GRADO EN INGENIERÔA INFORMĆTICA

¢w ! . ! Wh CLb 59 Dw ! 5h

НЛМС К НЛМТ

TĉTULO DEL TF

Juegos y Herramientas para Logopedas

nombre del documento

Memoria

DATOS DE LA ALUMNA O DEL ALUMNO		**DATOS DEL DIRECTOR O DE LA DIRECTORA**	
bh a . w9	AITOR	bh a . w9 м	a ! l59w
! t9[[l5h{	V£LEZ TRISTĆN	! t9[[l5h{ м	AZANZA SESE
5bl	прунпрол/	bh a . w9 н	MIKEL
		! t9[[l5h{ н	VILLAMA¤ E GIRON£S
		59t ! w¢ ! a 9b¢h	LENGUAJES Y SISTEMAS INFORMĆTICOS
FDO.:		FDO.:	
FECHA:		FECHA:	

Resumen

Este documento corresponde a la memoria del Proyecto de Fin de Carrera de Aitor Vélez, desarrollado para la titulación de Grado en Ingeniería Informática de Gestión y Sistemas de Información en la Escuela de Ingeniería de Bilbao de la UPV/EHU.

Este Proyecto de Fin de Carrera está enfocado a facilitar el día a día del gabinete de logopedia *Hitz-Kiribil*. Para ello, se ha realizado el diseño e implementación de un videojuego para dispositivos *Android*, y un programa de gestión para gestionar su información y gestionar los pacientes de una logopeda.

Por todo esto, el sistema estará compuesto por una aplicación *Android* y un programa de escritorio realizado mediante *Java*. En lo referente a su implementación, se ha utilizado el motor gráfico *Unity3D*, y librerías emergentes como *MyBatis*. Por otro lado, para el almacenamiento de datos, se ha utilizado una base de datos *MySQL* para los datos en remoto, y *SQLite* para los datos en local (en el dispositivo *Android*).

Índice

Índice de Figuras

Índice de Tablas

Introducción

La tecnología es un campo que en los últimos años ha avanzado a grandes escalones. En menos de 20 años, ha cambiado totalmente el mundo que conocían nuestros antepasados. De esta forma, nos encontramos en un mundo en el que, miremos donde miremos, la gran mayoría de los objetos están informatizados, creando una sociedad tecnológicamente dependiente. Debido a esto, todos los campos de nuestra vida cotidiana se han tenido que actualizar, desde la medicina, creando grandes avances en la búsqueda de medicamentos; hasta en las "tiendas del barrio", creando un sistema de ventas por Internet.

De esta forma, como dice el refrán, quien no avanza, retrocede. Por esto, en este punto entra en juego el gabinete de logopedia *Hitz-Kiribil*. Este gabinete, especializado en el tratamiento de niños, se ha encontrado con dos grandes problemas: gestionar la información de cada niño y cada sesión, y que los niños no solo hagan actividades en la consulta, si no que sigan trabajando en casa los ejercicios. Entonces, gracias a las nuevas tecnologías, ambas cosas se podrían controlar de una forma sencilla para la logopeda.

Por otro lado, hoy en día los videojuegos están tomando un papel fundamental, "habiendo movido 90.000 millones de dólares el pasado año 2015" (Fraga, 2016). Además, visto el interés sobre estos, se ha creado una categoría especial para los videojuegos con finalidad educativa, los *Serious Games.* Estos videojuegos no solo entretienen a quienes los juega, sino que también hacen que los jugadores aprendan sin darse cuenta de ello. (Joan Morales Moras, 2016)

Por todo esto, se pretende crear un programa de gestión para la logopeda, y una aplicación para los niños que acuden a su gabinete. Para esto, el proyecto se dividirá en dos partes. Por un lado, está el programa de gestión de los datos propios de los niños y administración de actividades para los mismos, junto con las sesiones que realice en la consulta. Por otro lado, la aplicación que utilizarán los niños para poder practicar fuera de la consulta. Dicha aplicación debe ser atractiva y divertida, debido a que debe motivar a los niños para a que trabajen los ejercicios mandados por la logopeda en sus ratos libres, por lo que esta segunda parte del proyecto va a estar encaminada a realizar un *Serious Game*.

Llegados a este punto, habría que preguntarse cuál será el argumento del videojuego. Echando la vista atrás en la historia de los videojuegos, se puede observar que los juegos que más éxito han tenido han sido los juegos de niveles y los de cuidar una mascota virtual (TodoTech, s.f.). El objetivo del primer tipo de videojuego consiste en ir completando niveles uno a uno, mientras se avanza por un escenario. En el caso del segundo tipo de videojuego, su objetivo es el de cuidar una mascota, dándole todos los cuidados necesarios (limpiar, dar de comer, pasear, etc.). Un ejemplo claro de los primeros son los videojuegos de la empresa *King*[1], como son *Candy Crush Saga*, *Candy Crush Soda Saga,* etc. En el caso del segundo tipo estaría el *Pou*, un juego estilo al conocido *tamagotchi*. Por otro lado, ambos tipos de juegos podrían ser infinitos. De esta forma, no se podrá terminar el juego en ningún momento.

