Universitat de les Illes Balears

Escuela Politécnica Superior

Memoria del Trabajo de Final de Grado

Juego serio para Entrenar la Identificación y Discriminación Auditiva en Niños con Discapacidad Auditiva

Yolanda Alemany Ruiz

Grado de Ingeniería Informática

Año académico 2017-18

DNI del alumno: 41542901-W

Trabajo tutelado por Cristina Suemay Manresa Yee

Departamento de Ciencias de la Computación e Inteligencia Artificial

Se autoriza a la Universidad a incluir este trabajo en el Repositorio Institucional para su consulta de acceso libre y difusión en línea, con finalidades exclusivamente académicas y de investigación	Autor		Tutor	
	Sí	No	Sí	No
	X		X	

Palabras clave del trabajo:
Juego serio, Discapacidad auditiva, Niños

A mi familia, que me han servido de apoyo incondicional durante estos años de esfuerzo y sacrificio, y que viven cada uno de mis logros con la misma ilusión con la que los vivo yo.

A mis compañeros, ya que sin ellos el camino hubiera sido mucho más aburrido.

A mi tutora, Cristina Manresa, por darme el apoyo, la energía y la motivación para sacar el trabajo adelante y por permitirme formar parte de este proyecto tan humano y bonito.

Índice

Índice 5
Tabla de acrónimos 7
Listado de tablas 8
Listado de Figuras ..... **Error! Bookmark not defined.**
Resumen 11
Motivación personal 12
1. Introducción 13
1.1. Marco del proyecto 13
1.2. Objetivo del Trabajo Final de Grado (TFG) 14
1.3. Estructura del documento TFG 14
2. Conceptos relacionados con el trabajo desarrollado 15
2.1. Discapacidad auditiva. 15
2.2. Dispositivos de ayuda. Audífonos e implantes cocleares 15
2.3. Logoterapia y entrenamiento auditivo. 16
2.4. Gamificación. Juego serio 17
2.5. Trabajos relacionados 18
3. El juego: +memory. 19
4. Metodología 21
5. Análisis. 23
6. Marco tecnológico 26
7. Arquitectura y estructura del software 27
7.1 Arquitectura (MVC) 27
7.2 Interfaces 27
7.3 Clases del programa 28
8. Datos de la aplicación 31
8.1. Modelo de la base de datos 32
8.2. Conexión con la base de datos 34
8.3. Consultar la base de datos 35
9. Interfaz de usuario 38

9.1. Diagrama de flujo de paneles 38
9.2. Cambio de panel 41
9.3. Tipos de paneles 44
9.3.1. Paneles de la zona administrador 45
9.3.2. Paneles de la zona usuario jugador 50
9.4. Adaptación a la pantalla. 53
9.5. Elementos gráficos adicionales 55
9.6. Adaptación de las imágenes 55
10. Control de la partida 57
10.1. Preparación de la partida 57
10.2. Ejecución de la partida. 58
11. Conclusiones y trabajo futuro 60
11.1. Conclusiones personales. 60
Anexos 61
A.1. Fuentes. 61
A.2. Consultas de creación de las tablas de la base de datos. 61
Referencias 64

Tabla de acrónimos

OMS	Organización Mundial de la Salut
UIB	Universidad de las Islas Baleares
ASPAS	Asociación de Padres de Personas con Discapacidad Auditiva
OCDS	Oficina de Cooperación al Desarrollo y Solidaridad
CUD	Cooperación Universitaria al Desarrollo
TFG	Trabajo Final de Grado
MVC	Modelo Vista Controlador
CRUD	Create, Read, Update, Delete

Listado de cuadros

5.-1	Requisitos funcionales de usuario
5.-2	Requisitos funcionales del sistema
5.-3	Requisitos no funcionales del sistema
7.2.	Información sobre las interfaces en la aplicación
9.3.1.	Información sobre la gestión de errores
9.5.	Componentes gráficos creados

Listado de figuras

2.3.	Fases del entrenamiento auditivo
4.	Actividades del Diseño Centrado en el Usuario DCU
7.3.-1	Organización de las clases del modelo
7.3.-2	Organización de las clases de la vista
7.3.-3	Organización de las clases del controlador
8.-1	Atributos de la clase principal del Modelo
8.-2	Atributos de la clase Juego
8.1.	Modelo de la Base de Datos
8.2.	Código para abrir y cerrar la conexión con la base de datos
8.3.-1	Consulta *delete()* en la tabla Tarjeta
8.3.-2	Consulta *read()* en la tabla Usuario
9.1.-1	Diagrama de flujo inicial
9.1.-2	Diagrama de flujo Zona administrador (gestión)
9.1.-3	Diagrama de flujo Zona Jugador / Juego
9.2.-1	Ejemplo de barra superior
9.2.-2	Asignación de nuevos valores a los punteros panelAnterior y panelActual, en el regreso al menú de usuario después de finalizar una partida (modo Juego)
9.3.-1	Panel inicial de la aplicación
9.3.-2	Panel para entrar al menú de usuario y a la zona administrador
9.3.-3	Código para crear el panel que permite seleccionar usuarios
9.3.1.-1	Panel inicial de la zona administrador
9.3.1.-2	Panel de visualización (Usuarios)

9.3.1.-3	Panel de visualización (Configuraciones)
9.3.1.-4	Panel de visualización (Tarjetas).
9.3.1.-5	Ventana con panel creación/edición (Usuarios)
9.3.1.-6	Ventana con panel creación/edición (Configuraciones)
9.3.1.-7	Ventana con panel creación/edición (Tarjetas)
9.3.1.-8	Panel de edición de tarjeta con mensaje de error
9.3.1.-9	Panel para añadir nuevo fonema y posición del fonema
9.3.2.-1	Panel que contiene el menú de usuario
9.3.2.-2	Panel para seleccionar la dificultad de la partida (modo Juego)
9.3.2.-3	Panel para seleccionar la configuración (modo Entrenamiento)
9.3.2.-4	Panel al inicio de la partida con las tarjetas del juego boca abajo
9.3.2.-5	Panel durante la partida con tres pares acertados
9.3.2.-6	Ventana que se muestra al finalizar una partida
9.3.2.-7	Ejemplo de adaptación a diferentes pantallas con resolución resolución 800x600 (imágenes de la izquierda) y 1920x1080 (de la derecha)
9.6.	Código de redimensión de imagen
10.1.-1	Código de creación de configuración en la preparación de la partida (modo Juego)
10.1-2	Código de preparación de las tarjetas de juego
10.2.	Código para comprobar la asociación de tarjetas de juego

Resumen

Los niños con discapacidad auditiva tienen que afrontar muchos obstáculos derivados de dicha discapacidad como, por ejemplo, mejora de habilidades comunicativas, habilidades psicomotoras, etc. Por otra parte, al utilizar dispositivos que les ayudan a mejorar su capacidad auditiva, todavía han de aprender a oír. En este Trabajo de Final de Grado (TFG) se presenta el proceso de desarrollo de un videojuego serio, +memory, con el cual los niños con discapacidad auditiva podrán mejorar algunas de las habilidades que les permiten oír mejor, estas son la identificación y discriminación auditiva.

Este juego forma parte del proyecto cooperativo *Diseño de experiencias interactivas dirigidas al bienestar de personas con necesidades especiales.* Cuyo objetivo general es diseñar experiencias interactivas en salud y educación (teniendo en cuenta la componente emocional y la social) para poder mejorar el bienestar de estar personas.

El diseño y desarrollo de la aplicación se ha llevado a cabo gracias al uso de la metodología: Diseño Centrado en el Usuario. El cual trata de obtener sistemas más usables aplicando conocimientos de ergonomía y usabilidad

Motivación personal

Para mí el ámbito social es el más importante en la vida, ya que independientemente de la profesión o el nivel de estudios de uno, el objetivo es vivir en una comunidad a gusto, con el menor número de complicaciones (aunque a veces vengan de nosotros mismos), y en una comunidad en la que todo el mundo aporte lo que sabe hacer.

¿Qué pasa cuando la vida no te ha dado todo lo que a otras les da? ¿Qué pasa cuando naces con un problema de salud que igual no te impide llevar una vida normal, pero es un pequeño obstáculo? Es en ese momento de la vida de algunas personas en el que me gustaría aportar mi granito de arena ya que con la ayuda de los conocimientos que he ido adquiriendo a lo largo de los años tanto en la carrera como en el mundo laboral, me siento capaz de poder ayudar de alguna manera a la sociedad. Algunos profesionales, como mi padre, son fontaneros y te arreglan un destrozo que has hecho en el desagüe de lavabo; otros te hacen y te ponen la hamburguesa que vas a disfrutar con la compañía de tus amigos. Y otros, como yo, podemos desarrollar una aplicación que permita entrenar las habilidades de una persona que no ha nacido con la mejor salud para mejorar su calidad de vida.

En resumen, me gusta aportar mis conocimientos a una labor social con la que pueda ayudar a otras personas. Y como adelanto, la labor social en la que me centro es ayudar a niños con discapacidad auditiva a desarrollar habilidades de la identificación y discriminación auditivas.

1. Introducción

El bienestar de una persona está asociado a múltiples aspectos de su vida, entre ellos, el intelectual, ambiental, emocional, social, ocupacional, espiritual y físico [1]. En el caso de las personas con discapacidad, las limitaciones físicas, cognitivas o perceptivas dificultan el relacionarse con el entorno y con las personas que la rodean, y como consecuencia, puede afectar el alcanzar la plenitud en todas las dimensiones del bienestar. En este proyecto se presenta un sistema interactivo para la mejora del bienestar de las personas con discapacidad auditiva. Los resultados obtenidos forman parte de un proyecto de cooperación financiado por la Universitat de les Illes Balears (UIB) en el que participo como miembro del equipo.

1.1. Marco del proyecto

El proyecto de cooperación *Diseño de experiencias interactivas dirigidas al bienestar de personas con necesidades especiales* (OCDS-CUD2016/13) tiene una duración de dos años (2017 hasta el 30 de septiembre de 2018) y una financiación de 10.900 euros. El proyecto se puede llevar a cabo gracias a la financiación obtenida en la XIII Convocatoria de ayudas para proyectos de Cooperación Universitaria al Desarrollo (CUD) (2016) de la Oficina de Cooperación al Desarrollo y Solidaridad (OCDS). La OCDS es una estructura solidaria que se encarga de gestionar los programas CUD y de voluntariado, además de otras actividades dentro de ámbitos relacionados con la solidaridad. [2, 3]

Este proyecto nace de una cooperación entre la Universidad de Cauca (Colombia), la Universidad Autónoma de Occidente (Colombia), la Universidad de San Buenaventura de Cali (Colombia) y la UIB. La responsable es la Dra. Cristina Manresa Yee que es profesora del Departamento de Ciencias Matemáticas e Informática de la UIB, y los participantes del proyecto son tanto profesores como estudiantes.

El objetivo general del proyecto es diseñar experiencias interactivas en salud y educación (teniendo en cuenta la componente emocional y la social). Gracias a esto, se puede mejorar el bienestar de estas personas. [3]

El proyecto busca crear sistemas interactivos, usables y accesibles, que motiven a las personas a interactuar con el sistema. La idea es diseñar experiencias con las cuales el usuario disfrute y a la vez le ayuden a trabajar aspectos de salud o educativos. ¿Cómo se puede conseguir? En base al desarrollo de aplicaciones gamificadas, colaborativas o juegos serios, cuya finalidad es entretener, motivar y conseguir la participación activa del usuario en una tarea que tiene un objetivo más allá del simple entretenimiento (p.e. mejorar la salud) y busca que impacte más allá del tiempo en el que utilice el sistema.

Se ha introducido el objetivo general del proyecto, pero se definen cuatro objetivos específicos:

1. Formación bidireccional e investigación
2. Diseño de experiencias interactivas en educación
3. Diseño de experiencias interactivas en salud
4. Diseño de experiencias interactivas emocionales

1.2. Objetivo del Trabajo Final de Grado (TFG)

El objetivo principal del TFG es desarrollar un videojuego serio dirigido a niños con discapacidad auditiva que les ayude a mejorar las habilidades de identificación y discriminación de sonidos, este juego recibe el nombre de ***+memory*** y se enmarca en el objetivo específico 2 comentado en el subcapítulo anterior, diseño de experiencias interactivas en educación.

El juego ha de permitir ser adaptado a las necesidades de cada usuario con el fin de trabajar con diferentes sonidos y con diferentes complejidades según decidan los profesionales encargados de configurar el juego. También ha de ser intuitivo y fácil de utilizar.

1.3. Estructura del documento TFG

El documento se divide en los siguientes capítulos:

1. Introducción: Se describe el marco del proyecto del cual participa este trabajo y el objetivo principal del TFG.
2. Conceptos relacionados: Se hacen mención y definen conceptos relacionados con el contexto del sistema desarrollado.
3. El juego. +memory: Se describen las funcionalidades del juego desarrollado.
4. Marco tecnológico: Se describen las herramientas y tecnologías que se han utilizado para desarrollar este trabajo.
5. Metodología: Se define la metodología utilizada en este trabajo.
6. Análisis: Se especifican los requisitos de usuario y de sistema.
7. Arquitectura y estructura del software: Se describe el patrón de arquitectura utilizado para el desarrollo de la aplicación y la organización de las clases a partir de dicho patrón.
8. Datos de la aplicación: Se definen los datos tratados por la aplicación y las consultas realizadas con la base de datos.
9. Interfaz de usuario: Se describe el flujo de paneles de la aplicación, los tipos de paneles y demás características de la interfaz.
10. Control de la partida: Se muestra cómo el controlador realiza el control durante una partida.
11. Conclusiones y trabajo futuro: Se presentan las conclusiones tanto del proyecto como personales y en base a analizar el resultado del trabajo se plantean una serie de tareas como trabajo futuro y unas conclusiones.

2. Conceptos relacionados con el trabajo desarrollado

En este subcapítulo se van a introducir los conceptos relacionados con el proyecto, para conocer los problemas con los que se enfrentan las personas con discapacidad auditiva, las habilidades que han de mejorar, el entrenamiento para lograrlo y cómo motivar al usuario mediante un entrenamiento alternativo y entretenido.

2.1. Discapacidad auditiva.

Según la OMS, más del 5% de la población mundial (466 millones de personas) padecen pérdida de audición discapacitante, de las cuales 34 millones son niños. Es decir, 5 de cada 1000 niños nacen con una pérdida auditiva discapacitante o lo sufren a lo largo de su infancia. El 60% de los casos en niños que empiezan a padecer una pérdida de audición en su infancia podría evitarse [4]. La deficiencia auditiva es una pérdida de audición que puede ser profunda o no, por lo tanto, según el grado, la persona puede oír poco o nada. Se dice que una persona presenta discapacidad auditiva cuando sufre una pérdida auditiva de cierto número de decibelios con el oído que mejor oye: en el caso de los adultos, más de 40 dB; en el de los niños, más del 30 dB.

La discapacidad auditiva se puede dar después del nacimiento. Las causas pueden dividirse en dos conjuntos [4]:

1. Causas congénitas, es decir, las que se dan en el momento del nacimiento o poco después.
2. Causas adquiridas, que pueden provocar la pérdida de audición en cualquier edad.

Una de las causas adquiridas principales en niños es la infección crónica del oído (otitis crónica). Esta enfermedad es prevenible ya que se puede tratar. Otra causa importante y evitable es la exposición al ruido, comúnmente, dicha exposición se da en entornos de trabajo altamente ruidosos (adultos) y también en entornos lúdicos.

La discapacidad auditiva tiene múltiples consecuencias en la vida quien la presenta [4]:

1. Consecuencia social y emocional. Problemas de comunicación con los demás durante la vida cotidiana, esto puede llegar a generar sensación de aislamiento y de frustración.
2. Consecuencia funcional. Aparte de las limitaciones en la comunicación, las personas con discapacidad auditiva suelen tener mayores índices de fracaso escolar y necesitan asistencia educativa.
3. Consecuencia económica a nivel mundial. En los costes para la discapacidad auditiva se incluyen los del sector sanitario, apoyo educativo, costes sociales, etc.

2.2. Dispositivos de ayuda. Audífonos e implantes cocleares.

La tecnología y la medicina de manera conjunta tratan de resolver distintas necesidades que tienen las personas con discapacidades. En este caso, las personas con pérdidas de audición pueden mejorar su situación con la ayuda de dispositivos, entre ellos, audífonos e implantes cocleares. Ambos dispositivos se han diseñado para mejorar la calidad de la audición y se pueden adaptar a las necesidades del paciente.

