(m.getStart()) {
        for (int i = 0; i < datos.size(); i++) {
            if (adap.getCheck(i) == true && adap.getState(i) != false) {
                Toast.makeText(getApplicationContext(), text: "El tiempo del jugador se termino", Toast.LENGTH_SHORT).show();
                adap.setCheck(i, check: false);
                frame = (FrameLayout) grilla.getChildAt(i);
                cart = (ImageView) frame.findViewById(R.id.imgP);
                cart.animate().alpha(1f).setDuration(500);
                m.setStart(false);
            }
        }
    }
    Toast.makeText(getApplicationContext(), text: "El tiempo del jugador se termino", Toast.LENGTH_SHORT).show();
}

public void modificacion(final Multijuego finalM, final String key){
    handler2.postDelayed(() -> {
            if((finalM.getTurno() % 2) + 1 == numJug){
                Toast.makeText(getApplicationContext(), text: "El tiempo del jugador se termino", Toast.LENGTH_SHORT).show();
                finalM.setPosicion(-1);
                finalM.setTurno(finalM.getTurno() + 1);
                finalM.setStart(true);
                database = FirebaseDatabase.getInstance().getReference().child("Multijuegos");
                database.child(key).setValue(finalM);
            }
    }, delayMillis: 8000);
}
```

Figura 4.55: Métodos para terminar el turno de un usuario.

Después se verificaba a que jugador le correspondía el turno, la variable "turno" se le sacaba el módulo 2 y se le sumaba 1, si este número correspondía al número de jugador asignado al usuario entonces era su turno haciéndole saber esto con un letrero verde con un texto, el usuario al hacer un movimiento provocaba que la variable "posicion" tomara el valor seleccionado y se mandara a la base de datos, está variable era la encargada de decir al dispositivo del otro usuario en donde se había hecho el movimiento para llamar en ambos celulares el método "juego()", en caso de no ser turno del usuario el letrero se muestra en color rojo con un texto informando que un no debe hacer movimiento y en caso de hacer un movimiento se le informa que aún no es momento de hacerlo y las cartas siguen boca abajo, vease la figura 4.55.

El método "juego" cumple con la misma función que en el modo solitario, funciona exactamente igual al momento de voltear las cartas y revelar el contenido por con unas ligeras modificaciones

```
if (m.getPinSala().equals(sala)) {
    if (llenado == true) {
        llenarLista();
        llenado = false;
    }
    if (m.getNumConect() != 2) {
        SpannableString texto = new SpannableString( source: "Espera a que tu rival se conecte \n  Pin:" + m.getPinSala());
        estado.setText(texto);
        estado.setBackgroundColor(Color.rgb( red: 253,  green: 39,  blue: 36));
    } else {
        if (m.getNumConect() == 2) {
            surrender(m.getSurrender());
            tiempoAgotado(m);
            modificacion(m, key);
            if ((m.getTurno() % 2) + 1 == numJug) {
                estado.setText("Es tú turno");
                estado.setBackgroundColor(Color.rgb( red: 11,  green: 255,  blue: 41));
                final Multijuego finalM = m;

                grilla.setOnItemClickListener((parent, view, position, id) -> {
                        finalM.setPosicion(position);
                        database = FirebaseDatabase.getInstance().getReference().child("Multijuegos");
                        database.child(key).setValue(finalM);
                        juego(position, finalM, key);
                });
            } else {
                estado.setText("Es el turno de tu contrincante");
                estado.setBackgroundColor(Color.rgb( red: 253,  green: 39,  blue: 36));
                grilla.setOnItemClickListener((parent, view, position, id) -> {
                        Toast.makeText(getApplicationContext(),  text: "Aún no es tu turno", Toast.LENGTH_SHORT).show();
                });
                if (m.getPosicion() > -1) {
                    juego(m.getPosicion(), m, key);
                }
            }
        }
    }
}
```

Figura 4.56: Código para hacer los movimiento y decidir turnos.

al momento de finalizar el turno, la variable “posicion” se le da el valor de -1, el “turno” se aumenta en 1 y el contador de tiempo se detiene.

Para el método “fin()” (figura 4.56) que es el que nos indica si el juego terminó se hicieron modificaciones, haciendo referencia a la tabla “Multijuegos“ se verificaba que la puntuación obtenida por el usuario fuera mayor que la máxima registrada, si era así su puntuación se registraba como la máxima obtenida junto con su *nickname*, esto para dar a conocer a ambos jugadores quien resultó ganador, luego se llamaba a una vista donde se informaba al ganador véase la figura 4.57.

```
public void fin(int jts1){
    handler2.removeCallbacksAndMessages( token: null);
    if(jts1 == datos.size()){
        inicializarFirebase();
        bu = false;
        database.child("Multijuegos").addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                for(DataSnapshot obj: dataSnapshot.getChildren()){
                    Multijuego j = obj.getValue(Multijuego.class);
                    if(j.getPinSala().equals(sala)){
                        if(j.getMaxPuntuacion()< total ) {
                            j.setMaxPuntuacion(total);
                            j.setGanador(nickName);
                            j.setNumConect(0);
                            database = FirebaseDatabase.getInstance().getReference().child("Multijuegos");
                            database.child(obj.getKey()).setValue(j);
                        }
                    }
                }
            }
            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });
        handler2.removeCallbacksAndMessages( token: null);
        new finJuegoMulti(contexto, actividad: MainMultijuego.this,sala,total);
    }
    else {
        Toast.makeText(getApplicationContext(), text: "Van " + jts1 + " cartas de " + datos.size(), Toast.LENGTH_SHORT).show();
    }
}
```

Figura 4.57: Método “fin()” adaptado.

Una vez que el usuario daba en “Aceptar” era redirigido a la ventana de inicio indicando que el juego había terminado como en la figura.

Para esta modalidad se tuvo que cuidar algunos aspectos, el primero fue condicionar el botón de regreso (figura 4.58), cuando el usuario lo presiona aparece una notificación pidiendo la confirmación del evento e indicando que se dará automáticamente la victoria al usuario adversario (figura 4.59), cuando se acepta la salida el otro usuario es notificado que su contrincante abandonó la partida y que él resultó ganador.

Otra de los aspectos fue que si el conjunto de cartas superaba el tamaño de pantalla al momento de deslizar hacía abajo para hacer un movimiento con las cartas de abajo esté no podría ser reflejado en la pantalla del otro usuario entonces para resolver este problema de agrego un botón que hiciera las cartas de tamaño más chico de tal manera que todas se

Figura 4.58: Vita para un juego finalizado.

```
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_BACK && event.getRepeatCount() == 0) {
        new confirmBack( contexto: this, actividad: MainMultijuego.this);
        return true;
    }
    return super.onKeyDown(keyCode, event);
}

public void ResultadoBack() {
    handler2.removeCallbacksAndMessages( token: null);
    sala="";
    total = 0;
    jts = 0;
    llenado = true;
    datos.clear();

    Intent intent = new Intent( packageContext: MainMultijuego.this, Main8Activity.class);
    intent.putExtra( name: "nickName",nickName);
    finish();
    super.onDestroy();
    startActivity(intent);

}

public void surrender(boolean sr){
    if(sr){
        Toast.makeText(getApplicationContext(),
                text: "¡En hora buena! Tu contrincante abandono la partida, seras mandado a la pantalla de inicio",
                Toast.LENGTH_SHORT).show();
        playAgain();
    }
}
```

Figura 4.59: Restricción del botón de regreso.

Figura 4.60: Vita para salir del juego.

ajustaran al tamaño de la pantalla y no se necesitara deslizar hacía abajo, a esta función se le llamo “Reducción”, vease las figuras 4.60, 4.61 y 4.62.