[1] https://king.com/es

Después de cotejar e investigar sobre ambos tipos, y estimar el coste gráfico necesario, lo mejor sería hacer un videojuego de niveles. Esta decisión ha sido tomada debido al gran coste de diseño gráfico necesario para hacer un videojuego sobre una mascota virtual, dado que habría que modelar y diseñar figuras en 3D. Además, el escenario de los niveles podrá ser introducido por la logopeda en caso de querer aumentar el repertorio de mapas (imágenes de los mapas y la posición, tamaño y número de botones en cada uno). Asimismo, la logopeda podrá introducir premios. De esta manera, los niños tendrán el objetivo de intentar conseguir y descubrir (si no se ha obtenido, aparecerá como una sombra) todos los premios existentes, y, a su vez, obtener sus objetos preferidos.

Aun con todo esto, puede surgir fácilmente la duda de qué tiene esto de nuevo respecto a lo que ya está inventado, o qué les va a atraer a jugarlo en vez de, por ejemplo, el ya mencionado *Candy Crush*. La respuesta es bastante sencilla. Cualquier persona que esté haciendo algo que le guste, estará todo el tiempo que pueda pegado a ello. Sabiendo esto, un niño podrá estar jugando horas sin darse ni si quiera cuenta que le están examinando, o incluso más importante, que está aprendiendo. Todo esto sin percatarse de que está realizando estos ejercicios que no le gusta hacer, sino que está intentando superar los niveles. Respecto a la segunda cuestión entraría la labor de los padres, de utilizar su autoridad para decidir cuándo pueden y no pueden jugar a cualquier cosa. De esta forma, los niños creerán que, a pesar de que no les dejan jugar a la hora de ir a la consulta, o a la hora de estudiar, tienen permiso de utilizar este videojuego, consiguiendo ese objetivo suyo de conseguir esos logros u objetos que desean en todo momento.

Estudio de Antecedentes

Los *Serious Games*, nombre acuñado por Clark Abt, se refieren a "aquellos juegos o videojuegos que no son concebidos con la diversión como fin único, sino con objetivos formativos o educativos muy concretos. Podríamos decir que la diversión no es su razón de ser, sino un camino hacia el aprendizaje - Aprender jugando." (Wonnova, 2016)

En este tipo de minijuegos, el error juega un gran papel. Debido a esto, el jugador nunca perderá o arriesgará nada. Esto ocurre porque en el aprendizaje es necesario equivocarse. Por este motivo, se quiere demostrar que mediante el método ensayo-error, el usuario aprenda. (Sánchez Gómez, 2008)

A pesar de que este tipo de juego lleva años existiendo, es ahora cuando más se está hablando de este tipo de aprendizaje. Por esto, muchas empresas han tenido que llevar su método de enseñanza a los videojuegos. Un ejemplo de esto son las empresas Santillana y Cuadernos Rubio. Ambas empresas han tenido que adaptar sus fichas y cuadernos a un formato digital, con el cual los niños puedan aprender de una forma más amena y desde un *Smartphone* o *Tablet* (con la aplicación *Pupitre* en el primer caso e *iCuadernos* en el segundo*)*, sin tener que llevar las fichas o cuadernos mencionados a todas partes (Gestionet, 2016).

Por otro lado, también existen juegos enfocados a necesidades especiales. El juego *Dyseggxia*, por ejemplo, está dedicado a ayudar a superar problemas de lectura y escritura en castellano a personas con dislexia. En el caso del juego *Sigueme*, su objetivo es el de potenciar la atención visual y entrenar la adquisición del significado (asociación de imágenes a palabras y su significado) en personas con autismo. Un tercer ejemplo sería el juego *Azahar*. Este juego tiene el propósito de dar la posibilidad a personas con autismo y/o discapacidad intelectual de mejorar su comunicación, la planificación de sus tareas y disfrutar de sus actividades de ocio. (Gestionet, 2015)

Como se puede apreciar, existen una gran variedad de alternativas, pero cada una de ellas especializada en un trastorno distinto, dependiendo de lo que se pretenda conseguir. Aun así, para abarcar los distintos trastornos a los que se enfrentan en el gabinete de logopedia, se necesitarían más de una de estas aplicaciones, convirtiéndose en una tarea un tanto molesta para la logopeda por tener todo centralizado. Además, seguiría existiendo el problema de que los ejercicios solo se puedan realizar delante de la logopeda, de forma que la logopeda pueda saber, todo momento, los resultados obtenidos por el niño, y, a su vez, impidiendo realizar un seguimiento fuera de la consulta.