El audífono consta de ciertos componentes externos, un altavoz, un amplificador y un micrófono [5]. Su finalidad es mejorar la percepción del sonido. Éste suele ser el primer dispositivo de ayuda que utiliza el paciente cuando se detecta la pérdida de audición. El implante coclear transforma las señales acústicas (sonidos y ruidos del ambiente) en señales eléctricas (energía eléctrica) que estimulan el nervio auditivo y envía señales al cerebro [6], sustituyendo así la función de la cóclea. Consta de diferentes partes, tanto internas como externas, por dicha razón el paciente tiene que someterse a una intervención quirúrgica para su implante y posterior uso y beneficio. La diferencia en la funcionalidad de ambos es que el implante no aumenta el sonido, sino que lo recoge y lo transforma para que el cerebro lo pueda tratar, mientras que el audífono amplifica y modifica las señales sonoras [7]. Con el uso del implante el paciente oye de una manera diferente de la audición normal.

Los expertos deciden qué dispositivo funcionará mejor para cada persona. En general, los implantes cocleares se suelen utilizar en pacientes que no consiguen un aprovechamiento correcto del audífono y también que tienen pérdidas auditivas localizadas en el nervio auditivo, y los audífonos, como ya se ha dicho, es lo primero que se suele utilizar al detectar la pérdida auditiva.

2.3. Logoterapia y entrenamiento auditivo.

El uso de dispositivos es útil pero el paciente necesita una ayuda extra ya que el sentido de la audición nos proporciona los sonidos, pero es la mente la que ha de identificarlos, diferenciarlos y darles un significado [3]. En resumidas cuentas, las personas con discapacidad auditiva que utilicen dispositivos auditivos de ayuda han de aprender a oír. La **logopedia** es fundamental para lograr este objetivo. Ésta es una disciplina profesional relacionada con las ciencias de la salud, la psicología y la lingüística aplicada [8]. En este caso, la logopedia trata la rehabilitación auditiva con un entrenamiento que consta de cinco fases: detección, discriminación identificación, reconocimiento y comprensión [9].

Figura 2.3: Fases del entrenamiento auditivo

A continuación, se describen las diferentes fases:

1. Detección. Detectar la presencia y ausencia de sonido. El paciente ha de notar la presencia de los sonidos y dirigir su atención hacia ellos. Se ha de trabajar con sonidos familiares y de distintas frecuencias.
2. Discriminación. Diferenciar las cualidades del sonido. Una vez que la personas es consciente del sonido se han de trabajar sus cualidades, es decir, ha de saber diferenciar las cualidades entre diferentes sonidos. Estas cualidades son la duración del sonido (largo o corto), la intensidad (suave o fuerte) y el timbre (grave o agudo). En esta fase, también se trabaja la memoria auditiva, el paciente ha de familiarizarse con los diferentes sonidos.
3. Identificación. Se ha de identificar un sonido dentro de un grupo cerrado. En esta fase, es necesario que la persona conozca el grupo en cuestión y que indique a qué sonido del grupo corresponde.

4. Reconocimiento. Se han de reconocer palabras o frases. Se trata de reconocer un sonido, pero, a diferencia de la Identificación, dentro de un grupo abierto.
5. Comprensión. Comprender un texto o diálogo. El objetivo es que la persona sea capaz de participar en un diálogo, logrando una comunicación interactiva.

Las fases del entrenamiento auditivo pueden llegar a solaparse, trabajando de manera simultánea en etapas diferentes. También cabe decir que las fases pueden dividirse en subfases de menor a mayor complejidad, por ejemplo, en la fase de reconocimiento, el aprendizaje va desde el reconocimiento de grupos semiabiertos a grupos de palabras que pueden no tener ninguna relación. Eso hace que el paciente pueda tener una progresión positiva dentro de una misma fase.

Este TFG se centra en entrenar las fases de Discriminación e Identificación. Para mejorar las habilidades incluidas en estas fases, se pueden realizar diferentes ejercicios. [10]

- Presentar al niño dos sonidos diferentes. El objetivo es que el niño ha de indicar qué sonido ha oído señalando una imagen u objeto. Esto pueden ser sonidos que, por ejemplo, provengan de un instrumento. Es importante que el niño también aprenda a oír sonidos de consonantes al inicio o final de las palabras y de diferentes frecuencias. [11].
- Entrenar con sonidos de diferentes volúmenes o longitudes. Para comprobar si el niño detecta la diferencia, puede repetir (imitar) el sonido que ha oído.
- Localizar el sonido dentro de un espacio. El niño ha de indicar de dónde proviene.

También se pueden trabajar exclusivamente las habilidades de identificación:

- Presentar al niño dos palabras parecidas fonéticamente, por ejemplo, “beber” y “deber”. Para analizar si el niño identifica las diferentes palabras.

2.4. Gamificación. Juego serio.

El niño puede encontrar el proceso tradicional de rehabilitación auditiva rutinario y no estar suficientemente motivado y a gusto para lograr los objetivos del entrenamiento. Para poder solucionar este problema, se pueden incluir aspectos afectivos y emocionales para incrementar su interés y poder aprender estas habilidades [3]. De aquí nace, el concepto de **gamificación**, más conocido en inglés *gamification*. Todavía no hay una descripción consensuada para este concepto, pero todas las fuentes coinciden en que es el proceso de introducir técnicas, elementos y dinámicas propias de los juegos y el ocio en actividades no recreativas para motivar la participación de las personas, alcanzar un objetivo o resolver un problema [10, 11]. En cambio, un juego serio es un juego digital creado con la intención de entretener y lograr al menos un objetivo adicional (por ejemplo, aprendizaje o salud) [12].

¿Por qué es más motivador el aprendizaje mediante el juego o técnicas de juego? Es una cuestión biológica, la dopamina es un neurotransmisor que ayuda a sentir curiosidad y motivación. Se encarga de motivar en momentos difíciles prometiendo una recompensa si se consigue alcanzar el objetivo. Si se consigue que la persona eleve sus niveles de este neurotransmisor, se obtendrá un incremento de su atención. El juego puede conseguir esos efectos haciendo que los niveles de dopamina se eleven y motivando a la persona, en este caso, al aprendizaje. [13]

La gamificación como los juegos serios buscan en el caso de la educación, que la persona pueda aprender de una manera entretenida y lúdica e incremente su motivación por aprender.

2.5. Trabajos relacionados

Cabe destacar varios trabajos relacionados con la discapacidad auditiva que trabajan diferentes aspectos relacionados con esta discapacidad:

- Juego serio para niños que sirve para aprender reglas sociales debido a un problema educativo. El jugador ha de tomar un papel en el cual tendrá que tomar la iniciativa y decisiones y han de cumplir unos objetivos. [14]
- Juego serio para niños orientado a la mejora de las habilidades motoras, en particular, desarrollar la percepción del ritmo) Camila Pérez. [15]
- Juego serio para niños para mejorar las habilidades de identificar y crear patrones rítmicos. [16]
- Proyecto para mejorar la habilidad de localización del sonido (saber de donde proviene) a través de teorías y bases para crear juegos electrónicos. Se establecieron 20 principios relevantes a tener en cuenta a la hora de crear un videojuego (interacción con el usuario, diseño de la historia, etc). [17]

3. El juego: **+memory**.

En la introducción se define brevemente el objetivo principal del juego. En este subcapítulo, se describen los objetivos generales y de qué manera se alcanzan.

El objetivo del trabajo consta de un juego serio, ver subcapítulo 2.4, desarrollado como aplicación de escritorio, cuyos usuarios finales son niños con discapacidad auditiva que necesitan desarrollar varias habilidades para aprender a oír, identificar y distinguir los sonidos. Las habilidades que se quieren trabajar forman parte de las 5 fases del entrenamiento auditivo explicadas anteriormente en el subcapítulo. Éstas son la discriminación y la identificación de sonidos dentro de un grupo cerrado.

+memory se asemeja al juego de asociación de cartas, *Memory*, en el cual en la pantalla el jugador observa un número finito de pares de cartas de las cuales no conoce su contenido ya que están dadas la vuelta. El jugador va seleccionando las cartas y éstas se voltean para que pueda ver el contenido por pares. Si el contenido del par está asociado no se vuelven a dar la vuelta. Si no lo está, éstas se vuelven a girar boca abajo. El objetivo del juego es que el jugador consiga que todas las cartas queden volteadas, es decir, que consiga asociar todos los pares de cartas. La dificultad del juego convencional, *Memory*, es, como su propio nombre indica el de recordar el contenido de las cartas para poder recordar su posición y así poder asociarlas.

La diferencia que hay entre el juego desarrollado en este trabajo y el juego convencional es que en el juego convencional se asocia pares de imágenes y con ***+memory*** no solo hay esa posibilidad, sino que también se pueden asociar combinaciones de imagen-sonido y sonido-sonido. Esta última combinación es la más compleja para el usuario (con discapacidad auditiva). Para entenderlo mejor, en una combinación imagen-sonido al voltear una carta con una imagen, ya saben que habrá una carta con un sonido que se corresponde a la imagen. Por ejemplo, el usuario puede voltear una carta y ver una imagen de un zapato y sabrá que hay una carta que al ser volteada se escuchará la palabra "zapato". Es decir, identificará el sonido dentro de un grupo conocido de palabras (fase 3 - Identificación, ver la figura 2.3.). En el caso de sonido-sonido, no se vería ninguna imagen, sino que se escucharía el sonido (la palabra oralizada) correspondiente en ambas cartas.

Como ya se ha comentado en los objetivos principales, ***+memory*** permite ser adaptado a las necesidades de cada usuario con el fin de trabajar con diferentes sonidos y con diferentes complejidades según decidan los profesionales encargados de configurar el juego. Para ello, el juego permite tener una amplitud de configuraciones y el usuario jugador puede elegir la configuración que desee y necesite en ese momento. El usuario administrador puede gestionar (crear, editar…) dichas configuraciones.

+memory tiene dos modos de juego: Modo Juego y Modo Entrenamiento. Para el modo juego no hace falta seleccionar una configuración, el usuario sólo selecciona una dificultad de entre tres (solo imagen, medio (imagen-sonido) y difícil (sonido-sonido). En el modo entrenamiento, el usuario necesita que haya una configuración creada anteriormente y juega a partir de las características de esa configuración.

Aparte de los aportes positivos al usuario referentes a la mejora de habilidades en el ámbito de la discapacidad auditiva, ***+memory*** también ayuda a mejorar otras habilidades gracias a que también aporta un entrenamiento cognitivo, mejorando la capacidad de memoria del usuario.

A continuación, se describe qué puede gestionar el usuario administrador:

- **Usuarios:** Los usuarios se dan de alta y de baja y se editan. Hace referencia a los jugadores del juego. Puede tener un nombre y fotografía (de perfil).
- **Tarjetas:** Las tarjetas se crean, se editan y se eliminan. Cada tarjeta es una asociación entre una palabra (texto), una imagen y un audio. La palabra es obligatoria, y al menos uno de los otros dos atributos ha de contener un valor, es decir, una tarjeta ha de tener asignado como mínimo una imagen o un audio, no puede tener asignada sólo la palabra.
- **Configuraciones:** Las configuraciones se crean indicando parámetros como el número de tarjetas o los sonidos a entrenar.

Tanto en la tarjeta como en la configuración se pueden asignar un grupo de fonemas[1] (con una posición) y un campo semántico[2] (p.e. Ropa: chaqueta, abrigo, pantalón). En cuanto a la posición del fonema, si una tarjeta tiene como palabra "copa" como fonemas-posición podría tener el fonema /k/ al inicio, /p/ al final. El administrador selecciona el texto del fonema y, opcionalmente, una posición (inicio, medio y final).

Por lo tanto, el jugador al seleccionar una configuración en el modo Entrenamiento, se realiza un filtrado de las tarjetas según las características definidas en la configuración. Por ejemplo, si una configuración tiene asignados dos fonemas /b/ y /m/ y ambas tienen como posición "Inicio", podrían seleccionarse tarjetas con las palabras "bota", "mota", "bote", mote", "besa" y "mesa". De esta forma el juego permite al jugador entrenar la discriminación de diferentes fonemas que pueden ser difíciles de distinguir.

Otros ejemplos de tareas de identificación y discriminación que realizan los logopedas incluyen los siguientes fonemas:

- /b/-/f/: base-fase, boca-foca, veo-feo, balda-falda, baba-faba, vino-fino.
- /ch/-/p/: chapa-papa, choza-poza, chillar-pillar, chino-pino, choca-poca, chata-pata.

Estas palabras podrían incluirse dentro del juego y se podrían preparar configuraciones que trabajen estas mismas palabras de forma más divertida.

[1] Un fonema es la articulación mínima de un sonido vocálico y consonántico

[2] Un campo semántico está constituido por un grupo de palabras relacionadas por su significado, compartiendo ciertas características comunes o semánticas.

4. Metodología.

En este capítulo se define la metodología utilizada en el proceso de desarrollo del sistema.

El diseño del sistema se ha realizado siguiendo la filosofía de Diseño Centrado en el Usuario (DCU). Este tipo de diseño trata de obtener sistemas más usables aplicando conocimientos de ergonomía y usabilidad. Según la Usability Professionals Association (UPA) [18], el DCU se puede definir como un enfoque de diseño cuyo proceso está dirigido por información sobre las personas que van a hacer uso del producto.

El proceso del DCU es cíclico, ver figura 4. y consta de un conjunto de actividades: En primer lugar, se ha de planificar el proceso de DCU; una vez realizada la planificación, se sigue un proceso iterativo donde se analiza el contexto de uso, se especifican los requisitos de usuario, se producen diseños y se evalúan. A partir de la última fase de este proceso se puede derivar a cualquiera de las demás fases, continuando el ciclo a partir de esa. Como resultado del proceso, se obtiene la solución diseñada.

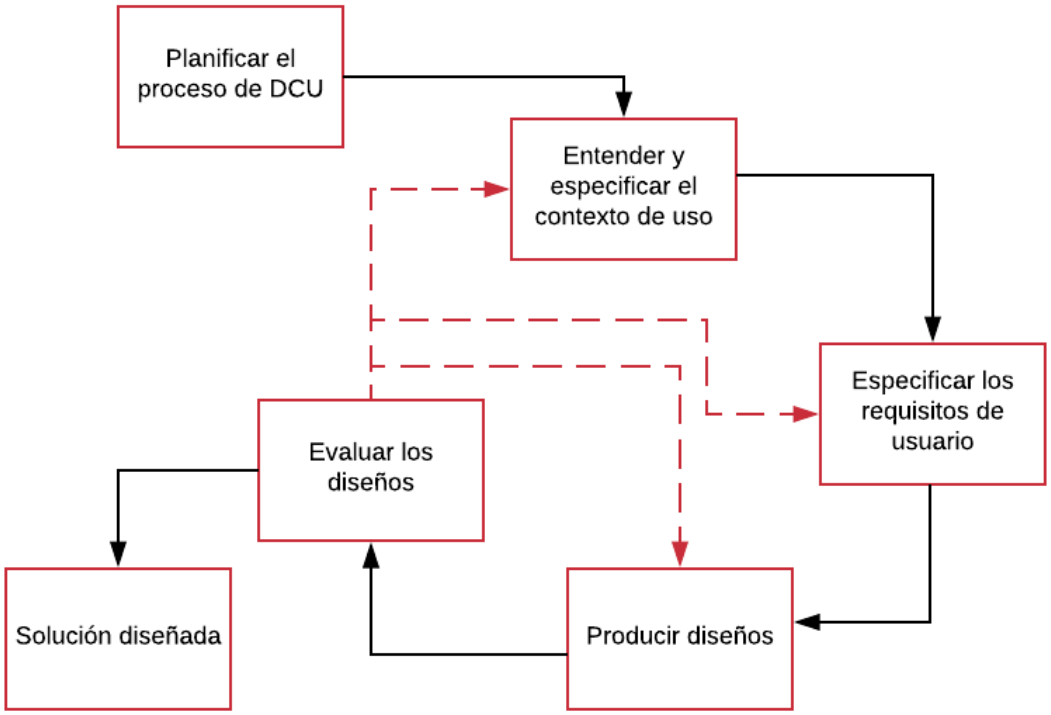


Figura 4.: Actividades del Diseño Centrado en el Usuario DCU

En este proyecto, las actividades del DCU se han aplicado de manera que a priori se ha tenido que realizar una búsqueda de información para conocer la discapacidad auditiva y qué necesidades tiene una persona que la presenta. También se ha utilizado documentación del proyecto al cual pertenece este trabajo. De esta manera, hay un acercamiento para entender el contexto donde se utiliza el sistema.

La Asociación de Padres de Personas con Discapacidad Auditiva (ASPAS) que participa en el proyecto de cooperación, también ayudó en la fase para especificar el contexto de uso y también

formó parte de la especificación de requisitos de usuario ya que refinaron los requisitos planteados inicialmente y añadieron de nuevos. La fase de diseño se realizó a través de prototipado en papel validado por una profesional de ASPAS y dividiendo el sistema en módulos donde cada uno de ellos presenta una funcionalidad. Los diseños aún no han sido evaluados por el usuario final, por lo que el alcance del trabajo llega hasta ese punto.

5. Análisis.

En este capítulo se especifican los requisitos del juego. Éstos se han obtenido gracias a una reunión con ASPAS los cuales aportaron requisitos específicos del entrenamiento que se lleva a cabo para mejorar las habilidades de identificación y discriminación de sonidos.