```
public void reduccion(){
    if(datos.size() != 0) {
        int numero =1;
        for(int k=1;k<datos.size()/2;k++){
            if(k*k == datos.size()){
                numero = k;
            }
            else if(k*k < datos.size()){
                numero = k+1;
            }
        }
```

Figura 4.61: Primera parte del método “reduccion()“.

```
grilla.setNumColumns(numero+1);
DisplayMetrics metrics = new DisplayMetrics();
getWindowManager().getDefaultDisplay().getMetrics(metrics);
int width = metrics.widthPixels; // ancho absoluto en pixels
int height = metrics.heightPixels;

int tamWid = width / (numero+1);
int tamHeig = (height - 52) / (numero+1);
```

Figura 4.62: Segunda parte del método “reduccion()“.

Y de esa manera la experiencia de un multijugador online quedaba completa.

Para finalizar el desarrollo de la aplicación se pensó en mejorar la estética de la aplicación haciéndola más atractiva visualmente para el usuario sin perder la simplicidad, para hacer las cosas, se contactó a la diseñadora Arianna Arriategui Tume (MuffiHF por su nombre artistico) para ayudar con el diseño de la vista de inicio.

Hasta el momento la aplicación no tenía nombre ni logo y en un momento de debate y creatividad respecto al tema se le decidió dar el nombre de “Memorato” para después crear el logo que se pondría en la aplicación dentro del ícono que se mostraría en el celular, como se ve en la figura 4.63.

Y el resultado final de la vista de inicio fue el siguiente presentado en la figura 4.64.

Ya con estos cambios realizados estéticos realizados, con un nombre y logo, Memorato se encontraba listo para pasar a la fase de experimentación.

```
        frame = (FrameLayout) grilla.getChildAt( index: 0);
        if (frame != null) {
            for (int l = 0; l < datos.size(); l++) {
                frame = (FrameLayout) grilla.getChildAt(l);
                AbsListView.LayoutParams paramsFram = new AbsListView.LayoutParams(tamWid,  h: tamHeig-15);
                FrameLayout.LayoutParams paramsImg = new FrameLayout.LayoutParams( width: tamWid-30,  height: tamHeig-15);
                if (frame != null) {
                  //  frame.setLayoutParams();
                    cart = (ImageView) frame.findViewById(R.id.imgP);
                    leo = (ImageView) frame.findViewById(R.id.imgR);
                    resp = (TextView) frame.findViewById(R.id.textR);
                    cart.setLayoutParams(paramsImg);
                    leo.setLayoutParams(paramsImg);
                    resp.setLayoutParams(paramsImg);
                    frame.setLayoutParams(paramsFram);
                    FrameLayout frame2 = (FrameLayout) frame.findViewById(R.id.FrameCarta);
                    frame2.setLayoutParams(paramsFram);
                    int letra = (tamHeig / 30);
                    resp.setTextSize(letra);
                }
            }
        }
    }
```

Figura 4.63: Tercera parte del método “reduccion()“.

Figura 4.64: Logo.

Figura 4.65: Vista de inicio rediseñada.

Capítulo 5

Experimentación

Para la experimentación se tuvo que considerar dos roles de los usuarios, cuando el usuario crea un memorama lo llamaremos docente y cuando esya jugando lo llamaremos alumno. Para el usuario docente se observa la interacción con la aplicación, la facilidad de uso y su experiencia para crear juegos, por su parte para el usuario se observaría su experiencia para conseguir jugar, la forma de jugar y que tanto se aprendía cuando interactuaba con la aplicación.

Para escoger a los usuarios docentes y los usuarios alumnos se pidio la colaboración de personal (alumnos y docentes) del "Instituto Soul Maya" con la preferencia de que ambos grupos pertenecieran a la misma aula de clase, esto con la finalidad de que los temas escogidos para trabajar fueran algunos de los vistos en clase y que los docentes pudieran crear los cuestionarios de acuerdo a sus necesidades dentro del aula y que los alumnos estuvieran familiarizados con los temas intentado que fueran a la par con los vistos recientemente en clases.

5.1 Analizando al Usuario con el rol de Docente

Por el lado del usuario docente se tomaron a los que impartían los cursos de biología, química y español para intentar experimentar con una variedad de temas. Antes de comenzar con la explicación se les pidió llenaran un pequeño cuestionario donde se les preguntaba sobre su experiencia como docente, si utilizaban alguna herramienta tecnológica dentro de sus clases, la experiencia que habían tenido con ella los resultados obtenidos y su opinión sobre usar el Smartphone como una herramienta de aprendizaje, en la figura 5.1 y 5.2 'se muestra la encuesta aplicada.

Después se procedió a explicarles un poco en qué consistía la aplicación y la idea sin entrar

Cuestionario de diagnóstico (Profesores)

Nombre:

Edad:

1.- ¿Cuántos años tiene cómo docente?

R:

2.- ¿Utiliza las TIC para el desarrollo de sus clases?, si la respuesta es sí, entonces por favor, díganos.

R:

¿De qué manera utilizas la tecnología para el desarrollo de las clases?

R:

¿Cuáles son las herramientas tecnológicas que utilizas?

R:

¿Cuál ha sido la experiencia al momento de implementarlas?

R:

¿Considera que utilizando las TIC el nivel de comprensión de sus alumnos es satisfactorio?

R:

¿Conoce alguna aplicación móvil que le sirva de auxiliar en sus clases? De ser así, menciónela.

R:

¿Qué tan viable considera el uso del Smartphone cómo herramienta de refuerzo para el aprendizaje de sus estudiantes?

R:

¿Le gustaría tener una app lúdica que pueda personalizar para reforzar los temas vistos en clase?

R:

Figura 5.1: Encuesta de diagnóstico para profesores.

en detalles para permitir que ellos descubrieran todas las funciones, dándoles un pequeño ejemplo con papeles donde se escribieron preguntas y respuestas de sus materias. Una vez captada la idea y la intención se les instaló la aplicación en sus celulares y mostrando un poco de la funcionalidad (lo más básico). Se les pidió que crearan juegos para los alumnos y que elaboraran cuestionarios/exámenes sobre los juegos que crearían, con la intención de que sirvieran para medir el conocimiento de los alumnos antes y después de usar la aplicación. Los docentes tuvieron que registrarse dentro de la aplicación, procedieron a elaborar los juegos y los cuestionarios/exámenes eligiendo como temas "Elementos" y "Tabla periódica" por el área de química, "Estructura celular" y "Función de organelos" para biología y "Verbos y adjetivos" para español, vease las figuras 5.3-5.9.

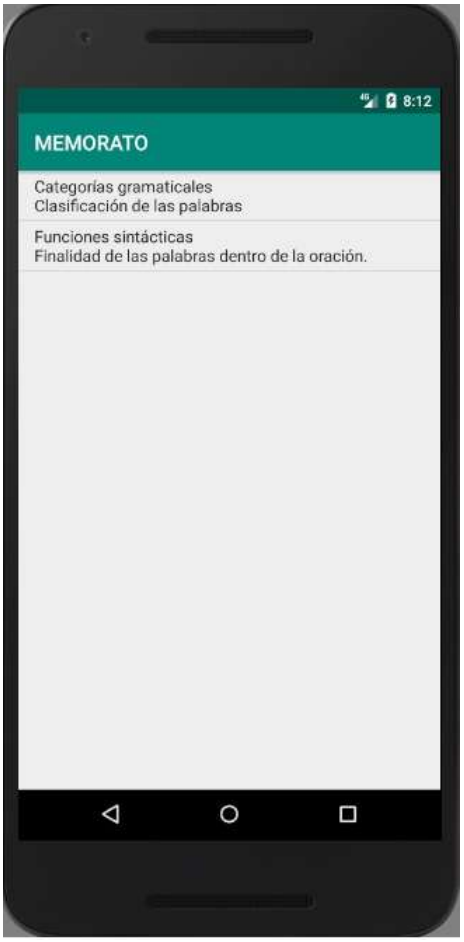


Figura 5.2: Perfil de español.