En nuestro caso, el videojuego captará varios tipos de juegos, cada uno enfocado a un trastorno distinto. De esta forma, la logopeda, con una única aplicación podrá tanto gestionar el proceso del niño, como administrar los minijuegos que necesita el niño realizar para mejorar o superar el trastorno.

Objetivos

El objetivo principal del proyecto consiste en dar la posibilidad de que los niños hagan ejercicios de forma amena, y que los resultados los pueda obtener la logopeda. Para esto se diseñará una aplicación (app) ejecutable en dispositivos portables.

Además, se pretende dar al logopeda la posibilidad de obtener un *feedback* continuo de la actividad de los niños. Junto con este *feedback*, se pretende hacer disponible a la logopeda de un historial completo y digital de toda actividad, digital y física, del niño. De esta forma, se conseguirá una mejor y más sencilla gestión de la información por parte de la responsable. Este segundo objetivo se cumplirá mediante la creación de un programa de escritorio.

Por último, para conseguir la máxima personalización posible, se creará un programa de gestión con el programa de escritorio anteriormente mencionado. Gracias a este nuevo programa, se podrá gestionar desde los ejercicios que podrá jugar cada niño, hasta las imágenes propias del juego.

Por tanto, resumiendo, este proyecto tendrá tres objetivos principales:

- Creación de una aplicación para la realización de actividades desde lugares distintos a la consulta de la logopeda.
- Creación de un programa de gestión intuitivo y fácil de utilizar.
- Creación de un programa para gestionar el historial de sesiones (físicas y virtuales) de los diferentes niños.

Análisis de herramientas y alternativas

1. Tecnologías

La primera decisión que tiene que tomar un programador antes de hacer nada es la de decidir sobre qué Entorno de Desarrollo Integrado (IDE) o motor gráfico trabajar. A pesar del gran abanico de posibilidades, el hecho de que la aplicación deba poder jugarse, a poder ser en cualquier lugar, nos elimina un gran porcentaje de posibilidades. Por esto, nos queda decidir entre los diferentes IDEs o motores gráficos con los que se puede desarrollar para dispositivos móviles.

1.1. Motores Gráficos y Entornos de Desarrollo

El término de Motor Gráfico ha tenido diferentes significados a lo largo de la historia. En sus comienzos, se conoció como una arquitectura que separaba el *software* del arte, el mundo virtual y las reglas, para dar así a la creación de juegos de disparos en primera persona (*First Person Shooter* - FPS) como *Doom*, creado por la empresa *Id Software*[2]. Más tarde, gracias a los desarrolladores, se creó la llamada "comunidad de *mods*" y los famosos *mods* para modificar el juego, añadiendo funcionalidades o nuevas características mediante *scripting*. Gracias a esto, se definió el término motor gráfico como el *software* que es extensible y puede ser utilizado como la base de diferentes juegos sin muchas modificaciones, siendo la arquitectura de programación dirigida por datos lo que diferencia los motores gráficos de cualquier otro trozo de *software* que puede ser un juego, pero no un motor gráfico. (Gregory, 2014)

De esta forma, se fueron creando diversos motores gráficos, pero cada uno especializado en un género de juego concreto, siendo a día de hoy imposible la existencia de un motor gráfico especializado en todo tipo de géneros. Varios ejemplos de estos, que a su vez son los más utilizados hoy en día, son los siguientes:

1.1.1. Unity3D

Historia

Unity 3D es un motor gráfico, acompañado de un IDE bajo el mismo nombre. Este motor fue creado por tres estudiantes para trabajar en sus proyectos de videojuegos pequeños desde *Mac*. Tras crearse la empresa bajo el nombre de *Unity Technologies*, y el lanzamiento de la *App Store* de *iPhone*, Unity 3D se convirtió en un programa necesario para entrar en este mercado, ya que fue el único motor gráfico que tenía este soporte. Por este hecho y por el gran alcance de funcionalidades que podía dar, muchas empresas de videojuegos como *Ubisoft* o *Electronic Arts* comenzaron a utilizar Unity 3D para realizar sus propios videojuegos (Brodkin, 2013).

Tras numerosas versiones y nuevas herramientas (*MonoDevelop*, *Unity Asset*, etc.), Unity 3D ha conseguido llegar a definir el motor como una plataforma de desarrollo de extremo, flexible y de alto rendimiento, utilizándose para crear intensas experiencias interactivas *2D*, *3D*, *VR* y *AR*. De esta forma, Unity 3D y su editor con funciones completas, sirven de base para desarrollar hermosos juegos o aplicaciones, y trasladarlos fácilmente a múltiples plataformas: dispositivos móviles, sistemas de entretenimiento en casa, computadoras personales y sistemas incrustados (Unity Technologies, 2001).