Antes especificar los requisitos es necesario indicar los dos roles de usuario diferentes que existen en juego: en primer lugar, el usuario administrador, que será el que se encargue de la gestión y la configuración del sistema; en segundo lugar, el usuario jugador, usuario que jugará a partir de la gestión realizada por el usuario administrador. Estos roles se han diferenciado sólo con el objetivo de distinguir las funcionalidades principales del sistema: la gestión (administrador) y el juego (jugador).

A continuación, se definen los requisitos:

Requisitos de usuario funcionales	
RF-1	El administrador creará configuraciones.
	Podrá escoger el nombre de la configuración, el número de parejas a asociar, un campo semántico y varios fonemas (y, por cada uno de ellos, la posición del fonema en la palabras de las tarjetas a filtrar) asociados. También tendrá que seleccionar la configuración de imagen y sonido (imagen-imagen, sonido-sonido o imagen-sonido). Tanto la elección del campo de semántico como del fonema y la posición del fonema serán opcionales.
RF-2	El administrador editará configuraciones.
	Una configuración podrá ser editada posteriormente a su creación.
RF-3	El administrador eliminará configuraciones.
	Una configuración podrá ser eliminada posteriormente a su creación.
RF-4	El administrador creará tarjetas.
	Podrá escoger la palabra asociada con la tarjeta, una imagen y/o un audio. Obligatoriamente tendrá que seleccionar uno de estos dos últimos. También opcionalmente podrá seleccionar un campo semántico y varios fonemas (y, por cada uno de ellos, la posición del fonema en la palabra) asociados.
RF-5	El administrador editará tarjetas.
	Una tarjeta podrá ser editada posteriormente a su creación.
RF-6	El administrador eliminará configuraciones.
	Una tarjeta podrá ser eliminada posteriormente a su creación.

RF-7	El administrador dará de alta a nuevos usuarios.
	Podrá seleccionar un nombre (obligatorio) y una fotografía (opcional).
RF-8	El administrador editará la información de usuarios dados de alta.
	Podrá editar tanto el nombre como la fotografía.
RF-9	El administrador dará de baja a usuarios.
	Podrá dar de baja a usuarios dados de alta anteriormente.
RF-10	El jugador seleccionará la configuración.
	Podrá elegir la configuración ya creada con la que desee jugar.
RF-11	El jugador seleccionará el modo de juego.
	Podrá elegir el modo Juego o modo Entrenamiento.
RF-12	El jugador seleccionará una dificultad en el modo Juego.
	Seleccionará una dificultad antes de iniciar una partida con el modo Juego.
RF-13	El jugador elegirá volver a jugar otra partida con la misma configuración o regresar al menú de usuario.
RF-14	El jugador pausará y reanudará la partida.
	Podrá pausar y reanudar la partida a su elección durante el juego.
RF-15	El jugador podrá seleccionar su perfil para acceder al menú de usuario.
	El usuario podrá seleccionar su perfil en un menú donde se muestran todos los perfiles.

Cuadro 5.-1: Requisitos funcionales de usuario

Requisitos del sistema funcionales	
RF-1	El sistema hará un control de errores a la hora de preparar una nueva partida.
	Cuando el usuario escoja una configuración, el sistema ha de buscar un cierto número de tarjetas acordes a la configuración seleccionada. Si no las encuentra, ha de avisar al usuario.

Cuadro 5.-2: Requisitos funcionales del sistema

Requisitos del sistema no funcionales	
RNF-1	El sistema dispondrá de una interfaz de usuario usable y fácil de entender.
	La interfaz gráfica ha de ser usable y fácil de entender tanto por los niños como por los administradores, siendo ambos usuarios finales del juego. La interfaz ha de ser lo suficientemente simple como para que no se tenga que dar una formación para su uso.
RNF-2	El sistema se desarrollará como aplicación de escritorio.
RNF-3	El sistema permitirá importar archivos de imágenes con formato .JPG y .PNG.
RNF-4	El sistema permitirá importar archivos de audio con formato .WAV.

Cuadro 5.-3: Requisitos no funcionales del sistema

En el siguiente capítulo se describen las herramientas utilizadas para desarrollar el sistema que cumpla con los requisitos especificados.

6. Marco tecnológico

A continuación, se exponen las herramientas que se ha utilizadon para el desarrollo de la aplicación y la justificación de su uso:

1. El lenguaje de programación ha sido **Java** [19]. Este lenguaje ha sido elegido ya que permite desarrollar todas las funcionalidades de la aplicación, incluyendo la conexión con la base de datos.
2. Como entorno de desarrollo se ha utilizado **NetBeans IDE** [20] ya que es el IDE oficial de Java 8. Entre otras cosas, permite escribir código de una manera rápida, por ejemplo, ofrece generadores de código (constructor, getter, setter) y una gestión de proyectos fácil de utilizar.
3. Para desarrollar la interfaz de usuario, se han utilizado las librerías gráficas **Swing** [21] y **Awt** de Java [22], puesto que se ha decidido tener pleno control de los gráficos mostrados en la aplicación. Estas librerías contienen las clases necesarias para crear elementos gráficos simples como, por ejemplo, etiquetas, botones y listas. Además, ofrece otras funciones como, repintado de objetos gráficos.
4. Para reproducir audios, se utiliza la librería de Java **Applet**. [23]
5. Como sistema gestor de base de datos se ha utilizado **MySQL** [24]. Se ha elegido por dos razones: es de código abierto y ya se tenía experiencia con su uso, tanto en los estudios como en las prácticas externas en empresa.

Para el uso de la base de datos, ésta se ha creado en un servidor local *localhost*. Con la ayuda de **Xampp** [25], que es un paquete de software libre de Apache Friends, se instaló MySQL y hace que se ejecute en un puerto determinado para poder acceder a la base de datos creada.

6. Para realizar la conexión desde NetBeans IDE a la base de datos, se utiliza **JDBC Java Database Connectivity** [26]. Ésta es una API que permite la ejecución de operaciones sobre la base de datos desde el lenguaje Java. Se ha de importar al proyecto Java. Para poder realizar las consultas se utiliza la librería **java.sql** [23] que ofrece el acceso y proceso de los datos almacenados en la base de datos.

7. Arquitectura y estructura del software

En este capítulo, se describe el patrón de arquitectura utilizado para el desarrollo de la aplicación y la organización de las clases a partir de dicho patrón.

7.1 Arquitectura (MVC)

Para desarrollar la aplicación, se ha utilizado la arquitectura **Modelo Vista Controlador MVC** [22]. Este patrón propone la separación de tres componentes: los datos de la aplicación (modelo), la interfaz de usuario (vista) y la lógica de control (controlador). Se basa en la separación de conceptos y facilitar la tarea del desarrollo. Comúnmente se utiliza para equipos de programación donde un grupo de desarrolladores modifican el código del mismo programa. Por ello, se separan las componentes de manera que una parte de ese grupo puede trabajar sobre el modelo, otra parte sobre la vista y otra última sobre el controlador. En el caso de este TFG, se ha elegido este patrón, en primer lugar, porque separar los componentes ayuda a la implementación del código ya que está mejor ordenado y, por último, porque teniendo en cuenta que este TFG forma parte de un proyecto, puede ser que otras personas que no hayan programado hasta el momento sobre el mismo código quieran realizar modificaciones o añadir funcionalidades adicionales. Por ejemplo, puede ser que se quiera mejorar la interfaz gráfica, en este caso los desarrolladores sabrán que todo el código referente a esa parte lo encontrarán en las clases de la vista. Esta razón hace que haya una mayor necesidad de orden y calidad para que el código resulte legible y comprensible. Otros beneficios son la facilidad de mantenimiento, la escalabilidad de la aplicación, es decir, la aplicación puede ser adaptada sin perder calidad, y además los componentes pueden ser reutilizados.

En ***+memory*** la vista toma un papel muy importante ya que todos procedimientos dependen de las acciones elegidas del usuario. Es decir, todas las funcionalidades de la aplicación se ejecutan mediante eventos. Por ejemplo, en el panel para el administrador se muestran tres botones. El programa se mantiene a la espera hasta que uno de estos botones sea pulsado.

7.2 Interfaces

Las tres componentes del MVC se han de poder comunicar entre ellas. Es por eso que se hace uso de **Interfaces**. En las interfaces se definen los métodos que debe tener obligatoriamente la clase que las implementa. En este documento se llama a este tipo de clases por su nombre en inglés *Interface* para que no haya confusión con la interfaz (*interface* traducido en español) de usuario.

Para cada parte de la arquitectura hay una clase principal y está implementa la *Interface* en cuestión (ver cuadro 7.2.).

Componente MVC	Clase principal	Nombre interfaz
Modelo	Datos	InterfaceModelo
Vista	Ventana	InterfaceVista
Controlador	Controlador	InterfaceControlador

Cuadro 7.2.: Información sobre las interfaces en la aplicación

En el InterfaceModelo,se encuentran definidos los métodos para la obtención y el almacenamiento de información referente al programa: tanto información sobre una partida como de la base de datos.

En el InterfaceVista se encuentra el método llamado por la clase principal del programa para mostrar la ventana del juego.

En el InterfaceControlador se encuentran los métodos para ayudar a gestionar los datos, es decir, si el administrador quiere crear una tarjeta, introduce datos sobre ésta en un panel y el controlador crea el objeto Tarjeta con esos datos. También contiene el método que hace los preparativos antes de iniciar una nueva partida.

Para lograr la comunicación entre partes del MVC, cada clase principal tiene como atributos punteros a las Interfaces de los demás componentes. De esta manera, pueden hacer llamadas a las clases declaradas en dichos atributos.

Otra curiosidad sobre este tipo de objeto es que las clases que los implementan han de tener obligatoriamente todas las clases definidas en la Interface correspondiente, pudiendo tener también otros métodos.

7.3 Clases del programa

La organización de las clases se basa en el MVC. Por lo tanto, éstas se separan en tres paquetes principales: Memory.Modelo, Memory.Vista y Memory.Controlador. A su vez, un paquete puede dividirse en varios subpaquetes. En las figuras 7.3.-1, 7.3.-2 y 7.3.-3 se pueden observar las clases organizadas.

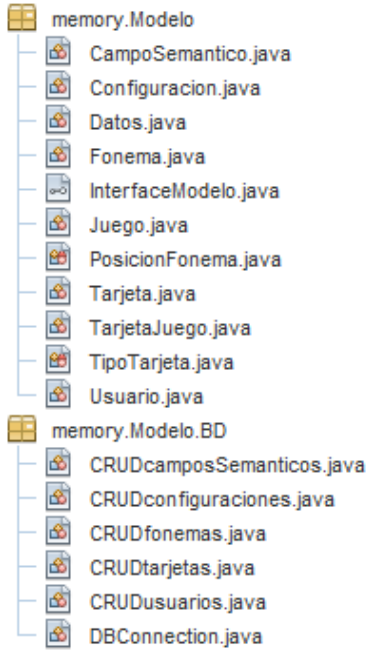


Figura 7.3.-1: Organización de las clases del modelo

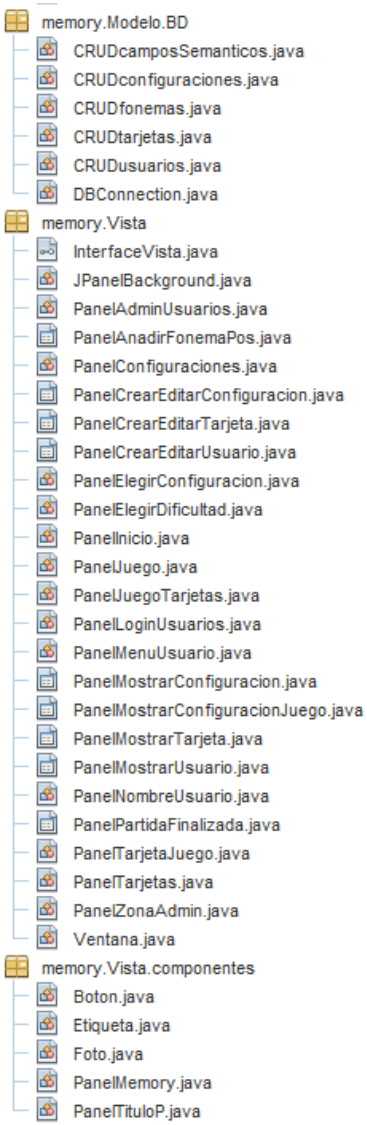


Figura 7.3.-2: Organización de las clases de la vista

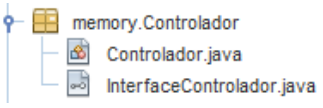


Figura 7.3.-3: Organización de las clases del controlador

En el paquete Modelo se encuentran las clases para instanciar objetos utilizados durante la gestión y el juego. También las clases con los CRUD's para obtener y modificar información de la base de datos. La clase principal del Modelo (que implementa la *interface*) es "**Datos**", ésta contiene todos los métodos llamados desde otros puntos del programa.

En el paquete Vista se encuentran todas las clases referentes a la parte gráfica de la aplicación (interfaz de usuario). La clase principal es "**Ventana**" y extiende (hereda) un *JFrame* en el cual se van intercambiando los diferentes paneles según las decisiones tomadas por el usuario. En este paquete también se incluyen todos los paneles que se visualizarán en la ventana y paneles que serán utilizados en otros paneles. Por ejemplo, el "PanelTarjetas" contiene varios paneles "PanelMostrarTarjeta", uno por cada tarjeta obtenida en la consulta a la base de datos. También se incluyen varias clases utilizadas como componentes mostrados en la interfaz y creados a partir de

clases de la librería Swing de Java, pero modificados para adaptarse a la aplicación y reducir el código. Por ejemplo, "Boton" es una clase que extiende (hereda) un "JButton" pero con las características visuales que se desean tener para un botón.

En el paquete Controlador se encuentra, aparte de la *interface*, la clase principal llamada "**Controlador**". Las funciones de ésta son las siguientes: (1) preparar objetos que contienen información para almacenar en la base de datos, por ejemplo, si se quiere añadir un nuevo registro en la tabla Tarjeta, (2) preparar el objeto Tarjeta a partir de los datos (palabra, imagen, etc.) obtenidos en la interfaz de usuario, posteriormente, hace una llamada a un método de la clase Datos pasándole como parámetro la Tarjeta creada; (3) otra función es la de preparar las *tarjetas de juego* para una nueva partida según la configuración que haya elegido el usuario (modo entrenamiento) o según la configuración de la dificultad seleccionada por el usuario (modo juego). Para saber más información sobre los modos del juego ir al subcapítulo 3.

En el siguiente capítulo, se define el tratamiento de los datos de la aplicación, clases de datos, consultas a la base de datos, etc.

8. Datos de la aplicación

El modelo es una parte muy importante de la aplicación ya que ésta ofrece al administrador una gestión de datos para el uso del juego, por lo que esta información añadida y/o modificada por el administrador se trata por la aplicación como un objeto y éste es almacenado en la base de datos (o actualiza el registro asociado al objeto).

Los elementos gestionados desde la aplicación son: Tarjeta, Configuración y Usuario. Se utilizan tres clases, una para cada uno de ellos, ClaseUsuario, ClaseTarjeta y ClaseConfiguración.

En la figura 8.-1 pueden verse los atributos de la clase principal del Modelo, Datos. Cuando en el menú para seleccionar usuarios se indica el perfil de usuario con el que se quiere jugar, éste se guarda en la clase Datos. Cuando se crea una partida nueva, ésta se guarda como Juego, también se guarda la configuración seleccionada por el usuario (modo entreno) o la creada según una dificultad (modo juego).

```
private InterfaceModelo datos;
private ArrayList<TarjetaJuego> tarjetasJuego = null;
private int filas, columnas;
private TarjetaJuego tarjetaLevantada1;
private TarjetaJuego tarjetaLevantada2;
private int paresAcertados = 0;
private int dificultad;
```

Figura 8.-1: Atributos de la clase principal del Modelo

Al obtener los fonemas o los campos semánticos de la base de datos se almacenan en una tabla hashing. Esto nos servirá a la hora de crear nuevos fonemas/campos semánticos, para consultar si existe alguno con ese nombre. Si existe, no se crea uno nuevo si no que se utiliza el existente. Si no, se añade a la base de datos y también a la tabla hashing.

Por último, otra clase importante es Juego. Un objeto de la clase Juego tiene como atributos los datos necesarios para el panel de juego (ver figura 8.-2): una lista con las tarjetas de juego mostradas en el panel, el número de filas y columnas del panel, los pares que ya se han asociado, la primera y segunda tarjeta levantada (si no están asociadas se vuelven a dar la vuelta, valor *null*) y la dificultad del juego (si se juega en Modo Juego).

```
private Usuario usuario_actual;
// configuración del modo entreno
private Configuracion configuracion_entreno;
// configuración del modo juego
private Configuracion configuracion_juego;
private Juego juego = null;

private HashMap<String, Fonema> fonemasHash;
private HashMap<String, CampoSemantico> camposSemanticosHash;
```

Figura 8.-2: Atributos de la clase Juego

Existen dos clases del tipo enumerado: TipoTarjeta, PosicionFonema. La primera hace referencia al tipo de tarjeta que se desea en el juego (sonido-sonido, imagen-imagen, sonido-imagen) y se selecciona en la creación de una configuración. La segunda indica la posición de un fonema en la palabra y se puede añadir (opcionalmente) al asignar un fonema a una configuración o a una tarjeta.