Una vez que los docentes alimentaron a la base de datos y terminaron con su experiencia con la aplicación se les dió un cuestionario donde se les preguntó sobre su interacción con la aplicación, las ventajas de uso y las mejoras que para ellos pudiera tener, así como la comodidad y facilidad para usarla, como se muestra en la figura 5.10 y 5.11.

Figura 5.3: Perfil de biología.

Examen 1 pin: 8

QUÍMICA ELEMENTOS

Nombre:

1.- ¿Cómo se llama la partícula subatómica con carga negativa?

R:

2.- ¿Cómo se llama la partícula subatómica con carga positiva?

R:

3.- ¿Cómo se llama la partícula subatómica con carga nula?

R:

4-. ¿Cuál es partícula subatómica que se encuentra girando alrededor del núcleo?

R:

5.- ¿El número atómico nos dice la cantidad de..... en el núcleo del átomo?

R:

6.- en un átomo neutro la cantidad de protones es igual al número de...

R:

7-. Si un átomo de un elemento gana electrones su carga será negativa y se conoce como...

R:

8-. Si un átomo de un elemento pierde electrones su carga será positiva y se conoce como...

R:

9-. Un elemento con el mismo número atómico y diferente masa se conoce como...

R:

10-. Son los isotopos de hidrogeno

R:

Figura 5.4: Examen 1 de química.

Examen 2 pin: 9

QUÍMICA TABLA PERIÓDICA

Nombre:

1-. ¿Las filas en horizontal en la tabla periódica se conocen cómo?

R:

2-. ¿Las filas en vertical en la tabla periódica se conocen cómo?

R:

3-. ¿Qué es la electronegatividad?

R:

4-. ¿Cómo aumenta la electronegatividad en la tabla periódica?

R:

5-. ¿Cómo aumenta la energía de ionización en la tabla periódica?

R:

6-. ¿Cómo aumenta la carga nuclear efectiva en la tabla periódica?

R:

7-. ¿Cuál es el elemento más electronegativo en la tabla periódica?

R:

8-. ¿Cuál es el elemento menos electronegativo en la tabla periódica?

R:

9-. ¿Cómo aumenta el radio atómico en la tabla periódica?

R:

10-. ¿Principal clasificación de los elementos en la tabla periódica?

R:

Figura 5.5: Examen 2 de química.

Examen 1 pin: 10

BIOLOGIA ESTRUCTURA CELULAR

Nombre:

1-. Unidad minima estructural de la vida.

R:

2-. Organelo que rodea y delimita la celular.

R:

3-. Apariencia semiliquida rica en azucares en el cual encuentro los organelos celulares.

R:

4-. Organelo responsable de contener la información genética.

R:

5-. Organelo visualizado como conjunto de sacos aplanados y aspecto rugoso.

R:

6-. Organelo visualizado como conjunto de sacos aplanados y aspecto liso.

R:

7-. Organelo el cual recibe su nombre gracias a su descubridor CAMILO GOLGI.

R:

8-. Organelo contituido por proteínas y ARNr

R:

9-. Organelo conocido como central de energía

R:

10-. Organelo presente en las células vegetales y constituye la mitad del tamaño de las células.

R:

11-. Da soporte a los organelos en el interior de la celula.

R:

Figura 5.6: Examen 1 de biología.

Examen 2 pin: 11

BIOLOGIA FUNCION DE ORGANELOS

Nombre:

1-. La membrana celular permite la entrada de ciertas sustancias por lo que se dice que es...

R:

2-. En el citosol se realiza la ruta metabolica del azúcar hasta piruvato conocida como.

R:

3- Acido nucleico que contiene la información genética para realizar proteínas.

R:

4-. Acido nucleico que lleva la información genética hasta los ribosomas.

R:

5-. Los nucleótidos se clasifican en.

R:

6-. Cantidad de amino acidos necesarios para generar una proteína

R:

7-. El ribosomas tiene la función de sintetizar...

R:

8-. El aparato de Golgi sirve para.

R:

9-.En la mitocondria sucede el ciclo de.

R:

10-. ¿Cuantos apts genero de una molecula de glucosa?

R:

Figura 5.7: Examen 2 de biología.

Examen de español

Nombre:

1. ¿Cuál es la función del verbo?
a) Núcleo del sujeto b) Indicar acciones, estados o existencia
c) Se presenta en el sujeto indefinido

2. ¿A qué llamamos complemento circunstancial?
a) Al lugar donde se presenta la acción
b) Determina las circunstancias en las que se realiza la acción
c) Requiere un complemento directo

3. Formación del predicado nominal
a) Presenta verbos copulativos y una cualidad del sujeto nombrada atributo
b) Contiene accidentes gramaticales c) Se conforma con complementos directos

4. La función del adjetivo es la siguiente:
a) Complementar al verbo b) Modificar al nombre c) Identifica al sujeto