[2] http://www.idsoftware.com/?/age_gate

Características

Al ser un motor gráfico multiplataforma, Unity3D nos ofrece la posibilidad de la creación de videojuegos en múltiples plataformas, y, sobre todo, en las plataformas móviles que nos interesan (Windows Phone, iOS y Android) de forma gratuita. A pesar de alguna diferencia respecto a los controladores, la variación de la plataforma solo influye en un clic a la hora de generar el proyecto.

A nivel gráfico, a pesar de que no nos interese, Unity nos permite utilizar una gran cantidad de características 3D, como el soporte de sombras en tiempo real, la profundidad del campo y la posibilidad de crear e importar escenarios abiertos totalmente editables y manipulables.

Seguido del apartado gráfico tocaría centrarse en las animaciones. En este apartado, Unity3D permite el control de dichas animaciones mediante jerarquías y transiciones sofisticadas del Estado de Máquina (conjunto de estados y transiciones que tiene un objeto, y eventos necesarios para cambiar entre ellos) y la invocación de eventos desde dentro de la reproducción entre algunas de sus características. Estas animaciones, junto con toda la jugabilidad que se implemente, se puede probar gracias a su apartado de simulación, donde podremos "jugar" a nuestra creación.

Referente al apartado de interacción entre los objetos y la física, Unity3D utiliza el motor *NVIDIA PhysX 3.3*, implementando a su vez físicas para diferentes tipos de cuerpos (sólidos, blandos, vestimentas, etc.). Por otro lado, gracias a la biblioteca *Box2D*, se dispondrá con una gama completa de efectores (herramientas para añadir fuerzas de atracción y repulsión), articulaciones y c*olliders* (componente encargado de definir la forma de un objeto). En el apartado del sonido nos ofrece la posibilidad de escuchar el ambiente desde el propio entorno de desarrollo, además de jerarquías de mezcladores, instantáneas, efectos predefinidos y sonido 3D.

En el campo de la programación de la lógica del juego (*scripting*), Unity3D nos permite elegir entre tres lenguajes de programación diferentes: *Javascript*, *C#* y *Boo* (dialecto de *Python*), todos ellos basados en la Programación Orientada a Objetos (POO). Además, Unity3D se integra con la plataforma *MonoDevelop*, permitiéndonos usarla para programar sin consumir tanta memoria como otros editores. Aun así, también existe la integración nativa con *Visual Studio* desde la versión 4.0.

En lo relativo al coste económico de Unity3D, existen cuatro tipos de licencia diferentes. De esta forma, el desarrollador se podrá adaptar a la licencia que más le convenga, dependiendo del proyecto en marcha y de la situación económica:

- Personal: La versión gratuita si no se superan los $100 mil por ejercicio fiscal (obligado comprar la siguiente versión en caso de superarse), pero a su vez, la más limitada. Aun así, permite la correcta utilización de la mayoría de las funcionalidades.
- Plus: Esta versión puede ser obtenida por $35/mes, y añade la posibilidad de desarrollar para realidad virtual, la monetización y la creación de pantallas de inicio personalizadas. En esta versión, el tope de ingresos por ejercicio fiscal es de $200 mil. Al igual que con la anterior, en caso de superarse ese límite el usuario se verá obligado a comprar la siguiente licencia.
- Pro: La versión más completa, pero a su vez más cara se puede obtener por $125/mes. Esta versión nos trae todas las funcionalidades de Unity3D y un tope de ingresos por ejercicio fiscal ilimitados.
- Enterprise: Versión dedicada a la necesidad de utilización del motor gráfico para un negocio. Esta versión no tiene un precio ni características fijas debido a que serían las acordadas entre la empresa contratante y *Unity Technologies*.

Por último, pero no por ello menos importante, Unity3D posee un gran apartado de documentación en su propia página, la cual, junto con la ayuda de la amplia comunidad que lo sostiene, permite la fácil solución de dudas y aprendizaje. Además, gracias al *Asset Store*, existe la posibilidad de importar *assets* (contenedor de prefabs (archivo que contiene cualquier creación, textura, script, configuración, etc.)) creados por otros diseñadores o programadores (de pago y gratuitas) a nuestro proyecto para propia utilización.