Por último, en el panel de juego se visualizarán las tarjetas de juego, los datos se encontrarán en la clase TarjetaJuego, ésta tendrá asociada una palabra (texto) y una imagen o un sonido según el tipo de la tarjeta. Un objeto Tarjeta, en la preparación de la partida, se dividirá en dos objetos TarjetaJuego. Por ejemplo, si el valor del tipoTarjeta del objeto Tarjeta es imagen-sonido, se declarará un objeto TarjetaJuego con imagen asignada y otra con sonido asignado.

8.1. Modelo de la base de datos

A continuación, se hace una descripción detallada del modelo de base de datos. En la figura 8.1. se pueden observar las tablas con las que cuenta la base de datos. Existen siete tablas, todas ellas tienen un atributo identificador, otros dos de tipo fecha (fecha de creación y fecha de la última edición realizada) y un atributo 'eliminado' el cual se refiere a si el elemento en cuestión ha sido eliminado (fonema_configuracion y fonema_tarjeta no contienen estos dos últimos atributos). Tanto las fechas como éste último atributo se han añadido para preservar los datos porque, aunque en la fase actual del sistema no se haga uso de ellos, en un futuro podría haber necesidad de obtener datos que ya hayan sido eliminados anteriormente. Por ejemplo, si se añadiera un historial donde se registran todas las partidas jugadas junto con la configuración utilizada en dicha partida. Si esa configuración ha sido eliminada por el administrador, en el historial debería verse igualmente la información de esa configuración utilizada. Por lo tanto, para el jugador y el administrador están eliminadas, pero en la base de datos se mantiene.

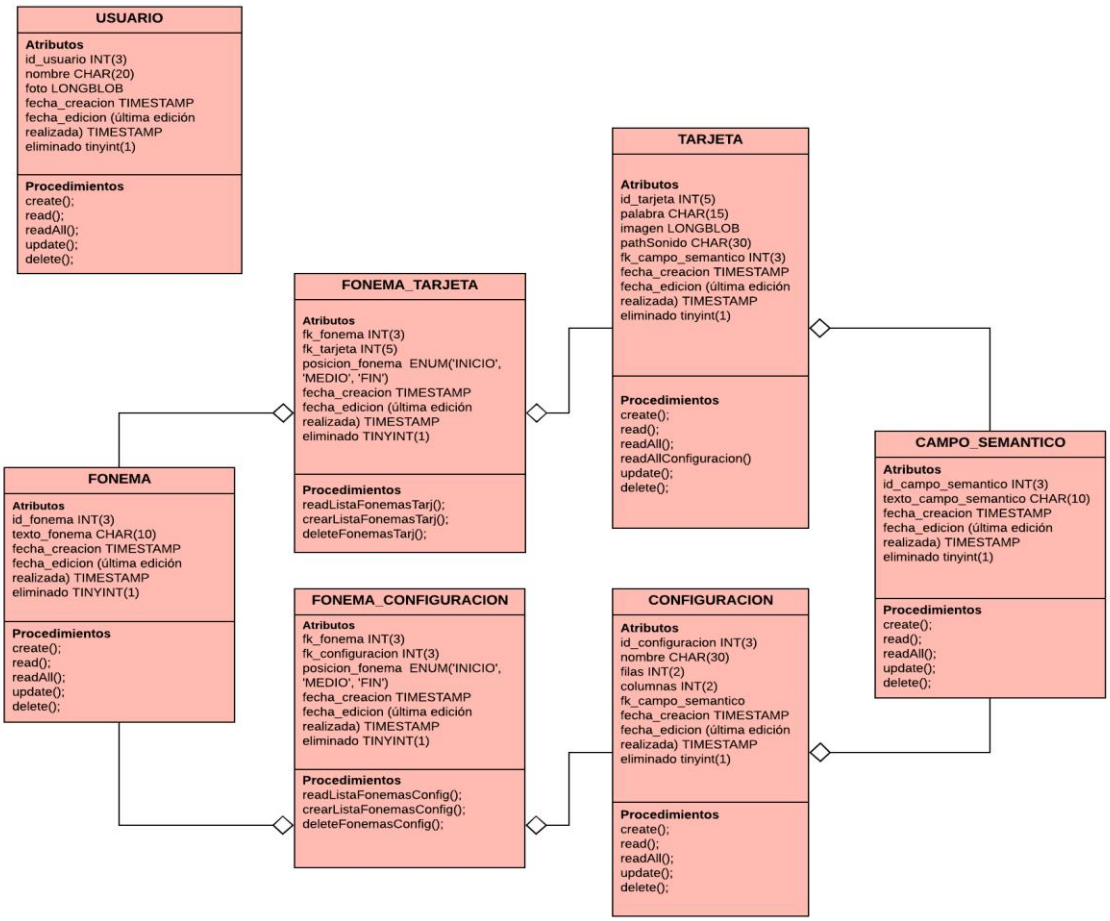


Figura 8.1.: Modelo de la Base de Datos

A continuación, se definen las tablas:

- La tabla **usuario** cuenta con el nombre del usuario (obligatorio) y una imagen, ambos se utilizarán, por ejemplo, en el menú de selección de usuario, donde se observan todos los usuarios (no eliminados) de la aplicación.
- La tabla **configuración** cuenta con el nombre (obligatorio), número de filas y columnas (de tarjetas de juego) en la partida (obligatorio), el tipo de configuración hace referencia a la combinación de tarjetas de juego, es decir, asociación de imagen-imagen, sonido-sonido o imagen-sonido (obligatorio), identificador de la tabla de fonema (*foreign key)*, posición del fonema en la palabra de la tarjeta y, por último, identificador de la tabla de campo_semantico (*foreign key)*.
- La tabla **tarjeta** incluye palabra (obligatorio), imagen, sonido, identificador de la tabla de fonema (*foreign key)*, posición del fonema en la palabra de la tarjeta y, por último, identificador de la tabla de campo_semantico (*foreign key)*.
- Las tablas **fonema** y **campo_semantico** cuentan con un texto.
- Las tablas **fonema_configuracion** y **fonema_tarjeta** son tablas intermedias que relacionan pares de fonema-configuración y fonema-tarjeta, respectivamente. Ambas tienen un

identificador del fonema y otro de la configuración o tarjeta. También una posición (enumerado, 'Inicio', 'Fin' y 'Medio') que puede tener valor nulo.

Las configuraciones son una parte muy importante de la aplicación ya que, a partir de la selección de una de ellas, se hará una búsqueda (SELECT) en la tabla de tarjeta con las características dadas por la configuración. Por ejemplo, si una configuración tiene como atributo tipo AMBOS, se ha de realizar una búsqueda de tarjetas con una imagen y un sonido asignados. Otro factor a tener en cuenta en el desarrollo es que a partir del número filas y columnas se han de encontrar un número determinado de tarjetas en la base de datos. Por lo tanto, si se necesitan 5*4 tarjetas de juego, se han de buscar (5*4)/2 tarjetas con las características de la configuración, para ello, se utiliza LIMIT en la selección (SELECT) para limitar el número de tarjetas obtenidas.

Como aclaración para los siguientes subcapítulos, se aclara la diferencia entre los conceptos de 'tarjeta' y 'tarjeta de juego'. Tarjeta se refiere a la asociación de una palabra, imagen y sonido, que se gestiona (crea, elimina, …). Una tarjeta se dividirá en dos tarjetas de juego al cargar la partida. Si la configuración seleccionada tiene como tipo 'Ambos', se dividirá en dos, una tarjeta de juego con una imagen y una tarjeta de juego con un sonido. Para entenderlo mejor, las tarjetas de juego son visualizadas por el usuario en la interfaz durante la partida.

En cada tabla, se consulta y modifica la información gracias a una serie de métodos básicos, éstos son los llamados CRUD (crear, leer, actualizar y eliminar). Se añade una método más a cada tabla, readAll(), con ella obtenemos todos los registros de la tabla (SELECT) que no estén eliminados (eliminado = FALSE).

Otro método añadido es *readAllConfiguracion()* con ella obtenemos las tarjetas que cumplan las características de una configuración. Este método es llamado antes de iniciar una nueva partida, para preparar las tarjetas de juego que se observan desde la interfaz durante el juego.

Para consultar cómo se han creado las tablas de la base de datos, consultar el anexo A.2.

8.2. Conexión con la base de datos

Desde la aplicación, se opera sobre la base de datos gracias a JDBC [21]. Para ello, se ha creado una clase que abre y cierra la conexión a la base de datos. Ver figura 8.2. para consultar el código que realiza esta funcionalidad. Para realizar la conexión se indica el driver, el nombre de usuario, la contraseña, la url de donde se accede a la base de datos (en este caso, se accede al servidor local, puerto 3306) y el nombre de ésta.

Se abre la conexión cada vez que se realice una consulta a la base de datos. Al finalizar las tareas que corresponden a la consulta, se cierra la conexión con la base de datos.

```
Connection conn = null;

final String url = "jdbc:mysql://localhost:3306/";
final String dbName = "memory";
final String driver = "com.mysql.jdbc.Driver";
final String userName = "root";
final String password = "";

public Connection open() {
    try {
        Class.forName(driver).newInstance();
        conn = DriverManager.getConnection(url + dbName, userName, password);

        if (!conn.isClosed()) {
            System.out.println("Database connection working using TCP/IP...");
        }

    } catch (Exception e) {
        Logger.getLogger(DBConnection.class.getName()).log(Level.SEVERE, null, e);
    }
    return conn;
}

public void close() {
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException ex) {
            Logger.getLogger(DBConnection.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Figura 8.2.: Código para abrir y cerrar la conexión con la base de datos

8.3. Consultar la base de datos

Para realizar las consultas a la base de datos, se ha creado una clase por cada tabla a excepción de fonema_configuracion y fonema_tarjeta que se incluyen en la clase de fonema). En la figura 7.3.-1 se muestran las clases utilizadas en las que se realizan las consultas a la base de datos. En ellas realizan los métodos CRUD y métodos adicionales. Cada tipo de consulta se realiza de manera diferente:

- Insert: Añadir un nuevo registro a una tabla pasándole como parámetro el objeto del cual se obtiene su información para realizar la consulta.
- Read: Obtener un registro de una tabla pasándole como parámetro su identificar.
- ReadAll: Obtener todos los registros de una tabla (excepto los eliminados, eliminado=falso).
- Update: Actualizar los datos de un registro en una tabla pasándole como parámetro el objeto del cual se obtienen los datos en cuestión.
- Delete: Actualizar un registro cambiando su valor de “eliminado” a verdadero. Se obtiene el identificador del registro en la tabla pasándole como parámetro.

A la hora de realizar las consultas hay una diferenciación en cuanto al código, se pueden dividir en dos tipos: consultas para modificar o añadir información en la base de datos (insert, update, delete) y consultas para obtener información de la base de datos (read, readAll). A continuación, se muestra un ejemplo de cada tipo: Para el primer caso, ver figura 8.3.-1, se realiza un Delete de un registro en

la tabla Tarjeta, en primer lugar, se abre la conexión con la base de datos, a continuación, se prepara la consulta y se asigna el identificador de la tarjeta. Finalmente, se ejecuta; En el segundo caso, ver figura 8.3.-2, se especifica el código Read para obtener un registro de la tabla usuario. Para ello, se prepara la sentencia indicando la cadena de caracteres que contiene la consulta (SELECT). Posteriormente, se ejecuta la consulta y se obtiene el resultado. Para almacenar los atributos del registro, se obtienen a partir del resultado, indicando el tipo de objeto a obtener. Por ejemplo, para obtener el Blob que contiene la imagen se utiliza *resultado.getBlob("foto")* donde "foto" es el nombre del atributo en la tabla de la base de datos.

```
public void delete(Tarjeta tarjeta) {
    try {
        DBConnection db = new DBConnection();
        Connection connDB = db.open();

        String query = "UPDATE tarjetas SET eliminado = 1 WHERE id_tarjeta = ?";
        PreparedStatement stmt = connDB.prepareStatement(query);
        stmt.setInt(1, tarjeta.getId());
        stmt.executeUpdate();

        db.close();
    } catch (Exception ex) {
        Logger.getLogger(CRUDusuarios.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Figura 8.3.-1: Consulta *delete()* en la tabla Tarjeta

```
public Usuario read(int id) {
    Usuario usuario = null;
    BufferedImage foto = null;
    try {
        DBConnection db = new DBConnection();
        Connection connDB = db.open();

        String query = "SELECT nombre, foto, fecha_creacion FROM usuarios WHERE id_usuario = ?";
        PreparedStatement stmt = connDB.prepareStatement(query);
        stmt.setInt(1, id);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            java.sql.Blob imgBlob = rs.getBlob("foto");
            if (imgBlob != null) {
                InputStream in = imgBlob.getBinaryStream();
                foto = ImageIO.read(in);
            }

            String nombre = rs.getString("nombre");
            usuario = new Usuario(id, nombre, foto);

            Timestamp fecha = rs.getTimestamp("fecha_creacion");
            usuario.setFechaAlta(fecha);
        }

        db.close();
    } catch (Exception ex) {
        Logger.getLogger(CRUDusuarios.class.getName()).log(Level.SEVERE, null, ex);
    }
    return usuario;
}
```

Figura 8.3.-2: Consulta *read()* en la tabla Usuario

Como apunte, después de ejecutar una consulta para insertar, *create()*, se requiere obtener el identificador del registro para guardarlo en el objeto de Java que lo almacena. Para ello, se utiliza la consulta "SELECT LAST_INSERT_ID()" del cual se obtiene como resultado el identificador del último registro insertado.

A parte de los métodos CRUD explicados anteriormente se han utilizado otros métodos adicionales. Uno de ellos es *readAllConfiguration()* con el cual a partir de una configuración pasada por parámetro se desea obtener un número concreto de tarjetas (este es el filtro comentado en anteriores subcapítulos). Este método se llama a la hora de preparar una nueva partida en modo Entrenamiento. A la hora de la selección se tiene en cuenta cada uno de los atributos de la configuración, es decir, el tipo (imagen-imagen, sonido-imagen, sonido-sonido), fonemas y posiciones (si se han asignado) y campo semántico (si se ha asignado). En la selección, también se limitan el número de tarjetas a obtener y se ordena la selección en modo aleatorio "SELECT …. ORDER BY RAND () LIMIT num_tarjetas_necesarias".

Por último, los registros de las tablas intermedias fonema_configuracion y fonema_tarjeta son asociaciones con identificadores de otras tablas (*foreign key)*. Por lo tanto, un registro de esta tabla se crea o elimina para añadir o eliminar una asociación. Los métodos relacionados con obtener información de estas tablas son *readListaFonemasConfig()* y *readListaFonemasTarj()* con los cuales se obtiene una lista de fonemas (con una posición si la tuviera) asociados con la configuración o la tarjeta, respectivamente. También *deleteFonemasConfig()* con el que se eliminan (permanentemente) asociaciones con una configuración en concreto, existe el mismo método para una tarjeta, deleteFonemasTarj(). Finalmente, el método para crear asociaciones entre fonemas y una configuración, o fonemas y una tarjeta, *crearListaFonemasConfig()* y *crearListaFonemasTarj()*, respectivamente.

Una vez que se conocen los datos, su significado en la aplicación y se obtención y modificación a partir de la base de datos, en el siguiente capítulo se describe cómo se ha desarrollado la interfaz de usuario con la cual se gestionan dichos datos, aparte de otras funciones.

9. Interfaz de usuario

La Vista es el componente del MVC más importante de la aplicación ya que casi todas las funcionalidades dependen de eventos que se crean a partir de las acciones del usuario. Una de las características más importantes especificada en los requisitos es dar forma de aplicación al juego, de esta manera, se implican varios factores:

- Ventana, la cual intercambia paneles según los deseos del usuario.
- Barra superior fija, la cual contiene el nombre del panel correspondiente, botón para ir hacia atrás (si lo permite el panel) y botón para salir del juego,
- La interfaz se ha de adaptar a la resolución de la pantalla.

En los siguientes subcapítulos, se detallan estas características y también cómo se han desarrollado. Además, se muestran los tipos de paneles y el flujo de paneles de ***+memory***.