5. El sujeto indefinido se reconoce porque...
a) Se presenta en modo imperativo b) No se distingue entre sujeto y predicado
c) No se puede determinar dentro de la oración

6. Forma de identificar al complemento directo
a) Recibe la acción directa del verbo b) Antecede al artículo determinado "la"
c) Se presenta con verbos intransitivos

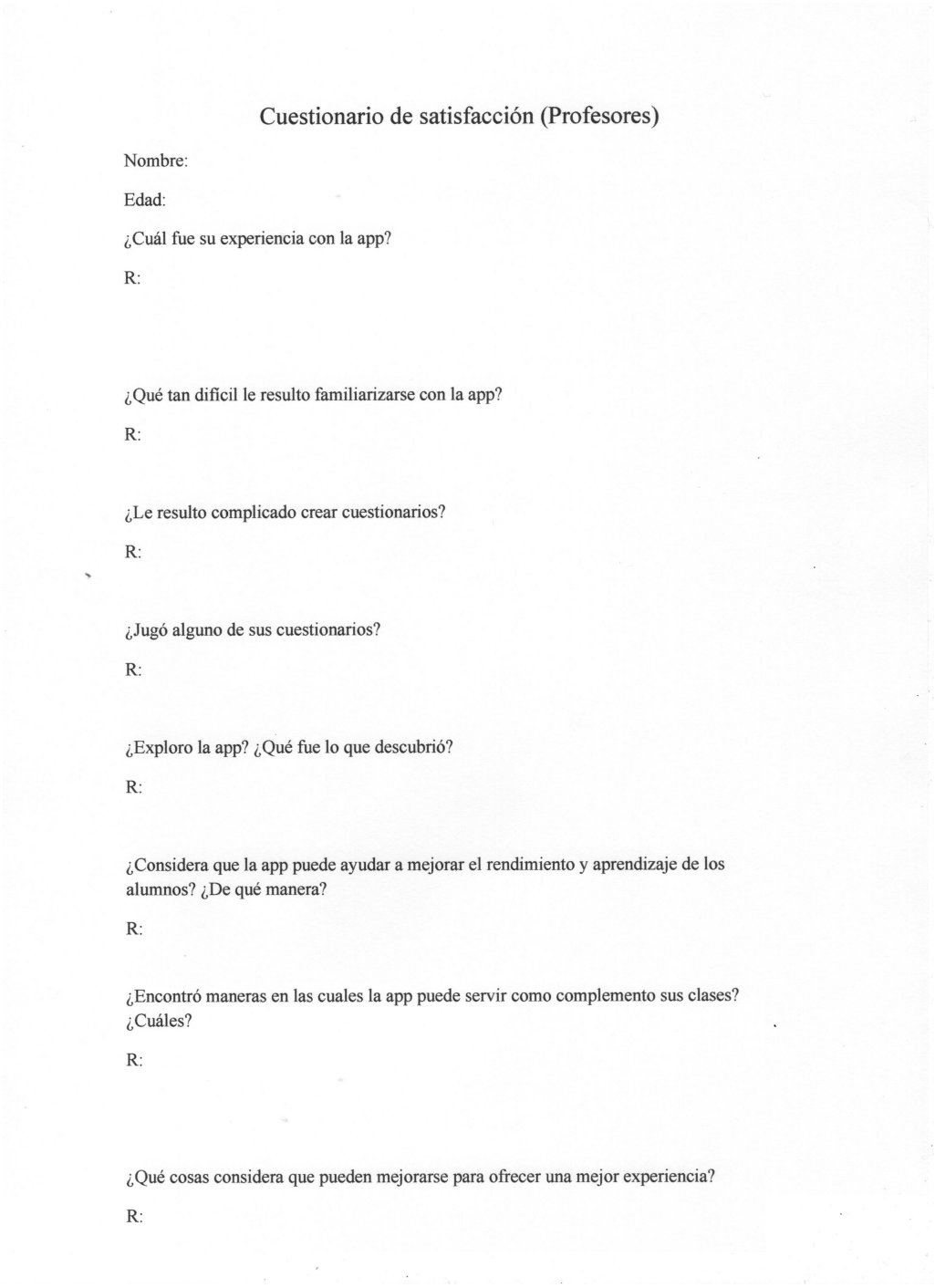
7. Al pronombre se le conoce por la siguiente finalidad:
a) Acompaña al nombre b) Modifica al nombre
c) Sustituye al nombre que se mencionó con anterioridad

8. La preposición es:
a) Propia b) Un nexo que relaciona términos c) Determina el tipo de oración

9. Definición de complemento indirecto:
a) Destinatario o beneficiario de la acción b) En quien recae directamente la acción c) Complementa al sujeto

10. Los sustantivos son:
a) Aquellos que nombran a personas dentro del sujeto
b) Los que se encargan de nombrar ideas, sentimientos, objetos o personas
c) Los que sustituyen al nombre

Figura 5.8: Examen de español.



Cuestionario de satisfacción (Profesores)

Nombre:

Edad:

¿Cuál fue su experiencia con la app?

R:

¿Qué tan difícil le resulto familiarizarse con la app?

R:

¿Le resulto complicado crear cuestionarios?

R:

¿Jugó alguno de sus cuestionarios?

R:

¿Exploro la app? ¿Qué fue lo que descubrió?

R:

¿Considera que la app puede ayudar a mejorar el rendimiento y aprendizaje de los alumnos? ¿De qué manera?

R:

¿Encontró maneras en las cuales la app puede servir como complemento sus clases? ¿Cuáles?

R:

¿Qué cosas considera que pueden mejorarse para ofrecer una mejor experiencia?

R:

Figura 5.9: Encuesta de satisfacción para profesores.

Los resultados arrojados por la encuesta fueron favorables ya que los usuarios profesores expresaron haber sentido muy amigable la interfaz, sencilla de utilizar y calificando la idea como algo novedoso y creativo, inclusive pidieron poder utilizar la aplicación dentro de sus clases y el permiso para distribuirla con futuros alumnos ya que al parecer habían pensado al manera de integrarlo dentro de sus clases ya fuera como método de repaso, para dar puntos extras o aplicar exámenes.

5.2 Analizando al Usuario Alumno

Para el usuario alumno fue más complejo, se seleccionaron alumnos de dos grupos donde los docentes daban clases bajo ciertas características las cuales fueron: Alumnos que aprendieran rápido, alumnos que tuvieran cierta dificultad para aprender pero estudiaran mucho, alumnos que a pesar de hacer un gran esfuerzo se les dificultara aprender y alumnos que tuvieran problemas de aprendizaje y no hiciera un esfuerzo extra escolar para aprender, teniendo un total de 9 alumnos, con la finalidad de estudiar el impacto de la app en difierentes tipos de alumnos, datos que serían obtenidos al evaluar por separado a cada uno antes y después del uso de la app.

Se reunieron a los alumnos dentro de un aula de clases donde antes de empezar con las explicaciones se les aplicaron los cuestionarios/exámenes con la intención de medir los sus conocimientos de los temas, a cada uno se les repartió un conjunto de 3 (Uno de cada materia) con la intención de medir sus conocimientos antes de usar la aplicación. Las estadísticas de las calificaciones obtenidas se muestran en las figuras 5.12, 5.13 y 5.14.

Después se les aplico un cuestionario donde a los alumnos se les pregunto sobre el uso del celular, la forma en que lo usan, sus experiencias previas con aplicaciones para sus estudios (si es que llegaban a utilizar) y si alguno de sus profesores las había utilizado así como su satisfacción con el uso de estos métodos, vease la figura 5.15.

La edad de los betatester se encontraba entre los 18 y 20 años, el tiempo de uso de su smathphone oscilaba entre las 8 y 12 horas al día. Entre las aplicaciones más utilizadas se encontraban Instagram, Facebook, Whatsapp y Youtube dando a entender que el uso más común para ellos eran las redes sociales.

La encuesta también arrojo que dentro de las aplicaciones que usaban para su aprendizaje se encontraban Duolingo que es una aplicación para aprender idiomas a través de diversas actividades y Youtube (Viendo videos sobre los temas de sus clases que no habían quedado claros) aunque la mitad de los encuestados dijeron no usar ninguna aplicación con fines educativos. Otros de los datos que se obtuvieron al momento de recopilar los datos fue que dentro de las aplicaciones más usadas por los profesores para mejorar la experiencia

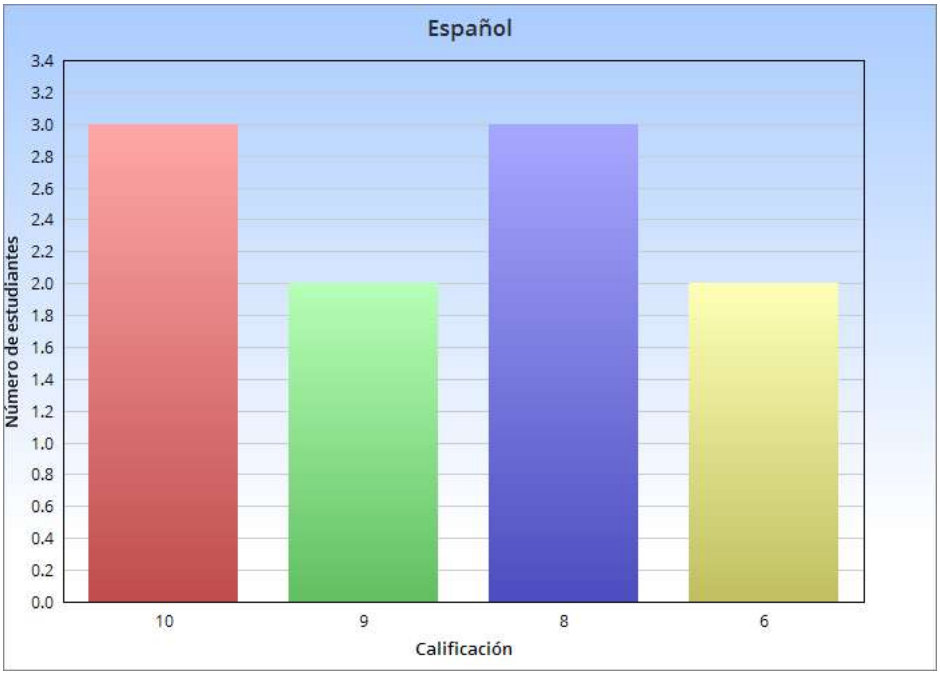


Figura 5.10: Gráfica de las calificaciones obtenidas en español antes de usar la app.