1.1.2. Unreal Engine

Historia

Tras el éxito del juego *Unreal,* en 1998 *Epic Mega* Games creó y sacó a la venta el motor gráfico *Unreal* con el objetivo de dar grandes herramientas a artistas y diseñadores. Con el paso del tiempo y la mejora de las tecnologías, Unreal Engine fue mejorado en diferentes aspectos, hasta crear el primer videojuego con el propio motor *Unreal Tournament*. Un año más tarde, en 1999, la empresa se cambió el nombre pasando a ser *Epic Games*. Tras la llegada de las diversas consolas y dispositivos (*PlayStation, Xbox, Android,* etc.), y la conversión del motor gráfico en un motor de físicas, crearon la posibilidad de diseñar videojuegos más "reales". De esta forma, a pesar de ser sólo de pago, *Unreal Engine* atrajo a grandes empresas de videojuegos y animación como *Activision*, *Ubisoft* y *Disney* logrando la fama actual.

El último paso de su historia lo logró en el año 2015. Además de la mejora gráfica y la optimización de la compilación de los scripts a tiempo real, esta última versión trajo consigo la posibilidad de utilizar el motor gráfico de forma gratuita. (Bleszinski, 2010)

Características

Al igual que *Unity3D*, *Unreal Engine* permite la creación de videojuegos en multiplataforma, entre la cuales se encuentra las móviles que estamos interesados (*Windows Phone*, *iOS* y *Android)*.

Unreal Engine ha logrado un gran acabado gráfico gracias a *DirectX 12* y al post-procesado de los efectos, dando una apariencia como si de una película se tratara. Estos efectos han sido tan valorados por los diseñadores, que no solo los diseñadores de videojuegos utilizan este motor, sino que también diseñadores de animaciones y películas animadas. Todas estas características se pueden probar en tiempo real gracias a su ventana de *gameplay* o simulación, la cual permitirá ejecutar y probar el juego o el proyecto realizado en cualquier momento.

En lo referente a las físicas *Unreal Engine 4* utiliza el motor de físicas *PhysX 3.3* para dirigir sus cálculos de simulaciones de físicas y realizar todos los cálculos de colisiones.

En la sección del sonido se nos permite la modificación y ejecución de los mismos en tiempo, junto con las características necesarias para combinarlos.

Otra de las partes fuertes de *Unreal Engine* es la relacionada con la programación. El lenguaje que se utiliza es *C++*, el cual es el mismo con el que está hecho el motor. Este lenguaje tiene la gran característica de ser un lenguaje muy potente y la posibilidad de ver las funciones realizadas en tiempo real. Esto último quiere decir que no tenemos que estar creando nuestra aplicación cada vez que hagamos una modificación, sino que se actualizará en el momento de guardar. Además, debido a que el motor gráfico es *open source*, se podrán editar las herramientas del propio motor gráfico, añadiendo nuevas características o adaptándolas a nuestras necesidades.

En lo relativo al coste económico, *Unreal Engine* ha sido siempre un motor gráfico de pago hasta la llegada de *Unreal Engine 4*, donde se introdujo una versión gratuita, pero con algunas limitaciones:

- Free Licence: La versión gratuita está dedicada al estudiante, desarrolladores individuales o pequeñas empresas. El soporte que se obtiene es únicamente el que pueda dar la comunidad, y si se gana más de $3000 en un cuatrimestre, *Epic Games* se llevará el 5% de las ganancias totales.
- Custom Licence: La versión de pago está más dedicada a las empresas grandes que necesitan soporte y organización en sus proyectos. Ese soporte y organización se negociará con la empresa a la hora de contratar este servicio, al igual que el precio que se pagará.

Por último, *Unreal Engine* posee un gran apartado de documentación en su propia página junto con trozos de *scripts* en su *GitHub*. Todo esto, se suma a los constantes cambios y evoluciones que tiene el motor gráfico gracias a los desarrollos y aportaciones externas. (Epic Games, 2004)

1.1.3. Android Studio

Historia

Android Studio fue anunciado en el año 2013 en la conferencia de Google I/O. Fue creado para reemplazar a Eclipse, la plataforma que los desarrolladores utilizaban para crear aplicaciones, tanto de escritorio, como de *Android*. De esta manera, con Android Studio, Google consiguió su propio IDE para el desarrollo de aplicaciones, pudiendo instalar todo el kit de desarrollo de Software (Software Development Kit - SDK) para desarrollar apps específicas adaptadas a la mayor parte de versiones. (Europa Press, 2013)

Características

A diferencia de los motores gráficos, *Android Studio* es una *IDE* para desarrollar aplicaciones para *Android* únicamente.

En lo referente a los gráficos, *Android Studio* tiene la posibilidad de crear *layouts* (Disposición de los elementos) por cada ventana, al igual que plantillas ya creadas. A su vez, tiene la posibilidad de posicionar los objetos que queramos introducir en nuestras ventanas mediante un movimiento de arrastre de los mismos.