9.1. Diagrama de flujo de paneles

En las figuras 9.1.-1, 9.1.-2 y 9.1.-3 se exponen los diagramas de flujo de los paneles en los cuales se muestran los paneles derivados (pasar de un panel a otro mediante un evento, por ejemplo, un click en un botón), los paneles contenidos en otros y las ventanas *pop up*, que pueden contener un objeto de una clase panel creada en el programa o no (por ejemplo, *JOptionPane* para confirmar). También podemos ver los paneles que se cargan directamente en la ventana. Conocer el flujo de paneles ayuda a entender mejor la ejecución del programa ya que, como se ha dicho anteriormente, los eventos que surgen mediante las acciones del usuario son los que influyen directamente en el flujo de ejecución.

Para no cargar demasiado los diagramas, se han definido los paneles (que pueden ser contenidos en la pantalla) desde los cuales no se puede ir al panel anterior, ya que el resto de los paneles de ese tipo sí que lo permite. También cabe añadir que hay más paneles, pero son simplemente por estética, no tienen ninguna otra función asignada.

Como adelanto, en los diagramas se muestra lógica interna del programa para mostrar el flujo de paneles en un caso en concreto, específicamente para comprobar si es posible realizar la partida. Pero existe más lógica en la aplicación que la mostrada en este subcapítulo.

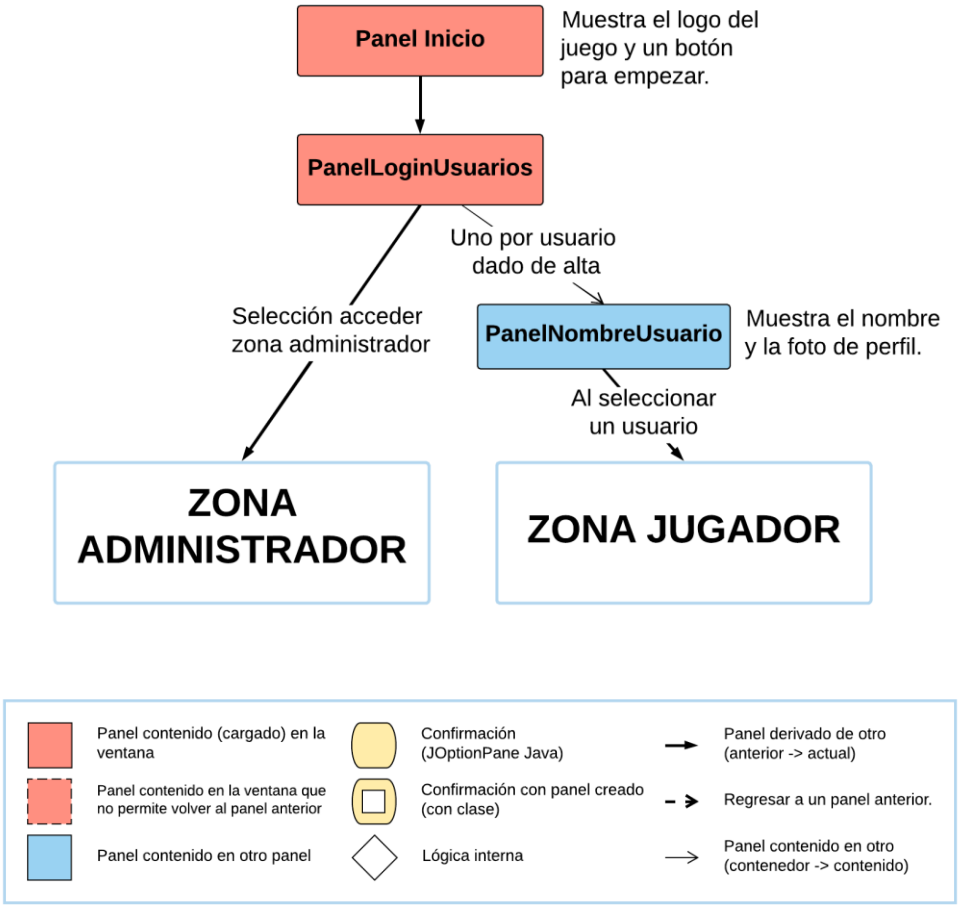


Figura 9.1.-1: Diagrama de flujo inicial

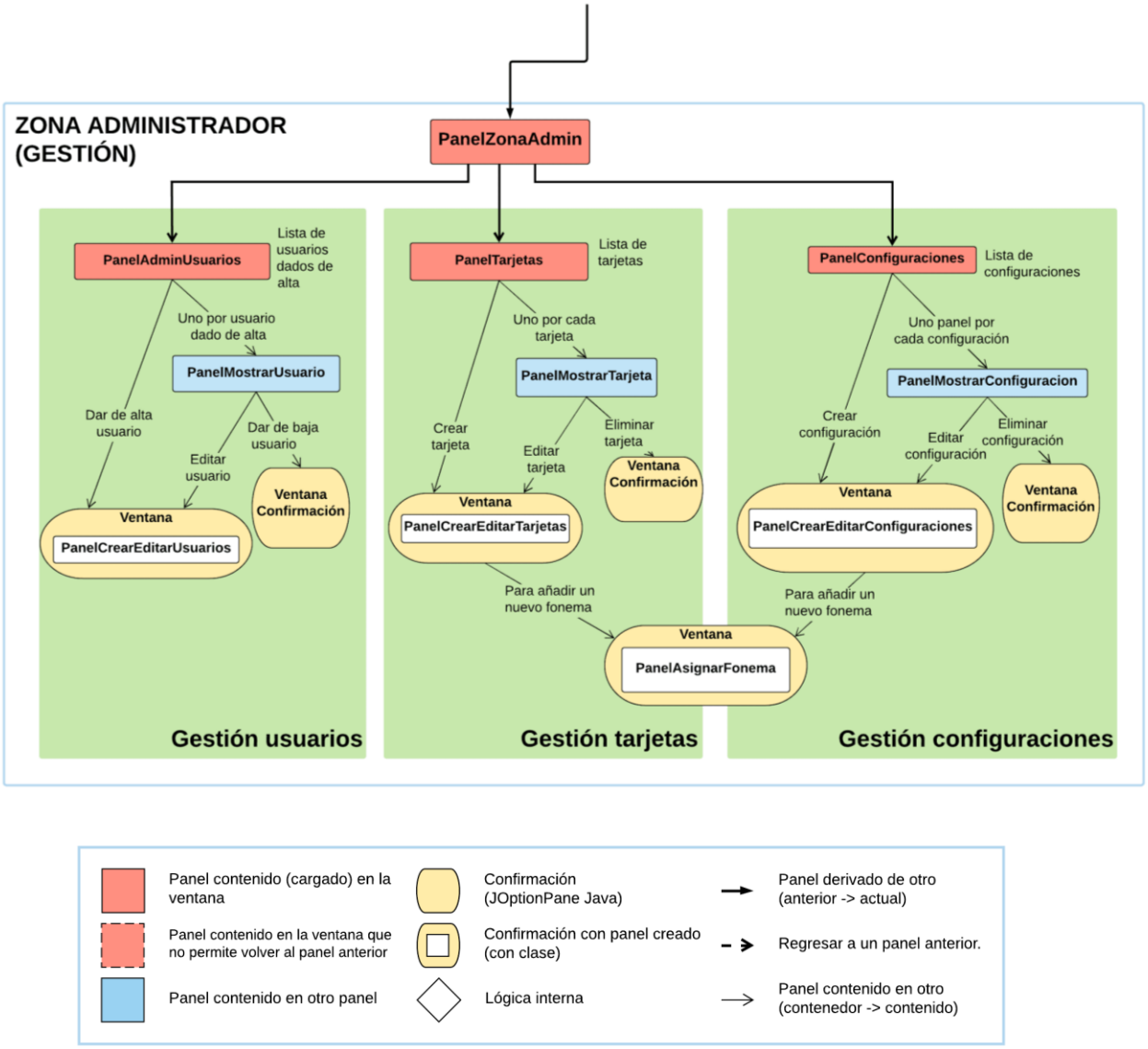


Figura 9.1.-2: Diagrama de flujo Zona administrador (gestión)

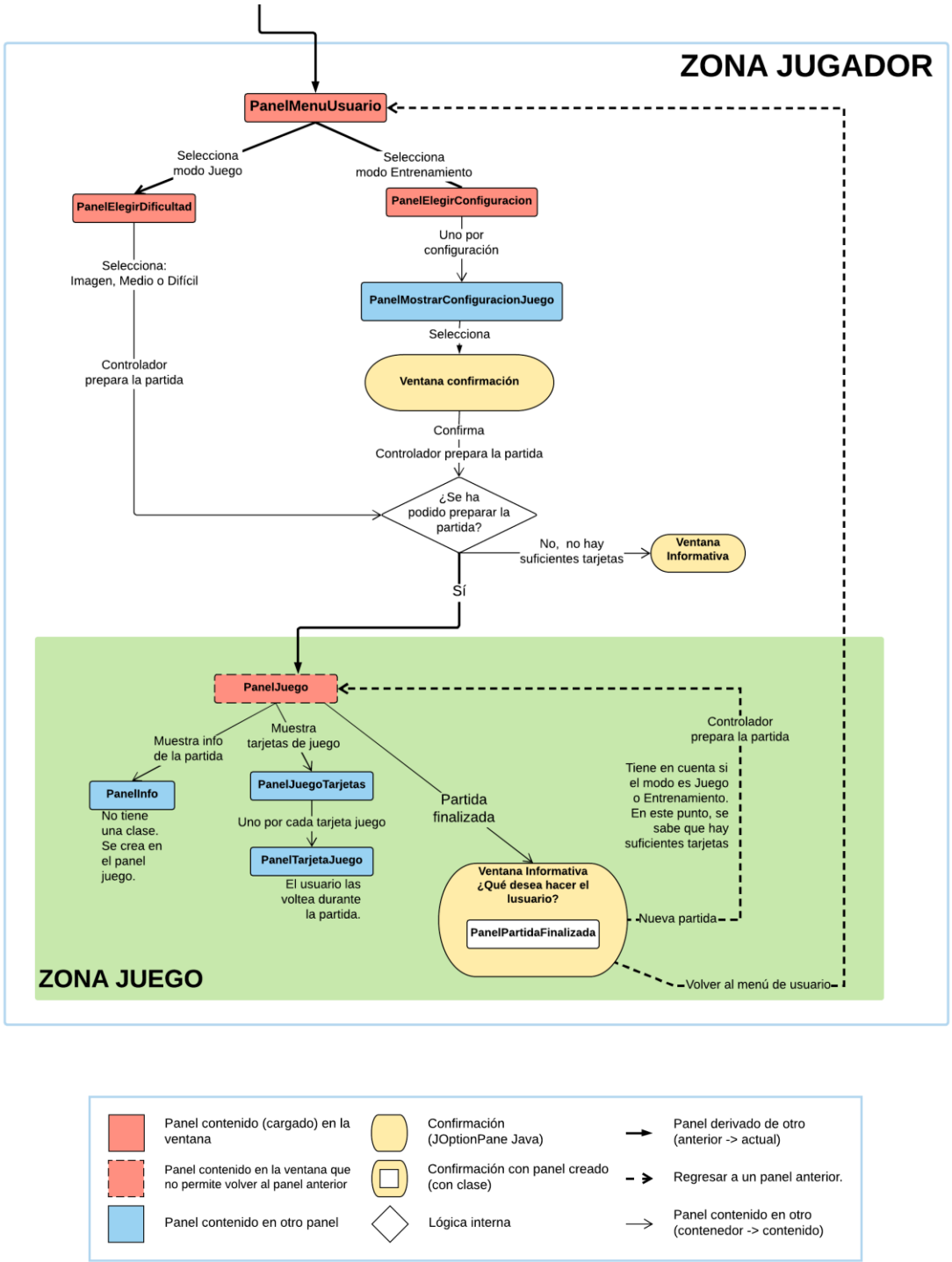


Figura 9.1.-3: Diagrama de flujo Zona Jugador / Juego

9.2. Cambio de panel

Una vez visto el diagrama de flujo que siguen los paneles, es interesante conocer cómo se realiza el intercambio de paneles contenidos en el Ventana. Esta clase tiene como atributos, aparte de otros, dos paneles ("panelAnterior" y "panelActual"). Cada vez que se desee cambiar de panel, se ha de cambiar el valor de dichos atributos. El valor "panelAnterior" será el del panel actual y éste último

guardará el valor de la declaración de un objeto (clase panel) que se desee en ese momento. En este subcapítulo, se habla sólo de paneles que puedan ser contenidos por la ventana.

Para poder llevar a cabo el intercambio de paneles con clases distintas, se hace uso de una de las características de la orientación a objetos: el polimorfismo, que hace referencia a la propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos. Por ello, la solución es crear una clase genérica que sea padre de todos los paneles. Con el uso de herencia, las clases hijas heredan los atributos y métodos de la clase padre. Esta clase padre recibe el nombre de **PanelMemory** y a continuación se muestran sus atributos:

- Ventana (Ventana): Las clases hijas pueden hacer llamadas a la Ventana. Por ejemplo, para obtener el puntero a la InterfaceControlador y poder llamar a alguno de los métodos del Controlador declarados en ésta.
- PanelAnterior (PanelMemory): Almacena el puntero al panel anterior al actual. Cuando la ventana contiene el panel de inicio (que muestra la ventana) este tiene este atributo con valor nulo.
- Título (cadena de caracteres): Título del panel actual.
- PermitidoAtras (boolean): Valor booleano que indica si es posible ir hacia atrás. Si es así, en el panel título, explicado en el subcapítulo 9., no mostrará el botón con dicha funcionalidad. Un panel que no puede regresar al anterior es el panel juego ya que para ello se ha de finalizar la partida.

En el constructor de PanelMemory se pasan como parámetro estos cuatro atributos. Para ello, es obligatorio que la primera línea de código dentro de los constructores de las clases hijas tenga la siguiente forma:

```
super(ventana, panelAnterior, título, permitidoAtras)
```

Éste no es un aspecto propio del programa, sino del lenguaje y se realiza para poder llamar al constructor de la clase superior (padre) con los valores deseados.

Otro dato curioso sobre PanelMemory es que éste no es hijo de un panel propio de la librería Swing, si no otra clase que sí que es hija de un JPanel (Swing). Esto se realiza para poder poner una imagen como fondo de los diferentes paneles. La clase tiene el nombre de JPanelBackground y se ha obtenido de fuentes externas (anexo A.1.). Esta tiene un método con el cual repinta el propio objeto de manera que dibuja la imagen y hace que se vea como fondo del panel.

Al inicio de este capítulo se ha comentado el concepto de interfaz de usuario que se desea tener en la aplicación, utilizando una barra superior fija en la que se muestra un botón (flecha) para volver hacia atrás, el título y un botón para salir del programa. Esta barra es un objeto de la clase PTitulo. Al cambiar el panel de la ventana, se ha de adaptar al nuevo panel, por ejemplo, cambio de título, ocultar botón para ir hacia atrás. etc. En la figura 9.2.-1 se muestra un ejemplo de barra superior dentro de la aplicación.

Figura 9.2.-1: Ejemplo de barra superior.

Una vez que se conocen las clases de objetos relacionados con el cambio de panel, se puede especificar mejor cómo se ha desarrollado. Cuando un objeto de una clase perteneciente a la vista desea cambiar de panel, ésta llama al método *notificar* de la ventana pasándole por parámetro una cadena de caracteres con la cual este método sabe que panel se ha de cargar. Los pasos son los siguientes:

1) Detectar (mediante un *switch*) que panel cargar.
2) PanelAnterior toma el valor del PanelActual.
3) PanelActual es un objeto nuevo de la clase panel deseada.
4) Cambiar título de la barra superior

Al realizar los cambios de paneles de esta manera siempre se puede acceder a cualquiera de los paneles anteriores ya que todos y cada uno de ellos se enlazan con el panel anterior. Todo esto se puede realizar ya que en PanelAnterior y PanelActual se guardan referencias a los objetos. Esto sirve a la hora de volver al panel anterior (cuando el usuario pulsa el botón correspondiente en la barra superior) donde el panel actual tomará el valor del panel anterior y el panel anterior el consiguiente. También es útil a la hora de regresar al menú de usuario al finalizar una partida, ver figura 9.2.-2, donde el panel actual toma el valor de dos paneles anteriores y el panel anterior uno más atrás. Esta técnica también permite no realizar excesivas declaraciones a nuevos objetos de clase panel, utilizando objetos ya creados anteriormente.

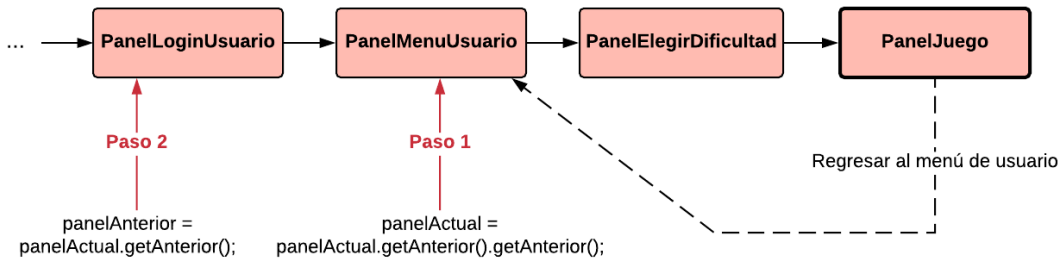


Figura 9.2.-2: Asignación de nuevos valores a los punteros panelAnterior y panelActual, en el regreso al menú de usuario después de finalizar una partida (modo Juego).