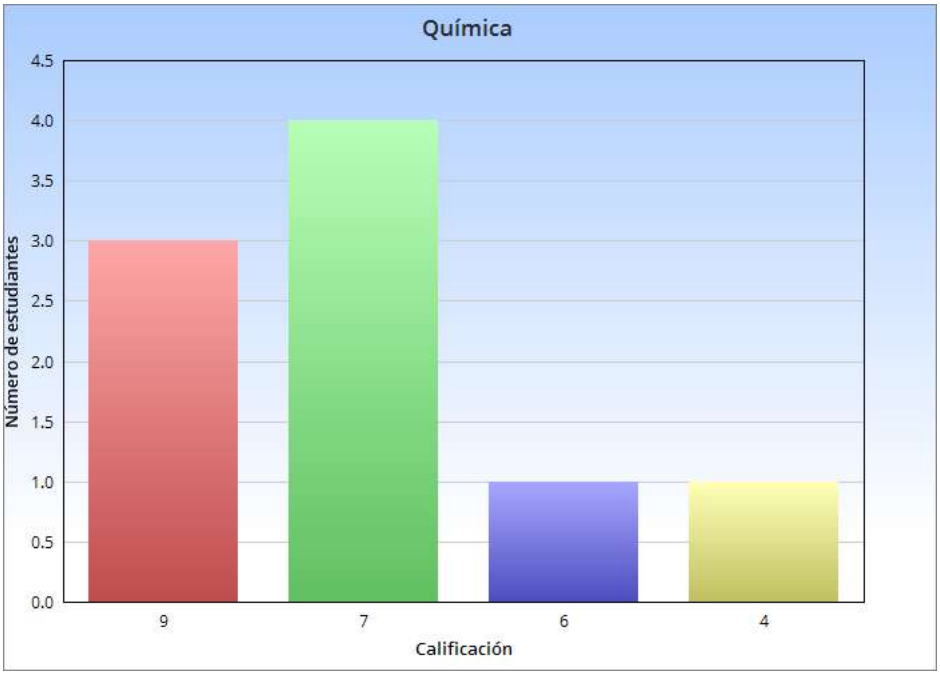


Figura 5.11: Gráfica de las calificaciones obtenidas en química antes de usar la app.

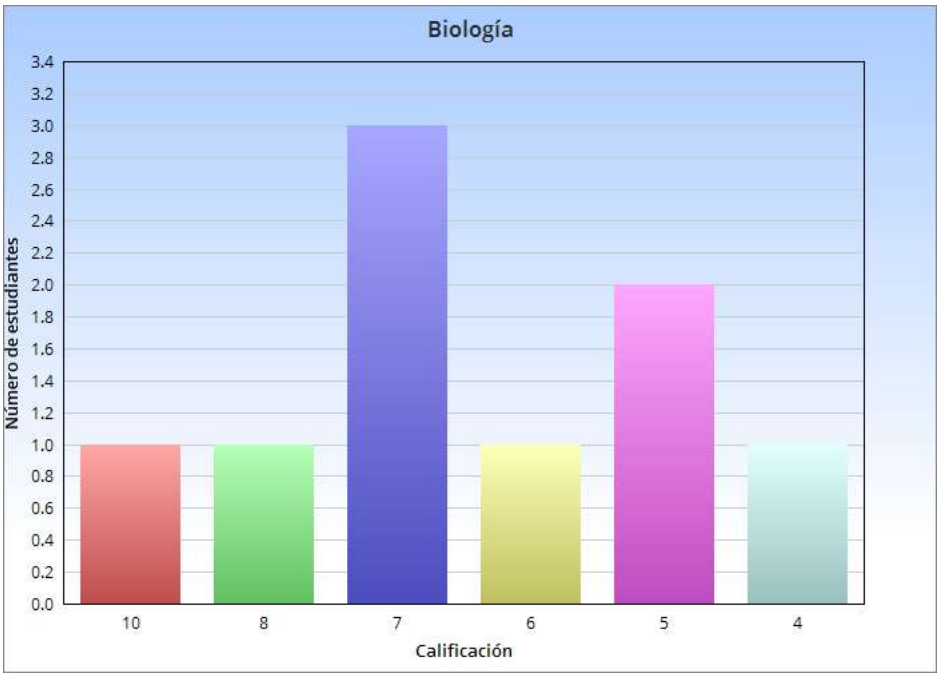


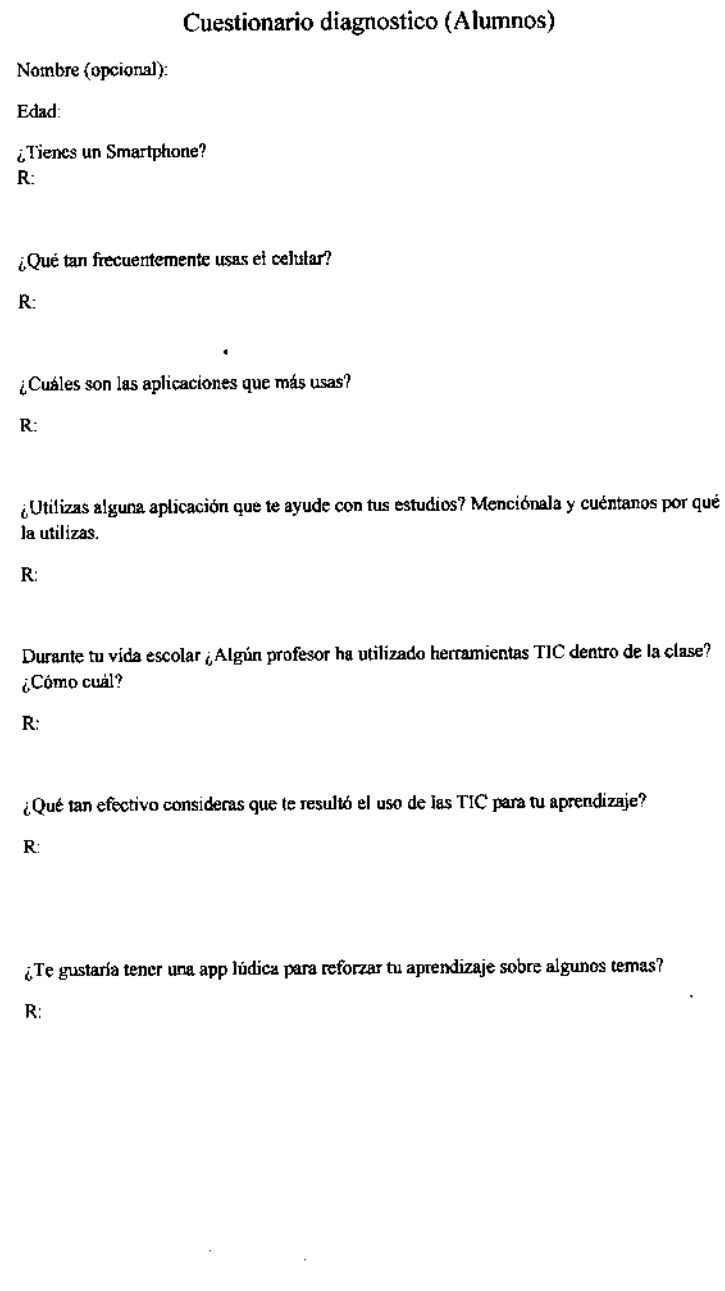
Figura 5.12: Gráfica de las calificaciones obtenidas en biología antes de usar la app.