El lenguaje de programación utilizado en *Android Studio* es *Java*, junto con la posibilidad de uso de código externo en C++ gracias a la *NDK (*Kit de desarrollo Nativo). Por otro lado, *Android Studio* lleva incorporado un sistema de compilación flexible basado en *Gradle (*Herramienta de automatización de la construcción del código). Gracias a estas dos características, *Android Studio* se convierte en una *IDE* potente para el diseño de aplicaciones *Android*.

En el tema de la parte económica, *Android Studio* es completamente gratuito en todas sus versiones, independientemente de las ganancias obtenidas. (Google, 2016)

1.1. **Elección**

Para obtener una elección más clara, hemos construido una tabla con las principales diferencias más importantes. (Ver Tabla 1)

	Unity 3D	Unreal Engine	Android Studio
Gratuito	X	X	X
Multi-plataforma	X	X	
Lenguaje	C#/Bloo/Javascript	C++	Java
Complejidad	Normal	Alta	Normal

Tabla 1: Diferencias IDE

Basándonos en las características propias de cada *IDE* o motor gráfico, el siguiente paso consiste en tomar una decisión. Para comenzar por algún lado, a pesar de que en un principio la aplicación va a estar dedicada a un tipo de dispositivo, *Android Studio* se podría descartar para dar la posibilidad de ampliar el repertorio de dispositivos.

Por otro lado, en referencia a los dos motores gráficos, *Unreal Engine* está más dedicado a diseño y videojuegos 3D que necesitan de una gran potencia. Por eso la utilización de C++ en la programación de los mismos. Unity3D, en cambio, está más enfocado a juegos más simples, siendo a su vez más fácil la utilización y aprendizaje. En el tema económico y de licencias, nos daría igual qué motor elegir debido a que no vamos a monetizar el videojuego, pudiendo utilizar de esta forma ambas versiones gratuitas. Por todo, esto, y porque el videojuego será un juego 2D sin necesidad de utilizar gráficos y físicas exageradas, la *IDE* elegida para la realización del videojuego es *Unity3D*, la cual, como hemos mencionado anteriormente, es utilizada por el motor gráfico del mismo nombre. Además, entre los lenguajes de programación permitidos, se va a utilizar *C#* debido a que es el lenguaje más sencillo de los tres, además de ser el más parecido a Java (lenguaje de programación utilizado durante la carrera).

Además, para la realización del programa de gestión se utilizará *Eclipse*. Debido a que ha sido la *IDE* utilizada durante la carrera para programar en *Java*, y que ya estamos adaptados a ella.

2. Plataformas

Llegados a este punto, sabemos que para realizar el videojuego vamos a utilizar el motor gráfico *Unity 3D*. Por otro lado, conocemos este motor, y que, con el mismo, se puede realizar videojuegos para los tres tipos de plataformas móviles existentes (*Android*, *iOS*, *Windows Phone*), por lo que podría surgir la duda de por qué elegir entre ellas. La respuesta a esta pregunta es debido a que no sirve de nada crear aplicaciones que no se van a utilizar. Esta afirmación viene porque, como se puede apreciar en la Figura 1, *Android* está presente en el 81.7% de los *smartphones* en el mercado. Por esto, en un principio vamos a realizar una aplicación compatible con los dispositivos *Android*, aunque también abiertos a crear la aplicación disponible también para los demás sistemas operativos móviles si existiera la necesidad. (Statista, Inc., 2016)

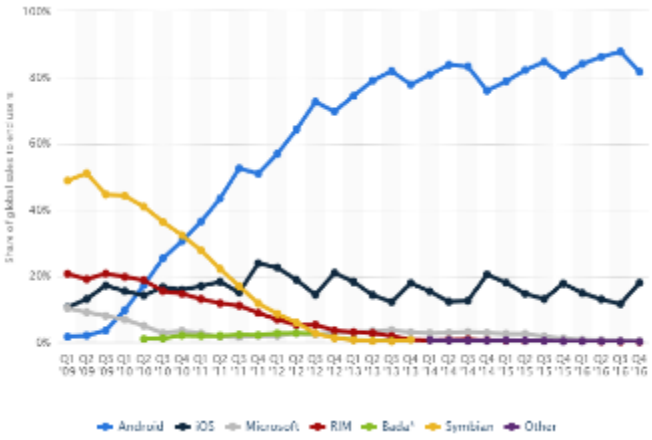


Figura 1: Porcentaje de SO móviles mundialmente

Alcance

1. Fases

El proyecto se va a dividir en 7 fases o apartados (ver *Figura 2*), las cuales más tarde se desglosarán y explicará más detenidamente. Estas fases se desarrollarán mediante un ciclo de vida en cascada. El motivo de elegir este tipo de ciclo de vida es porque se ha considerado que es el modo más sencillo, sobre todo para desarrollar un programa individualmente. Esta sencillez se ha detectado debido a que este tipo de ciclo funciona paso a paso, es decir, para hacer una fase, hay que haber hecho todas las anteriores De esta forma, estas 7 fases serán las siguientes:

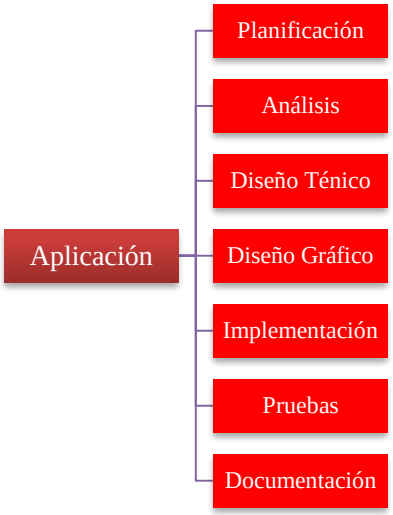


Figura 2: Esquema Aplicación

- Planificación y Captura de Requisitos: Se acordará con los tutores de proyecto, y estos a su vez con la logopeda, las necesidades y posibilidades que existen para realizar el proyecto y sus diferentes características.
- Análisis y Diseño: Se investigará y decidirá cuál es la mejor plataforma para la cual desarrollar la aplicación y se indagará sobre las distintas herramientas disponibles para su realización. A su vez, se estudiarán las diferentes leyes que repercuten en este proyecto, relacionadas con la protección de datos en especial.
- Diseño Técnico: Se procederá al apartado técnico de la aplicación, donde se planteará la estructura que llevará el proyecto para conseguir todos, o la gran mayoría de los objetivos o metas, y a su vez cumpliendo con lo acordado y decidido en las dos anteriores fases (Planificación y Análisis).
- Diseño Gráfico: Se diseñará el apartado gráfico de la aplicación para que sea lo más visual y atractiva posible de cara a los niños.
- Implementación: Se implantará lo diseñado y acordado en los anteriores apartados. Esta parte se podría considerar como la “columna vertebral del proyecto” debido a que es donde se le dará forma.
- Pruebas: Se realizarán las pruebas pertinentes una vez implementado y diseñado todos los apartados (técnicos y gráficos).
- Documentación: Se llevará a cabo la documentación de la memoria del proyecto, mediante la obtención tanto de la información pre-realización, el diseño y las experiencias obtenida tras realizarlo. A su vez, se preparará la defensa del mismo.

2. Tareas

A continuación, observaremos cada fase individualmente.

2.1. Planificación

La primera parte del proyecto estará dividida en dos partes principales (Ver Figura 3).

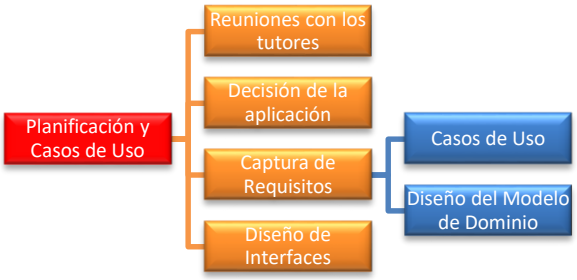


Figura 3: Esquema Planificación

2.1.1. Reuniones con los tutores

- Duración: 2h
- **Descripción:** Varias reuniones con los tutores para consensuar el tipo de aplicación que se hará y alguna de las especificaciones que tendrá (dadas por la logopeda).
- Salidas/Entregables: -
- Recursos necesarios: Papel y bolígrafo

2.1.2. Decisión de la aplicación

- Duración: 1h
- **Descripción:** Decisión del tipo de aplicación que se realizará partiendo de lo hablado con los tutores.
- **Salidas/Entregables:** Tipo de aplicación a realizar
- Recursos necesarios: -.

2.1.3. Captura de Requisitos

- **Duración**: 20h
- **Descripción:** Análisis de los casos de uso y diseño del modelo de dominio (de aplicación y programa de gestión).
- **Salidas/Entregables**: Casos de Uso y Modelo de Dominio.
- **Recursos necesarios**: Herramientas para la realización de diagramas (*Visual Paradigm*).

2.1.4. Diseño de interfaces

- **Duración**: 10h
- **Descripción:** Diseño completo de las interfaces (de aplicación y programa de gestión).
- **Salidas/Entregables**: Diseño de interfaces
- **Recursos necesarios**: Papel y lápiz.