9.3. Tipos de paneles.

En este subcapítulo se describen con más detalle y se muestran mediante figuras los diferentes tipos de paneles esquematizados en el subcapítulo 9.1. donde se visualiza el flujo de paneles en la aplicación. En estos diagramas se pueden apreciar los diferentes tipos de paneles. En primer lugar, tenemos los paneles que permiten entrar en dos zonas en las que esquemáticamente se separa la aplicación. Estos paneles son: el panel de inicio (ver figura 9.3.-1), instancia de "PanelInicio", donde simplemente se muestra un botón para empezar que irá al siguiente panel; y el panel donde se loguean los usuarios seleccionando el perfil que quieran, instancia de "PanelLoginUsuario", que también cuenta con un botón para acceder a la zona administrador (ver figura 9.3.-2).

Figura 9.3.-1: Panel inicial de la aplicación.

Figura 9.3.-2: Panel para entrar al menú de usuario y a la zona administrador.

A partir del último panel mencionado se puede acceder a dos zonas: zona usuario y zona administrador, en las cuales el usuario puede seleccionar un tipo de juego con el objetivo de iniciar una nueva partida o gestionar los datos de la aplicación (usuarios, tarjetas y configuraciones), respectivamente.

En primer lugar, es interesante conocer cómo se ha desarrollado el panel para acceder al menú de usuario y a la zona administrador, estas dos funciones se realizan mediante un panel donde se selecciona el usuario deseado y un botón, respectivamente.

El panel contenido en el PanelLoginUsuario tiene forma de tablero y se puede realizar *scroll* (ver figura 9.3.-2), es decir, tiene una barra lateral derecha que permite desplazarse por el panel visualizando de esta forma todos los usuarios dados de alta. Para poder dar forma de tablero al panel se ha modificado su diseño utilizando *GridLayout* de la librería AWT, al cual se le pasa por parámetro el número de filas y columnas deseadas (en este caso, se establece un número máximo de columnas, 4). Al añadir los usuarios al panel, éste se va adaptando al diseño especificado. Al finalizar este último paso, a partir del panel obtenido, se crea un objeto panel de la clase JScrollPane que, finalmente, será el que se añada a PanelLoginUsuario.

Este tipo de paneles "cuadrícula" se utilizan en varios sitios de la aplicación:

- Zona administrador. Al mostrar una lista con los usuarios, las tarjetas y las configuraciones. Ver figuras 9.3.1.-2, 9.3.1-2 y 9.3.1-3..
- Zona usuario. Al seleccionar una configuración antes de iniciar una nueva partida (modo entrenamiento). Ver figura 9.3.2.-3**.**

En todos los casos, el uso de la cuadrícula permite adaptarse dinámicamente ya que durante la ejecución pueden crearse nuevos elementos y estos se han de visualizar también una vez añadidos.

```
private JScrollPane inicializarPanelListaUsuarios() {
    panelUsuarios = new JPanel();

    // Acceso a la base de datos para obtener los usuarios dados de alta.
    ArrayList<Usuario> usuarios = new CRUDusuarios().readAll();
    int numFilas = usuarios.size() / 4;
    numFilas++;

    // Asignar al layout un tablero
    panelUsuarios.setLayout(new GridLayout(numFilas, 4));

    numUsuarios = usuarios.size();
    for (int i = 0; i < numUsuarios; i++) {
        Usuario usuario = usuarios.get(i);
        panelUsuarios.add(new PanelNombreUsuario(ventana, usuarios.get(i).getId(),
                usuarios.get(i).getNombre(), usuarios.get(i).getFoto()));
    }
    // Necesario para posteriormente ajustar el tamaño del panel
    height = 250 * numFilas;

    // Hacer del panel creado un panel que permita "scroll".
    JScrollPane panelUsuario = new JScrollPane(panelUsuarios);

    panelUsuarios.setBackground(Color.WHITE);

    return panelUsuario;
}
```

Figura 9.3.-3: Código para crear el panel que permite seleccionar usuarios.

9.3.1. Paneles de la zona administrador.

A continuación, se definen los paneles de la zona administrador. En la figura 9.3.1.-1 se muestra un panel que incluye tres botones, cada uno de ellos para acceder a las tres gestiones de la aplicación (gestión de usuarios, gestión de configuraciones y gestión de tarjetas).

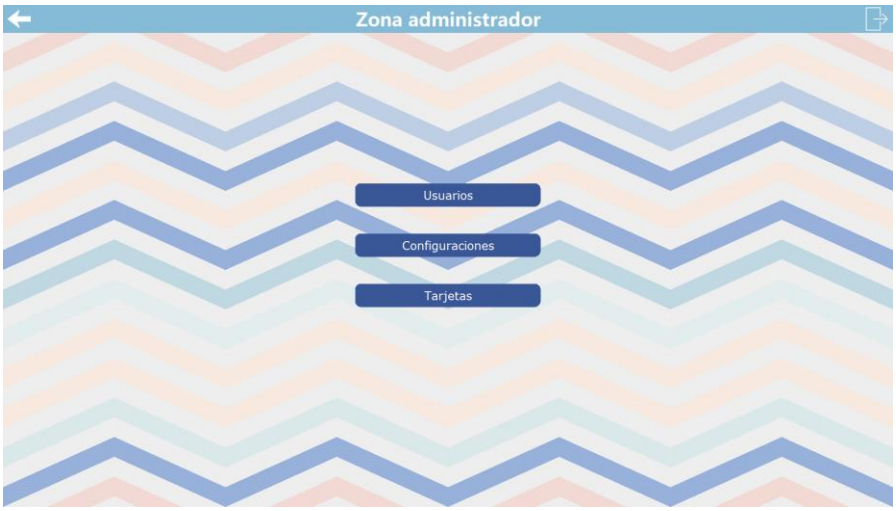


Figura 9.3.1.-1: Panel inicial de la zona administrador

En cada gestión, se tienen los siguientes tipos de paneles:

- Lista de elementos, ver figuras 9.3.1.-2, 9.3.1.-3 y 9.3.1.-4. Cada uno de ellos consta de un panel con la información del elemento en cuestión que incluye dos botones (botón para

editar y otro para eliminar). El panel con la lista también consta de un botón para añadir (crear) un nuevo elemento. Tiene la característica explicada anteriormente (cuadrícula de N filas x 1 columna, siendo N el número de elementos a visualizar).

- Panel de edición / Edición, ver figuras 9.3.1.-5, 9.3.1.-6 y 9.3.1.-7: Los paneles creación/edición contienen formularios para crear o editar elementos. En la figura **9**.3.1.-8 se muestra un ejemplo de panel de edición.

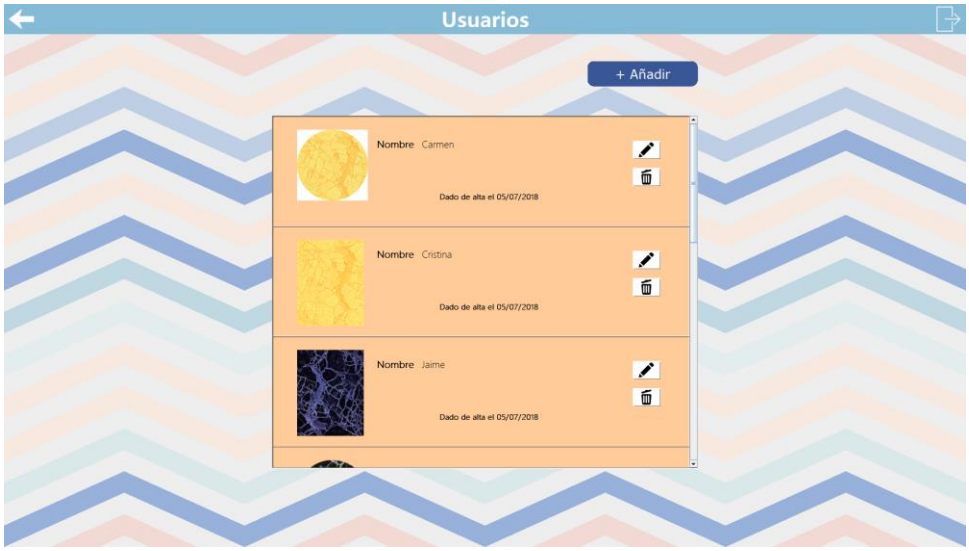
Este panel se visualiza en una ventana tipo *pop up* que a la hora de ser creada se inhabilita la utilización de la ventana principal gracias al método ventana.setEnabled(false), al cerrarse la ventana de creación y edición ha de volver a habilitarse.

La ventana que contiene el panel de edición/creación se visualiza tanto cuando se pulsa el botón de añadir nuevo elemento como si se pulsa el de editar un elemento en concreto. Esto puede realizarse ya que el panel se adapta a éstas dos opciones. Si se desea editar, las componentes de éste se rellenan con los datos actuales del elemento. Si se desea crear, aparecen inicializadas.

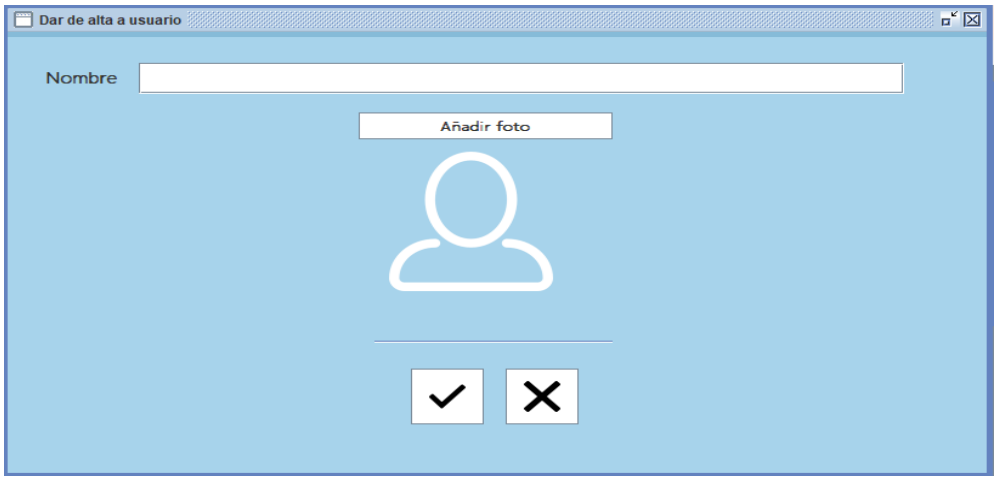
En los paneles crear/editar usuarios y tarjetas, el administrador puede seleccionar una imagen correspondiente al elemento, en el de la tarjeta también puede seleccionar un archivo de audio. Para ello, se utiliza un un objeto de la clase *JFileChooser*, que es una ventana emergente con la funcionalidad de seleccionar archivos locales. Utilizando esta clase se pueden poner filtros: selección sólo de archivos (no directorios), deshabilitar la selección múltiple y elegir las extensiones permitidas (.WAV para audio y .JPG y .PNG para imágenes).

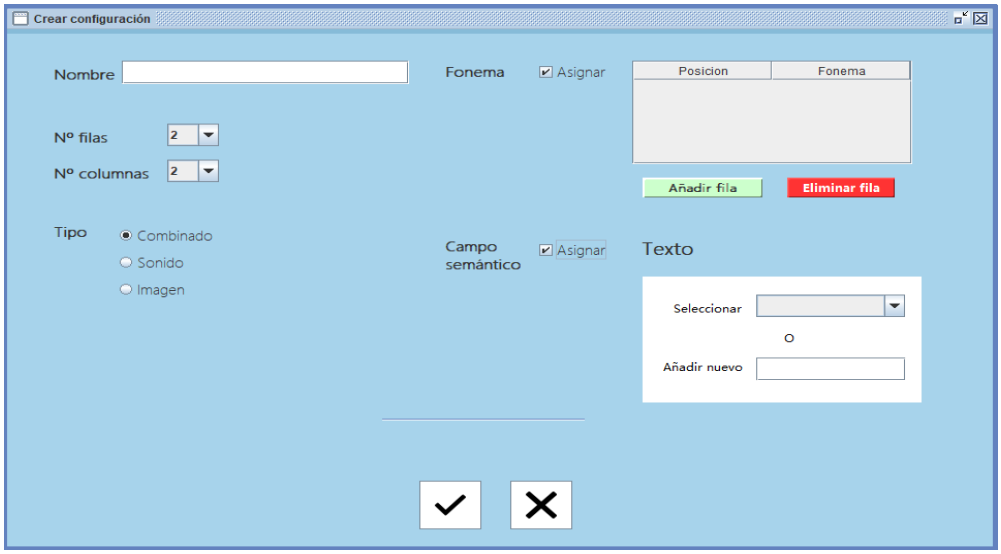
En los paneles crear/editar configuraciones y tarjetas, el usuario puede seleccionar un fonema y/o un campo semántico. Al pulsar el botón para añadir un nuevo fonema, se visualiza una ventana *pop up* con el panel para seleccionar o añadir un fonema y asignar también la posición del fonema en la palabra. Ver figura 9.3.1.-9.

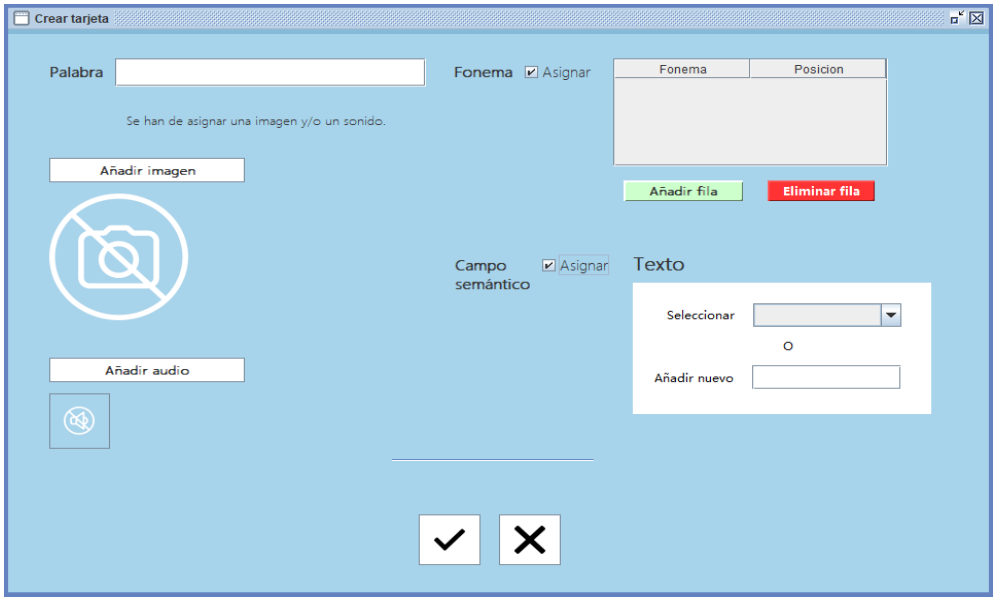
Para la eliminación de elementos se utiliza una ventana *pop up* de la clase *JOptionPane* para que el usuario pueda confirmar si desea realizar la eliminación.



Figuras 9.3.1.-2, 9.3.1.-3 y 9.3.1.-4 (de arriba a abajo): Paneles de visualización (usuarios, configuraciones y tarjetas).







Figuras 9.3.1.-5, 9.3.1.-6 y 9.3.1.-7 (de arriba a abajo): Ventanas con panel creación/edición (usuarios, configuraciones y tarjetas).