en sus clases destacaban las diversas herramientas que ofrece Microsoft Office, Kahoo una plataforma para crear y jugar cuestionarios en línea seleccionando una respuesta de las ofrecidas al momento de la aparición de una pregunta y Geogebra, un software dedicado al entendimiento de las matemáticas y sus diversas ramas a través de la interacción del usuario con las herramientas que ofrece.

Todos expresaron que el uso de TIC y aplicaciones dentro de una clase facilita el entendimiento y comprensión de los temas y así mismo mostraron interés en tener una app que les ayudara con su aprendizaje.

Para dar un ejemplo sobre la intención de la aplicación se les hizo un ejemplo con tarjetas de papel donde se escribió el nombre y cumpleaños de cada uno para después hacerlos buscar cada pareja como en la aplicación, una vez que los usuarios alumnos entendieron el concepto se les instaló la aplicación en sus celulares (cuidando que todos tuvieran dispositivo Android y los que no, se les proporciono uno) y se les pidió que se registraran.

Se les explicó de manera muy breve y simple (sin entrar en detalles) la interfaz de la aplicación y se dejó que exploraran la aplicación, al principio tuvieron muchas dudas las cuales se les intento responder de manera que los guiara un poco pero los invitara a descubrir por ellos mismo lo que resolvía su duda. Se les proporcionó el “nickname” de los profesores que habían hecho los cuestionarios/exámenes y de acuerdo al tema elegido era el juego que le correspondía.



Cuestionario diagnostico (Alumnos)

Nombre (opcional):

Edad:

¿Tienes un Smartphone?
R:

¿Qué tan frecuentemente usas el celular?

R:

¿Cuáles son las aplicaciones que más usas?

R:

¿Utilizas alguna aplicación que te ayude con tus estudios? Menciónala y cuéntanos por qué la utilizas.

R:

Durante tu vida escolar ¿Algún profesor ha utilizado herramientas TIC dentro de la clase? ¿Cómo cuál?

R:

¿Qué tan efectivo consideras que te resultó el uso de las TIC para tu aprendizaje?

R:

¿Te gustaría tener una app lúdica para reforzar tu aprendizaje sobre algunos temas?

R:

Figura 5.13: Encuesta de diagnóstico para alumnos.

Al pasar del tiempo los usuarios fueron descubriendo las diversas cosas que podían hacer y lo compartían entre ellos, así como su entusiasmo por el juego fue creciendo haciendo notoria su emoción por la aplicación. Al cabo de unos minutos estar jugando descubrieron la función multijugador y después de preguntar y ser guiados hacía la respuesta su emoción aumento al grado de comenzar a competir (lo cual era la intención dentro de un juego), conforme fueron explorando se fueron encontrando pequeños errores lo cuales se apuntaron para corregir una vez terminada la prueba.

Después de un lapso de hora y media se les pidió que dejaran su celular por un momento y se les volvió a aplicar los cuestionarios/exámenes para medir lo aprendido durante el uso de la aplicación obteniendo los resultados mostrados en las figuras 5.16, 5.17 y 5.18.

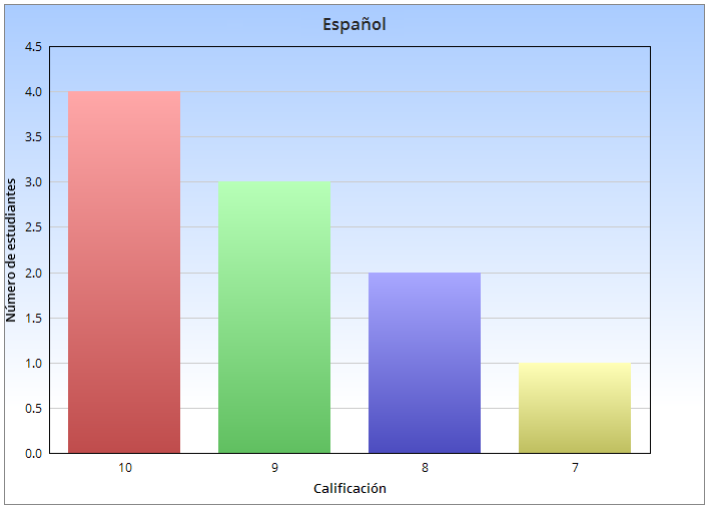


Figura 5.14: Gráfica de las calificaciones obtenidas en español después de usar la app.

Para ese día había sido mucho trabajo por lo que se les dejo descansar y al cabo de unos días se les envió por correo el cuestionario de satisfacción (figuras 5.19 y 5.20) al momento de interactuar con la aplicación para conocer las cosas en las que se había que mejorar y que tanto éxito se podría conseguir.

En general los usuarios (tanto como docentes y alumnos) se mostraron entusiasmados en el uso de la aplicación, resaltando el comentario de que era una idea novedosa, tuvieron un poco de complicaciones al inicio para familiarizarse con la aplicación pero después de un tiempo y de explorar un poco dijeron haberla dominado (no en su totalidad pero si en gran parte), descubrieron casi la mayoría de las funciones, inclusive algunos se aventuraron a crear sus propios juego al momento de la experimentación, dentro de las cosas a mejorar se destacó poder usar imágenes, un sistema multijugador para más usuarios, que las imágenes se adaptaran a la pantalla de manera automática, que el registro se pudiera hacer a través de sus cuentas de google y al cerrar la aplicación no tuvieran que volver a hacer login.

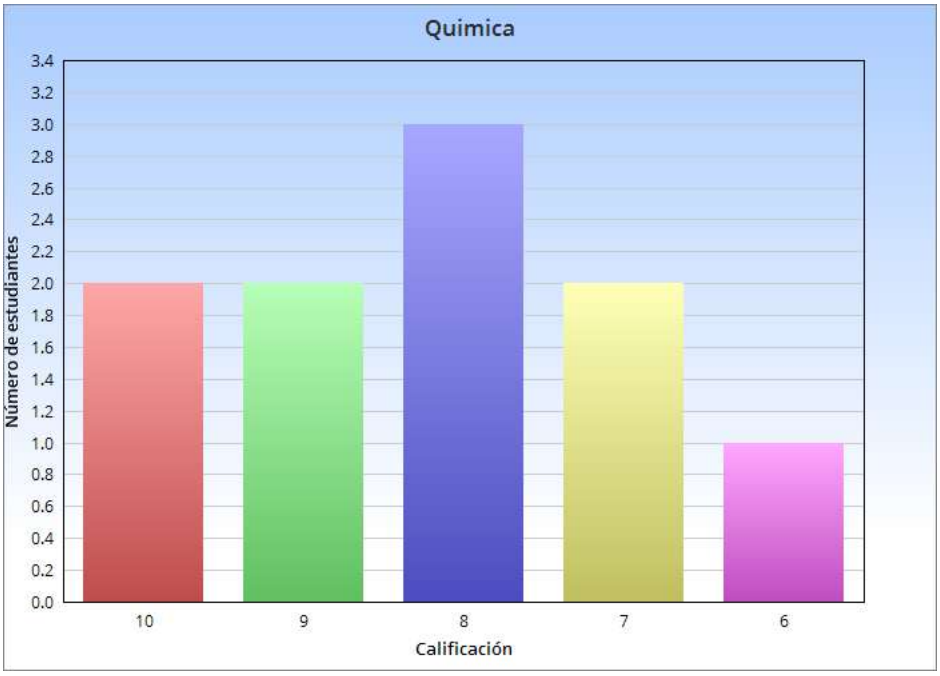


Figura 5.15: Gráfica de las calificaciones obtenidas en química después de usar la app.