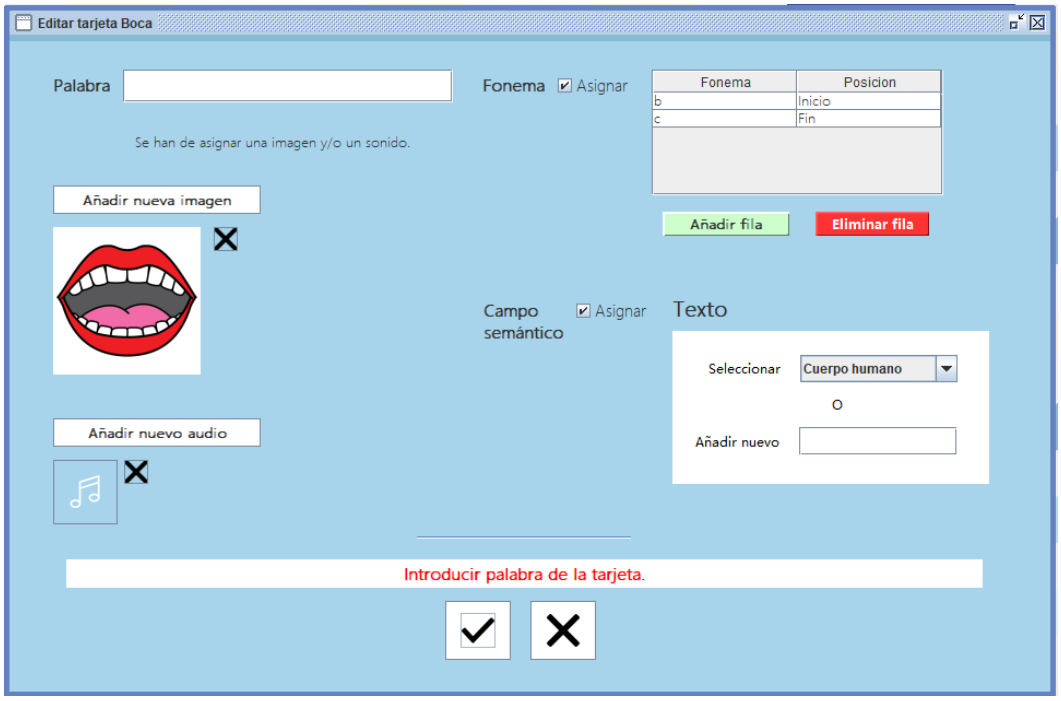


Figura 9.3.1.-8: Panel de edición de tarjeta con mensaje de error.

Figura 9.3.1.-9: Panel para añadir nuevo fonema y posición del fonema.

Como ya se ha comentado, los paneles creación/edición contienen formularios para crear y editar elementos. Cuando el administrador selecciona el botón de guardar, se ha de realizar una gestión de errores antes de realizar la acción y almacenar (o modificar) los datos en la base de datos. Esta gestión se realiza dentro de la propia clase del panel. Se tienen en cuenta las siguientes consideraciones:

Crear/Editar usuarios	El campo nombre es obligatorio.
Crear/Editar configuraciones	El campo nombre es obligatorio.
	La multiplicación de filas y columnas ha de ser un número par.
	Comprobar si se ha seleccionado un audio y/o imagen, al menos uno de los dos.
	Otras consideraciones: Si el número de filas es mayor al número de columnas, intercambiar valores. Si el fonema o el campo semántico se han añadido nuevos, comprobar si ya existen, si es así, utilizar los existentes.
Crear/Editar Tarjetas	El campo palabra es obligatorio.
	Otras consideraciones: Si el fonema o el campo semántico se han añadido nuevos, comprobar si ya existen, si es así, utilizar los existentes.
Añadir nuevo fonema	Obligatorio seleccionar un fonema o añadir un nuevo (una de las dos).

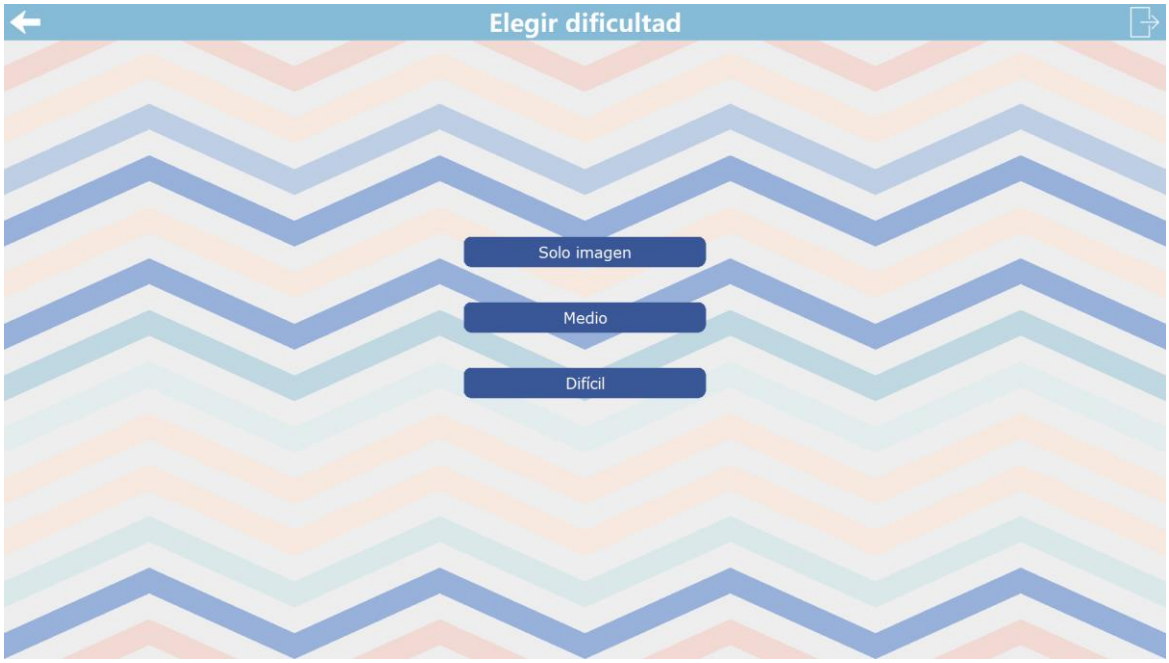
Cuadro 9.3.1.: Información sobre la gestión de errores.

9.3.2. Paneles de la zona usuario jugador

En esta zona los paneles tratan sobre la elección del modo de la partida y la propia partida. La figura 9.3.2.-1 muestra el menú de usuario donde puede seleccionar el modo (Juego o Entrenamiento).

Figura 9.3.2.-1: Panel que contiene el menú de usuario.

Si se pulsa el botón "Jugar" se muestra en la pantalla un panel en el cual se puede seleccionar la dificultad del juego, ver figura 9.3.2.-2. Si se pulsa el botón "Entrenar", a continuación, se muestra un panel para seleccionar la configuración a partir de la cual se desea entrenar, ver figura 9.3.2.-3. Al seleccionar la configuración, aparece una ventana *pop up* para que el usuario confirme que desea empezar la partida. Si no se encuentran tarjetas suficientes para la configuración seleccionada, se da un aviso al usuario mediante el una ventana *pop up* y permite seleccionar otra configuración.



Figuras 9.3.2.-2 y 9.3.2.-3 (de arriba a abajo): Panel para seleccionar la dificultad de la partida (modo Juego) y panel para seleccionar la configuración (modo Entrenamiento).

Finalmente, una vez cargada la partida se muestra el panel respectivo, figura 9.3.2.-4, para ambos modos de juego se muestra el mismo panel, solo cambia el título de la barra superior (aunque ésta forme parte de un panel diferente). Se muestra un panel con la información y un panel cuadrícula con las tarjetas del juego. Cada tarjeta puede estar boca abajo o volteada, y puede contener una imagen (al voltearla se muestra la imagen relacionada con la tarjeta y la palabra), o un sonido (al voltear se muestra una imagen de "sonido" y se reproduce el audio de la tarjeta. El usuario ha de pulsar las tarjetas para que estas se volteen. Cada dos tarjetas volteadas, si están asociadas, permanecen volteadas y se actualiza el porcentaje de completado contenido en el panel de información, y, en el caso de las tarjetas de sonido, también se mostrará el texto de la palabra asociada a la tarjeta. Si se voltean dos y no están asociadas, a los tres segundos, se vuelve a poner boca abajo. Ver figura 9.3.2.-5 para ver el panel durante la partida. Al finalizar la partida se muestra una ventana *pop up* con un panel, ver 9.3.2.-6, en el cual el jugador puede elegir si volver al menú de usuario o volver a jugar una partida, en este último caso se vuelve a preparar la partida según la dificultad (modo Juego) o la configuración (modo Entrenamiento) seleccionados para la anterior partida.

Figuras 9.3.2.-4: Panel al inicio de la partida con las tarjetas del juego boca abajo.

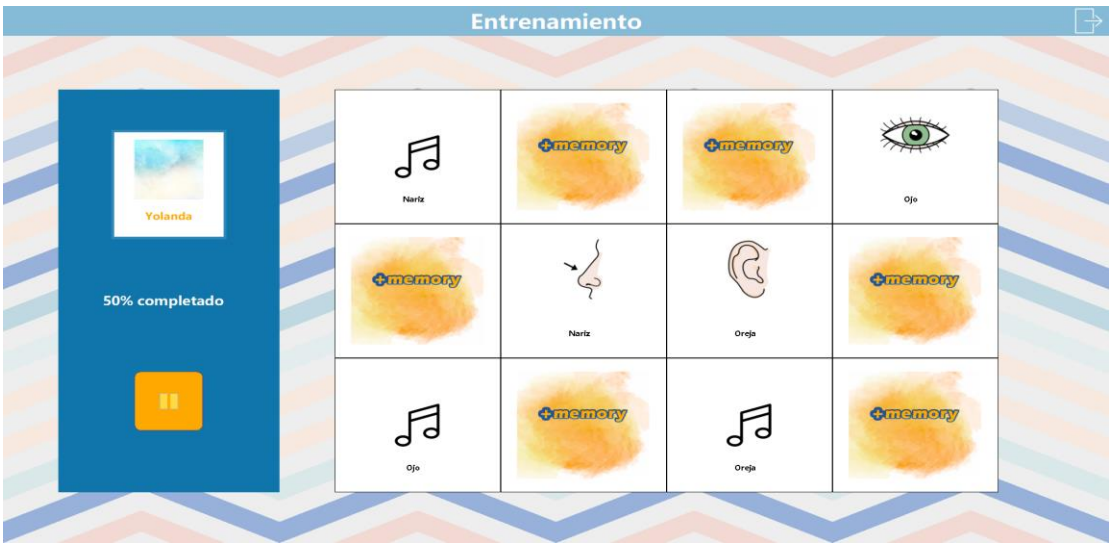


Figura 9.3.2.-5: Panel durante la partida con tres pares acertados.

Figura 9.3.2.-6: Ventana que se muestra al finalizar una partida.

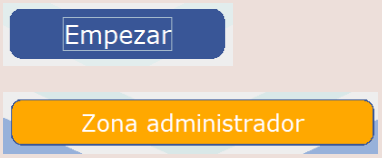
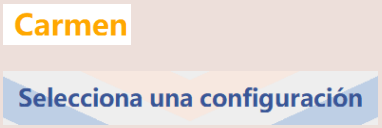
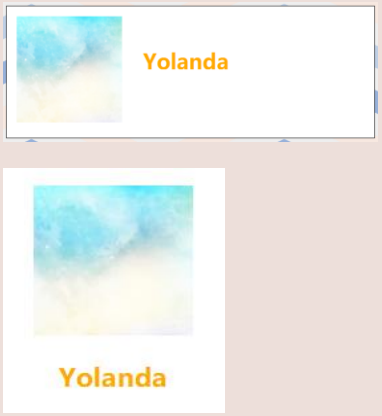
9.4. Adaptación a la pantalla.

La aplicación ha sido desarrollada de tal manera que puede ser adaptada a diferentes resoluciones de pantalla. A la hora de diseñar los paneles, el posicionamiento no se puede realizar de manera absoluta. La solución es el posicionamiento de los elementos según porcentajes del tamaño de la pantalla, posicionamiento relativo. Por ejemplo, puede colocarse un botón en un localización "x = 20% de la anchura de la pantalla" y "y = 30% de la altitud de la pantalla. Para ello, gracias al método de Java "Toolkit.getDefaultToolkit().getScreenSize();" se obtiene un objeto Dimension con la dimensión de la pantalla del usuario. En la figura 9.3.2.-7 se visualizan ejemplos de adaptación a la pantalla en dos resoluciones diferentes: 1920x1080 y 800x600.

Figura 9.3.2.-7: Ejemplo de adaptación a diferentes pantallas con resolución resolución 800x600 (imágenes de la izquierda) y 1920x1080 (de la derecha).

9.5. Elementos gráficos adicionales

En el subcapítulo anterior, se han definido y mostrado los diferentes paneles del juego. En las figuras se han podido ver elementos gráficos que no se han mencionado, por ejemplo, el panel con la fotografía de perfil y nombre del usuario (un panel por cada usuario) es el panel para seleccionar usuario. En este subcapítulo, se describen brevemente las diferentes clases creadas y utilizadas como elementos gráficos adicionales de la aplicación. Estas clases son hijas de una clase de la librería Swing o son paneles que contienen información para el usuario. Estas clases se utilizan desde diferentes paneles y, también, sirve para establecer una estandarización de algunos elementos, por ejemplo, los botones utilizados en la interfaz. Algunos de ellos tienen diferentes constructores, por lo que según cómo se declaren pueden visualizarse de una manera o de otra. A continuación se muestran estos elementos:

Clase Boton	Según la llamada al constructor, puede variar su tamaño y color.	
Clase Etiqueta	Según la llamada al constructor puede variar su tamaño y color.	
Clase PanelNombreUsuario	Contiene una imagen y una etiqueta (clase Etiqueta). Puede variar su estilo y tamaño.	

Cuadro 9.5.: Componentes gráficos creados

9.6. Adaptación de las imágenes

Como se ha podido observar, en la aplicación se visualiza un gran número de imágenes.Estas imágenes son objetos BufferedImage y tienen que ser mostradas en paneles de diferentes tamaños y proporciones (con ancho=alto, ancho < alto o ancho > alto), sin perder su proporción original, es decir, sin sufrir deformaciones. Para ello, se ha creado una clase adicional llamada Foto, la cual contiene un método para realizar esa redimensión sin perder su proporción, ver figura 9. 6. Se le pasa por parámetro la imagen a redimensionar y el ancho y alto máximos el cual ha de tener la imagen.

```
public static BufferedImage resizeAdaptado(BufferedImage img, int maxWidth, int maxHeight) {

    // Alto img < Ancho img
    if (img.getHeight() < img.getWidth()) {
        float ratio = img.getWidth() / (float) img.getHeight();
        img = resize(img, maxWidth, (int) (maxWidth / (float) ratio));

    // Alto img > Ancho img
    } else if (img.getHeight() > img.getWidth()) {
        float ratio = img.getHeight() / (float) img.getWidth();
        img = resize(img, (int) (maxHeight / (float) ratio), maxHeight);

    // Alto img = Ancho img
    } else if (img.getHeight() == img.getWidth()) {

        // Ancho max > Alto max
        if (maxWidth > maxHeight) {
            float ratio = img.getHeight() / (float) img.getWidth();
            img = resize(img, (int) (maxHeight / (float) ratio), maxHeight);

        // Ancho max < Alto max
        } else if (maxWidth < maxHeight) {
            float ratio = img.getWidth()/ (float) img.getHeight();
            img = resize(img, maxWidth, (int) (maxWidth / (float) ratio));

        // Ancho max = Alto max
        } else {
            float ratio = img.getHeight() / (float) img.getWidth();
            img = resize(img, maxWidth, maxHeight);
        }
    }
    return img;
}
```

Figura 9.6.: Código de redimensión de imagen

En este capítulo se han visto los diferentes paneles, su visualización según el flujo de ejecución del programa. En el siguiente se describe cómo se controla y realiza internamente la ejecución de la partida.

10. Control de la partida.

En este capítulo, se describe cómo se realiza el control de los datos durante la ejecución de la partida y, también, durante su preparación. Estas funcionalidades forman parte del Controlador (componente del MVC).

10.1. Preparación de la partida.

En primer lugar, se ha de diferenciar la preparación de la partida según el modo escogido por el usuario (modo Juego o modo Entrenamiento). En el modo Juego el usuario elige una dificultad, a partir de ella, el controlador crea una configuración cuya información cambia según la dificultad (ver figura 10.1.-1):

- Imagen: El tipo de tarjeta es Imagen.
- Medio: El tipo de tarjeta es imagen-sonido.
- Difícil: El tipo de tarjeta es sonido.

Para las tres dificultades anteriores, el número de filas y columnas es el mismo, 3 filas x 4 columnas.

En el modo Entrenamiento, la configuración no se ha de crear ya que el usuario selecciona una en concreto.

```
Configuracion configuracion = new Configuracion();

configuracion.setFilas(3);
configuracion.setColumnas(4);

switch (dificultad) {
    // Imagen-imagen
    case 1:
        configuracion.setTipo(TipoTarjeta.Imagen);
        break;
    // Imagen-sonido
    case 2:
        configuracion.setTipo(TipoTarjeta.Ambos);
        break;
    // Sonido-sonido
    case 3:
        configuracion.setTipo(TipoTarjeta.Sonido);
        break;
}

datos.setConfiguracionJuego(configuracion);
datos.setConfiguracionEntreno(null);
```

Figura 10.1.-1: Código de creación de configuración en la preparación de la partida (modo Juego).

A partir de este punto, para ambos modos se realizan los mismos tratamientos. En primer lugar, se llama al método del modelo que obtiene una lista de tarjetas a partir de la configuración. Este método ya ha sido explicado en el subcapítulo 8.3. Si la lista obtenida no contiene elementos significa que no se han encontrado tarjetas suficientes para la configuración. Si es así, desde la interfaz se muestra un aviso al usuario y finaliza la preparación de la partida. Si no es así, se preparan las

tarjetas de juego (cada cuadrícula del panel de juego contendrá datos (imagen, sonido, texto...) correspondiente a una tarjeta de juego. Por lo tanto, hay una por cada campo de la cuadrícula.