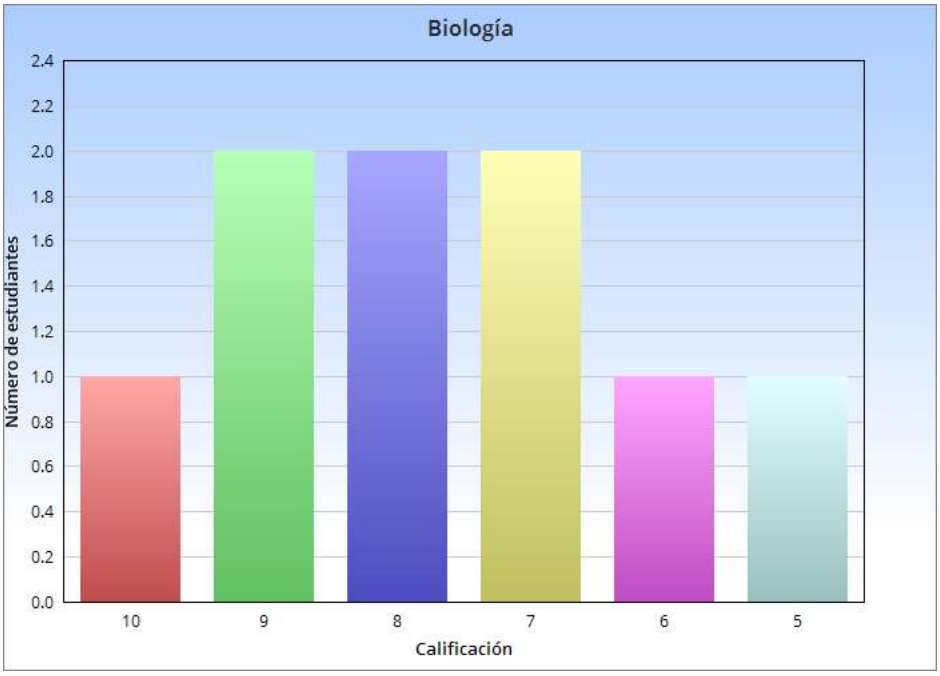


Figura 5.16: Gráfica de las calificaciones obtenidas en biología después de usar la app.

Cuestionario de satisfacción (Alumnos)

Nombre (Opcional):

Edad:

¿Cuál fue su experiencia con la app?

R:

¿Qué tan difícil le resulto familiarizarse con la app?

R:

¿Cómo le resulto la forma de juego?

R:

¿Aprendió de los temas al momento de jugar?

R:

¿Exploro la aplicación? ¿Qué descubrió?

R:

¿Cómo le resulto la experiencia online?

R:

¿Considera que la aplicación podría ayudarte con tus estudios?

R:

Figura 5.17: Encuesta de satisfacción para alumnos

¿Cuáles fueron las cosas que le gustaron de la app?

R:

¿Cuáles fueron las cosas que le desagradaron de la aplicación?

R:

¿Qué cosas considera que se pueden mejorar para ofrecer una mejor experiencia?

R:

Figura 5.18: Encuesta de satisfacción para alumnos.

Todos expresaron haber aprendido en diferentes medidas y que es una herramienta que los podría ayudar con sus cursos, inclusive los profesores dijeron haber encontrado múltiples maneras de usar la app dentro de sus clases, todos disfrutaron la experiencia, Se procuró crear una aplicación sencilla de manejar y al parecer se logró algo muy cercano al objetivo.

Los resultados arrojados por la app fueron muy satisfactorios ya que el propósito era crear una app donde los estudiantes aprendieran, se divirtieran y que al profesor le resultara útil, dado la experiencia no divergimos mucho de ello.

Capítulo 6

Conclusiones

Actualmente la tecnología se ha convertido en algo de nuestro día a día, su desarrollo ha dado pasos cada vez más grandes y hemos adaptado nuestra vida a esos cambios, con su desarrollo se intenta cubrir nuestras necesidades construyendo herramientas que faciliten el cumplimiento de tareas y a partir de la creación de los smarthphone se ha buscado que sea algo con la mayor cantidad de herramientas a nuestra disposición en algo tan pequeño. Las aplicaciones se han ido evolucionado permitiéndonos hacer cosas de manera muy simple desde comunicarnos, guardar recuerdos, leer libros, hacer compras, realizar movimientos bancarios, ver películas o jugar, y se ha convertido en una herramienta indispensable contando la gran mayoría de las personas con uno.

La situación actual por la que está pasando el mundo (COVID-19) nos ha hecho ver que en momentos de crisis la tecnología puede ser una gran aliada, ya que las autoridades decretaron que debíamos permanecer en nuestro hogar entonces con los smarthphone la gente se comunica con sus familiares y amigos, hacen compras en el súper, se mantienen informados y se entretienen todo esto sin la necesidad de salir de sus casas, sin poner en riesgo su salud y acatar las recomendaciones impuestas por el gobierno.

También nos mostró que en cuestión de educación no estamos del todo preparados para algo así, a pesar de los grandes esfuerzos del sector educativo por intentar cubrir esta necesidad con la tecnología los resultados han sido muy poco favorables resaltando el déficit que se tiene en esta área, siendo un sector del cual no se ha sacado todo el provecho que se puede y a pesar de contar con un fuerte aliado como lo es la gamificación no se han enfocado los esfuerzos suficientes para su desarrollo.

La gamificación es una técnica educativa a la cual se le puede sacar mucho provecho ya que bajo su concepto centra sus esfuerzos en pasar la mecánica de los juegos al ambiente educativo-profesional siendo su finalidad conseguir mejores resultados para mejorar habilidades

o adquirir conocimiento, con todo un mundo por explorar y aprovechar nos abrió las puertas a la realización de éste proyecto.

Entonces tomando en cuenta que muchos jóvenes y niños tienen acceso a un smarthphone que puede ser una herramienta muy útil, que la tecnología toma un papel muy importante en momentos difíciles y nos facilita muchas tareas, que las aplicaciones educativas cuentan con buenas bases pero poco desarrollo existe la necesidad de crear aplicaciones que puedan apoyar al aprendizaje de una manera divertida, sencilla y cómoda, actualmente existen muchas pero suelen tener dos inconvenientes, o están enfocadas a niños o están enfocadas a un tema en específico haciendo que no existan que cuenten con una diversidad tanto para edades como para temas.

Este proyecto se centró en desarrollar una aplicación que cubriera todas las necesidades anteriormente mencionadas, que fuese fácil y divertido de utilizar tomando como base las técnicas mecánicas y dinámicas que ofrece la gamificación a través del juego del memorama que tuvo como objetivo implementar su mecánica de juego al aprendizaje de conceptos de cualquier área dentro de una aplicación móvil llamada "Memorato". La idea era que una tarjeta contuviera una parte de la información y la otra su complemento, es decir que cualquier cosa que se pudiera representar en forma de pregunta y respuesta podía colocarse en las cartas del juego.