La preparación de las tarjetas de juego consiste en desglosar cada tarjeta de la lista de tarjetas (obtenida en la llamada al modelo) en dos tarjetas de juego (ver figura 10.1-2). Las tarjetas de juego que se vayan creando, se añaden a una lista de tarjetas de juego. Según el tipo de tarjeta (atributo de la configuración) se guardan en los pares de tarjetas los datos necesarios. Por ejemplo, si el tipo de tarjeta es "Ambos" se ha de asignar a una tarjeta de juego una imagen y a la otra un audio. Cada tarjeta de juego también ha de guardar una referencia a la tarjeta asociada, ya que, durante la partida, se ha de comprobar si el par de tarjetas de juego seleccionadas por el usuario están asociadas. Una vez que se han creado todas las tarjetas, se ordenan de forma aleatoria gracias a *Collections.shuffle()*.

```
private ArrayList<TarjetaJuego> prepararTarjetasJuego(TipoTarjeta tipo, ArrayList<Tarjeta> tarjetas) {
    ArrayList<TarjetaJuego> tarjetasJuego = new ArrayList<>();

    for (int i = 0; i < tarjetas.size(); i++) {
        TarjetaJuego tarjetaJuego1 = new TarjetaJuego();
        TarjetaJuego tarjetaJuego2 = new TarjetaJuego();

        tarjetaJuego1.setPalabra(tarjetas.get(i).getPalabra());
        tarjetaJuego2.setPalabra(tarjetas.get(i).getPalabra());

        if (tipo == TipoTarjeta.Ambos) {
            tarjetaJuego1.setImagen(tarjetas.get(i).getImagen());
            tarjetaJuego2.setPathAudio(tarjetas.get(i).getPathSonido());

        } else if (tipo == TipoTarjeta.Imagen) {
            tarjetaJuego1.setImagen(tarjetas.get(i).getImagen());
            tarjetaJuego2.setImagen(tarjetas.get(i).getImagen());
        } else {
            tarjetaJuego1.setPathAudio(tarjetas.get(i).getPathSonido());
            tarjetaJuego2.setPathAudio(tarjetas.get(i).getPathSonido());
        }
        tarjetaJuego1.setTarjetaAsociada(tarjetaJuego2);
        tarjetaJuego2.setTarjetaAsociada(tarjetaJuego1);

        tarjetasJuego.add(tarjetaJuego1);
        tarjetasJuego.add(tarjetaJuego2);
    }

    // Ordenamos las tarjetas de manera aleatoria.
    Collections.shuffle(tarjetasJuego);

    return tarjetasJuego;
}
```

Figura 10.1-2: Código de preparación de las tarjetas de juego.

Una vez que se crea la lista con las tarjetas de juego, se ha de declarar un objeto Juego, el cual tiene los datos necesarios para la ejecución de la partida: la lista de tarjetas de juego, número de filas y de columnas y la configuración y modo de juego, ya que al finalizar la partida el usuario podría volver a jugar una nueva partida utilizando la misma dificultad (modo Juego) o configuración (modo Entrenamiento). También tiene como atributo el número de pares asociados, inicialmente 0, que permite que el usuario pueda visualizar el porcentaje de aciertos de la partida.

10.2. Ejecución de la partida.

Cada vez que el usuario selecciona una tarjeta de juego, la vista llama a un método del Controlador el cual "levanta" la tarjeta, es decir, al objeto Juego se le asigna la tarjeta levantada. El Juego tiene dos atributos adicionales a los comentados en el anterior subcapítulo: "tarjetaLevantada1" y "tarjetaLevantada2". Si no tiene asignada la primera, se asigna y el usuario puede volver a seleccionar otra tarjeta. Si es la segunda seleccionada, se asigna la tarjeta levantada al juego y se comprueba si están asociadas. En la figura 10.2. se muestra el código perteneciente a este proceso.

```
@Override
public boolean levantarTarjeta(TarjetaJuego tarjetaJuego) {
    // Es la primera tarjeta que se levanta
    if (datos.getJuego().getTarjetaLevantada1() == null) {
        datos.getJuego().setTarjetaLevantada1(tarjetaJuego);

        return true;

    } else if (datos.getJuego().getTarjetaLevantada2() == null) {
        datos.getJuego().setTarjetaLevantada2(tarjetaJuego);

        if (datos.getJuego().getTarjetaLevantada1().getTarjetaAsociada().equals(datos.getJuego().
                getTarjetaLevantada2())) {
            datos.getJuego().incrementarParesAcertados();
            datos.getJuego().getTarjetaLevantada1().levantarTarjeta();
            datos.getJuego().getTarjetaLevantada2().levantarTarjeta();
            datos.getJuego().getTarjetaLevantada1().getPanel().repaint();
            datos.getJuego().getTarjetaLevantada2().getPanel().repaint();
            datos.getJuego().setTarjetaLevantada1(null);
            datos.getJuego().setTarjetaLevantada2(null);

            return true;
        }
    }

    return false;
}
```

Figura 10.2.: Código para comprobar la asociación de tarjetas de juego.

Si las tarjetas están asociadas, se incrementa el número de pares asociados del Juego. En los dos objetos TarjetaJuego correspondientes, se marca la tarjeta como levantada. Finalmente, a los dos atributos de tarjetas levantada del Juego se les asigna valor nulo para que el usuario pueda seleccionar otro par de tarjetas de juego. Si las tarjetas no están asociadas, el método devuelve valor falso y desde la vista se da una espera de 3 segundos (para que el usuario pueda ver o oír el contenido de las tarjetas de juego) una vez pasado este tiempo, se vuelve a voltear el par de tarjetas y se repintan.

La partida sigue este proceso, seleccionando pares de tarjetas, hasta que todas las tarjetas queden volteadas finalizando así la partida.

En el siguiente capítulo se realizará un resumen del contenido del TFG, se definirá un trabajo futuro y unas conclusiones finales.

11. Conclusiones y trabajo futuro

En este TFG se ha presentado una aplicación interactiva que ayuda a entrenar la identificación y discriminación auditiva en niños con discapacidad auditiva. La mejora en las habilidades auditivas puede ayudar a incrementar el bienestar de estos niños.

Se han logrado cumplir los objetivos del TFG. Se ha desarrollado una aplicación de escritorio para su uso en centros como el de ASPAS, que se adapta a la resolución de la pantalla, configurable de manera que se adapta a las necesidades del usuario, y que mediante el juego serio presenta actividades educativas de manera entretenida y motivadora.

El trabajo futuro de esta aplicación incluye la implementación de nuevas funcionalidades en torno a la gestión del juego. Por ejemplo, la aplicación de filtros para visualizar las listas de usuarios, configuraciones y tarjetas, de manera que el usuario pueda encontrar los elementos de las listas según los atributos que elija.

Por otra parte, otra funcionalidad que podría ser añadida es la de que gestionar las configuraciones asociadas a las tres dificultades en el modo Juego. De esta manera, el administrador puede modificar también la configuración asociada a este modo según le convenga.

Por último, es primordial realizar pruebas con los usuarios finales ya que en este TFG quedaban fuera de su alcance. Ahora bien, se cuenta con un primer prototipo para evaluar con usuarios finales y que está implementado de forma estructurada para facilitar las futuras iteraciones sobre el prototipo.

11.1. Conclusiones personales.

Gracias al proceso de creación ***+memory*** y al proyecto *Diseño de experiencias interactivas dirigidas al bienestar de personas con necesidades especiales* me he dado cuenta de que el diseño y desarrollo de sistemas interactivos puede ser muy beneficioso para las personas con necesidades especiales. También es de gran importancia la multidisciplinaridad en los equipos, hay que establecer relaciones entre profesionales del contexto del sistema (logopedas, en este caso) y los desarrolladores para poder crear sistemas con los cuales se pueden obtener muchos beneficios para el usuario (educacionales, salud, etc).

Por otra parte, en este trabajo he utilizado muchos conocimientos adquiridos a lo largo de la carrera, por ejemplo, de las asignaturas de bases de datos (Base de Datos I y Base de Datos II) a la hora de tratar los datos, de la asignatura de Aplicaciones Distribuidas en Internet e Interfaces de Usuario ya que en la aplicación se ha desarrollado para ser usable, intuitiva y se ha seguido la metodología de Diseño Centrado en el Usuario, explicada en esta asignatura, también la asignatura de Algoritmos Avanzados en aspectos de uso de arquitectura MVC y de creación de documentación. Todas estas entre muchas otras. También me han servido de ayuda los conocimientos adquiridos tanto en las prácticas externas en empresa como en el mundo laboral, a la hora de programar generando código limpio y de manera organizada.

Anexos

A.1. Fuentes.

- Iconos: www.flaticon.com
- Imagen de fondo: www.freepik.es
- Imágenes y audios de las tarjetas: www.arasaac.org
- Clase JBackgroundPanel: Guille Rodriguez Gonzalez - www.driverlandia.com

A.2. Consultas de creación de las tablas de la base de datos.

Usuarios	CREATE TABLE usuarios (id_usuario INT(3) AUTO_INCREMENT, nombre CHAR(30) NOT NULL, foto LONGBLOB, fecha_creacion TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP, fecha_modificacion TIMESTAMP NOT NULL ON UPDATE CURRENT_TIMESTAMP, eliminado TINYINT(1) DEFAULT 0, PRIMARY KEY (id_usuario))
Fonemas	CREATE TABLE fonemas (id_fonema INT(3) AUTO_INCREMENT, texto_fonema CHAR(10) NOT NULL, fecha_creacion TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP, fecha_modificacion TIMESTAMP NOT NULL ON UPDATE CURRENT_TIMESTAMP, eliminado TINYINT(1) DEFAULT 0, PRIMARY KEY (id_fonema))
Campos_semánticos	CREATE TABLE campos_semanticos (id_campo_semantico INT(3) AUTO_INCREMENT, texto_campo_semantico CHAR(20) NOT NULL, fecha_creacion TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,

	fecha_modificacion TIMESTAMP NOT NULL ON UPDATE CURRENT_TIMESTAMP, eliminado TINYINT(1) DEFAULT 0, PRIMARY KEY (id_campo_semantico))
Configuraciones	CREATE TABLE configuraciones (id_configuracion INT(3) AUTO_INCREMENT, nombre CHAR(30) NOT NULL, filas int(2) NOT NULL, columnas int(2) NOT NULL, tipo enum('Sonido', 'imagen', 'Ambos') NOT NULL, campo_semantico INT(3), fecha_creacion TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP, fecha_modificacion TIMESTAMP NOT NULL ON UPDATE CURRENT_TIMESTAMP, eliminado TINYINT(1) DEFAULT 0, PRIMARY KEY (id_configuracion), FOREIGN KEY (fonema) REFERENCES fonemas(id_fonema), FOREIGN KEY (campo_semantico) REFERENCES campos_semanticos(id_campo_semantico))
Tarjetas	CREATE TABLE tarjetas (id_tarjeta INT(5) AUTO_INCREMENT, palabra CHAR(30) NOT NULL, imagen LONGBLOB, path_sonido CHAR(35), campo_semantico INT(3), fecha_creacion TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP, fecha_modificacion TIMESTAMP NOT NULL ON UPDATE CURRENT_TIMESTAMP, eliminado TINYINT(1) DEFAULT 0,

	PRIMARY KEY (id_tarjeta), FOREIGN KEY (fonema) REFERENCES fonemas(id_fonema), FOREIGN KEY (campo_semantico) REFERENCES campos_semanticos(id_campo_semantico))
Fonemas_configuraciones	CREATE TABLE fonemas_configuraciones (id_fonema_config INT (3) AUTO_INCREMENT, fonema INT(3) NOT NULL, configuracion INT(3) NOT NULL, posicion ENUM('Inicio', 'Medio', 'Fin'), PRIMARY KEY (id_fonema_config), FOREIGN KEY (fonema) REFERENCES fonemas(id_fonema), FOREIGN KEY (configuracion) REFERENCES configuraciones(id_configuracion))
Fonemas_tarjetas	CREATE TABLE fonemas_tarjetas (id_fonema_tarjeta INT(3) AUTO_INCREMENT, fonema INT NOT NULL, tarjeta INT NOT NULL, posicion ENUM('inicio', 'medio', 'fin'), PRIMARY KEY (id_fonema_tarjeta), FOREIGN KEY (fonema) REFERENCES fonemas(id_fonema), FOREIGN KEY (tarjeta) REFERENCES tarjetas(id_tarjeta))

Referencias

[1] Hettler, B. Six Dimensions of Wellness Model (1976), National Wellness Institute, Inc., NationalWellness.org

[2] Universitat de les Illes Balears - Disseny d'experiències interactives dirigides al benestar de persones amb necessitats especials (2016). [online]

http://cooperacio.uib.cat/Projectes-CUD/projectes-cooperacio/convocatoria/2016-convocatoria_13/colombia_experiencies_interactives/. Último acceso: Junio 2018

[3] Manresa-Yee, C., Mas-Sansó, R. & Cano,S. (2018). Juego serio para entrenar habilidades auditivas en niños con discapacidad auditiva. Revista Colombiana de Computación, 19(1), 56-68.

[4] Organización Mundial de la Salud. Sordera y pérdida de la audición. [online] http://www.who.int/es/news-room/fact-sheets/detail/deafness-and-hearing-loss. Último acceso: Junio 2018.

[5] Audifon. ¿Qué es un audífono? [online] https://www.audifon.es/glosario-audifon-audifono. Último acceso: Junio 2018

[6] Federación AICE. ¿Qué es un implante coclear? [online]

http://implantecoclear.org/index.php?option=com_content&view=article&id=76&Itemid=82

[7] Mi hijo sordo. ¿Qué diferencia hay entre un audífono y un implante coclear? [online] http://www.mihijosordo.org/diferencia-audifono-implante.php. Último acceso: Junio 2018

[8] Asociación Aelfa-if. Logopedia. [online] http://www.aelfa.org/logopedia.asp. Último acceso: Junio 2018

[9] Asociación Aspas. Entrenamiento auditivo. [online] http://www.aspasleehablacomunica.com/fases-del-entrenamiento-auditivo/. Último acceso: Junio 2018

[10] Fullana Rivera, N. (2006) Age-related effects on the acquisition of a foreign language phonology in a formal setting. Appendix A, Universitat de Barcelona

[11] Agung, K.B., Purdy, S.C. Kitamura, C. (2005) The Ling Sound Test Revisited," Aust. New Zeal. J. Audiol., vol. 27, no. 1, pp. 33-41.

[12] Dörner, R., Göbel, S., Effelsberg, W., Wiemeyer, J. (2016). Chapter 1. Introduction. En: Serious games. Foundations, concepts and practice. Springer

[13] Centro de Comunicación y Pedagogía. Juego serio: gamificación y aprendizaje. [online] http://www.centrocp.com/juego-serio-gamificacion-aprendizaje/. Último acceso: Junio 2018

[14] Yi, C., Kim, T. (2015) Serious Game Design for Auditory Training of Hearing Impaired Children, *TechArt J. Arts Imaging Sci.*, vol. 2, no. 1, pp. 46–51

[15] Pérez Arévalo , C. Póster - Desarrollo de la percepción del ritmo en niños con discapacidad auditiva a través de juegos serios. [online]

http://cooperacio.uib.cat/digitalAssets/439/439491_poster_percepcion_ritmo.pdf. Último acceso: Junio 2018

[16] Jouhtimäki, J., Kitunen, S., Plaisted, M., Rainò, P. (2009) The Brave Little Troll – A Rhythmic Game for Deaf and Hard of Hearing Children, Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '16., p. 60558

[17] International Organization for Standardization, (2010) ISO 9241-210: Ergonomics of human–system interaction - Human-centred design for interactive systems

[18] NoSoloUsabilidad. Diseño Centrado en el Usuario (DCU). [online] http://www.nosolousabilidad.com/manual/3.htm. Última modificación: 2016. Último acceso: Junio 2018

[19] Java. [online] https://www.java.com/es/download/. Último acceso: Junio 2018

[20] Netbeans. [online] https://netbeans.org/. Último acceso: Junio 2018

[21] Oracle. Swing. [online] https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html. Último acceso: Junio 2018

[22] Oracle. Awt.[online] https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html

[23] Oracle. Applet. [online] http://www.oracle.com/technetwork/java/applets-137637.html. Último acceso: Junio 2018

[24] MySQL. [online] https://www.mysql.com/. Último acceso: Junio 2018

[25] ApacheFriends. [online] https://www.apachefriends.org/es/index.html. Último acceso: Junio 2018

[26] Oracle. JDBC. [online] http://www.oracle.com/technetwork/java/javase/jdbc/index.html. Último acceso: Junio 2018