Memorato se comenzó a desarrollar del centro hacía afuera ya que la primera parte creada fue el juego y a partir de ahí se fueron agregaron las de más herramientas que dieran al usuario una experiencia agradable y cómoda, como agregar sus propios juegos, buscar juegos creados por otros usuarios o ver sus puntuaciones obtenidas. Cuando los objetivos fueron cumplidos y la aplicación se encontraba en una fase Beta se decidió hacerla dar un paso muy grande que fue implementar un sistema multijugador online que permitiera a un usuario jugar contra alguno de sus amigos y poner a prueba sus conocimientos compitiendo con alguien más, ya que una de las partes más divertidas de un juego es competitividad, de esta manera se buscó despertar el interés del usuario y motivarlo a jugar, así al mismo tiempo que se divertía podía ir aprendiendo.

El concepto del memorama es algo que desde niños tenemos al ser un juego muy simple de entender y con el que la gran mayoría de las personas se encuentran identificados, eso permitió que al momento de la experimentación los sujetos de prueba rápidamente se familiarizaran con él arrojando buenos resultados a pesar de que la mayoría nunca antes habían interactuado con una aplicación similar, los usuarios expresaron sentirse cómodos al momento de jugar y que era algo de su agrado además de que las pruebas de conocimiento que se les hicieron antes y después de la interacción con la aplicación mostraron que hubo un aprendizaje lo cual nos dió a entender que la aplicación se encontraba por buen camino.

Aún quedan muchas cosas por hacer dentro de la aplicación dentro de ellas se encuentran

desarrollarla para IOS ya que por el momento se encuentra solo disponible para android, crear un modo multijugador donde se pueda jugar contra varios jugadores al mismo tiempo y no solo en un 1 vs 1, poder agregar imágenes tanto como pregunta o como respuesta, etc. Que se espera para versiones posteriores desarrollar entre otras ideas que por cuestiones de tiempo no se tuvo la oportunidad de explorar pues la idea da para muchas formas de implementarse usando la mecánica de otros juegos, ya que mostró ser una buena herramienta de aprendizaje que puede tener mucho alcance y ayudar a los usuarios lo cual es y será siempre el objetivo.

Bibliografía

[Exc20] Zepeda, H. (2019). *En México, 70% de estudiantes ha reprobado matemáticas.* 25 de mayo del 2020, de Excelsior Recuperado de: https:www.excelsior.com.mxnacionalenmexico70deestudiantesha-reprobadomatematicas/1248182?fbclid=IwAR0Hsivb4tcSTy4aHIyzkTi-tXrdyYK8PnZtQc1o5iWygWzkM7WHDGft5R7g

[Exp] Aguilar, M. (2018). *En México el número de mexicanos con acceso a internet subió a 71.3 millones.* 25 de mayo del 2020, de Expansión Recuperado de: https://expansion.mx/tecnologia/2018/02/20/en-mexico-hay-713-millones-de-mexicanos-con-acceso-a-internet?fbclid=IwAR3xBTQbdzBzn2GR1IMu-aIa_BhSqod4T8LA2kFYlrp1UbpoLusIclWJhIk

[edua] educaplay. (2017). *Cómo crear Juegos de Relacionar Mosaico.* 6 de junio del 2020, de educaplay Recuperado de: https://es.educaplay.com/tipos-de-actividades/juegos-de-relacionar-mosaico.html?fbclid=IwAR0LnF0MWszxmN0lECc63GIZ9ANSZ43JRXK-xzy5ad50GC2fw0pv1Hv2EEU

[edub] educaplay. (2017). *Recursos educativos.* 6 de junio del 2020, de educaplay Recuperado de: https://es.educaplay.com/recursos-educativos/?q=&type=memory-games&fbclid=IwAR0EbOc6lD9caqlm5MYjj5lTHagw_VyySjcT2uEA-MjI4Qdvh3kjycDVFYM

[edu20] eduplay. (2017). *Juego de relacionar columnas.* 6 de junio del 2020, de educaplay Recuperado de: https://es.educaplay.com/tipos-de-actividades/juegos-de-relacionar-columnas.html?fbclid=IwAR1s5nUxE9DielWT-IrI9elUV0EeckuIvztyBxPZrpiXGLQw0NgduDjakAk

[Cer20b] Cerebriti. (2016). *Juegos de relacionar.* 7 de junio 2020, de Cerebriti Recuperado de: https://www.cerebriti.com/juegos-de-relacionar/tag/mas-jugados/?fbclid=IwAR07Emplp0pgCvWFo-fr06OlKsLBPX0fTKOkPVf1gxbViHrcwwnIUOcrK-4

[Cer20a] Cerebriti. (2016). *Juegos de lengua*. 7 de junio del 2020, de Cerebrito Recuperado de: https://www.cerebriti.com/juegos-de-lengua/relaciona-palabra-imagen?fbclid=IwAR3MXfQ0hfR-PebFcdbdtK9db0Km-O3819N625vTRz1M27-i2pJfehoeKV4

[yMM20] Montero, Y. (2015). *Experiencia de Usuario: Principios y Métodos*. España: Addison-Wesley.

[Val20] Silberschatz, A., Korth, H., & Sudarshan, S. (2014). *Fundamentos de bases de datos (Vol. 6)*. España: Mcgraw Hill.

[Rui20] Carbonell, V., & Tomas. J. (2018). *Firebase: Trabajar en la Nube*. México: Marcombo.

[Wik20] Muñoz, J., & Damiano, R. (2012). *Bases de datos*. España: Alfaomega. Base de datos, jun 2020.

[Man20] Krug, S. (2000). *No me hagas pensar*. Estados Unidos: New Riders.

[Gar20] Garrett, J. (2011). *The elements of User Experience*. Estados Unidos: New Riders.

[Gai20] Hierro, E. (2013). *Gamificación*. Colombia: Empresa Activa.

[Mol20] Richards, J. (2010). *GAMIFICACIÓN*. España: Alfaomega.

[Goo20] Google. (2019). *Add firebase to your android project*. 12 de junio del 2020, de Google Recuperado de: https:firebase.google.comdocsandroidsetup?authuser=0&fbclid=IwAR2QohFD-PEXguNr59l7xKdf9VOCcIVFIOsTLtmsfsSJdjO12KhG7lYqDEU

[pp] Dibujos para pintar. (2018). *Actividades escolares*. 25 de mayo del 2020, de Dibujos para pintar Recuperado de: http:www.dibujosparapintar.comactividades_escolares.html

[Wik] Hebuterne, S. (2018). *Desarrolle una aplicación android. Programación en Java con android studio*. España: Eni.

[Sou20] S, A. K. (2018). *Mastering Firebase for Android Development: Build real-time, scalable, and cloud-enabled Android apps with Firebase*. Birmingham: Packt.

[Est20] Luján Castillo, J. D. (2017). *Android Studio - Aprende a desarrollar aplicaciones*. España: Alfaomega, RC Libros.

[Gar08] Salazar, J. (2008). *Fundamentos del Aprendizaje*. México: Editorial Trillas.

[Edu17] educaplay. (2017). *eduplay*. 25 de mayo del 2020, de educaplay Recuperado de: https://www.educaplay.com

[Cer18] Cerebriti. (2018). *Cerebriti*. 25 de mayo del 2020, de Cerebriti Recuperado de: https://www.cerebriti.com

[Dib18] Dibujos para pintar. (2018) *Dibujos para pintar*. 25 de mayo el 2020, de Dibujos para pintar Recuperado de: https:www.dibujosparapintar.com