2.2. **Análisis**

En el apartado de análisis, podemos distinguir cinco grandes apartados, los cuáles, algunos de ellos se pueden dividir en partes más pequeñas para una mejor organización (Ver Figura 4)

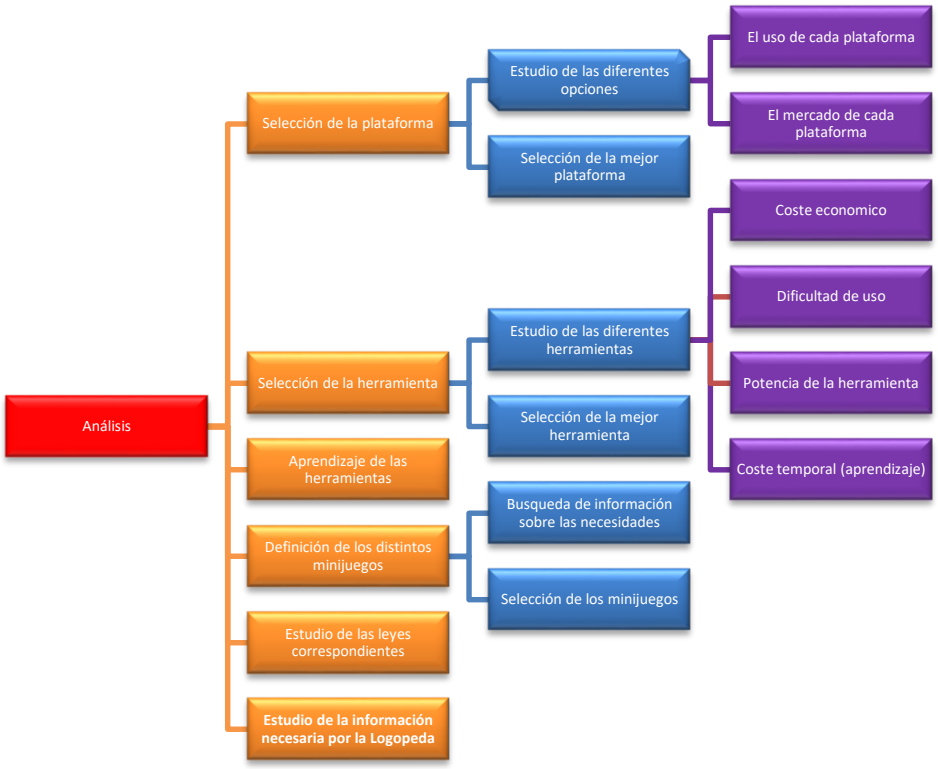


Figura 4: Esquema Análisis

2.2.1. Selección de la plataforma

- Duración: 3h
- **Descripción:** Estudio y selección de la plataforma sobre la que realizar la aplicación entre las diferentes plataformas posibles según su mercado en la población joven y según el porcentaje de utilización global.
- **Salidas/Entregables:** Plataforma en la que realizar.
- **Recursos necesarios:** Ordenador con Internet.

2.2.2. Selección de la herramienta

- Duración: 3h
- **Descripción:** Estudio y selección de entre un gran número de herramientas existentes en el mercado (gratuitas y de pago) teniendo en cuenta el coste económico, la dificultad de la herramienta, la potencia de la herramienta y el coste temporal en aprender a utilizarla o a perfeccionar los conocimientos para la realización del proyecto.
- **Salidas/Entregables:** Herramientas a utilizar.
- **Recursos necesarios:** Ordenador con Internet.

2.2.3. Aprendizaje de la herramienta

- Duración: 12h
- **Descripción:** Obtención o mejora del conocimiento sobre la utilización de las distintas herramientas que se van a utilizar.
- **Salidas/Entregables:** Conocimiento de las herramientas.
- **Recursos necesarios:** Ordenador con Internet, libros de guías, herramientas a aprender.

2.2.4. Estudio de la información necesaria por la Logopeda

- Duración: 4h
- **Descripción:** Investigación de los diferentes documentos sobre educación especial y la información que se necesita sobre un paciente.
- **Salidas/Entregables:** Listado de características necesarias de un paciente.
- **Recursos necesarios:** Ordenador con Internet.

2.2.5. Definición de los distintos minijuegos

- Duración: 12h
- **Descripción:** Estudio de diferentes tipos de ejercicios relacionados con la logopedia y selección del máximo número de ellos que puedan ser realizados mediante un videojuego.
- **Salidas/Entregables:** Listado de minijuegos.
- **Recursos necesarios:** Ordenador con Internet, libros de ejercicios.

2.2.6. Estudio de las leyes correspondientes

- Duración: 6h
- **Descripción:** Estudio de las diferentes leyes que tienen relación con la Protección de Datos.
- **Salidas/Entregables:** Listado de minijuegos.
- **Recursos necesarios:** Ordenador con Internet.

2.3. **Diseño Técnico**

El tercer apartado, encargado del diseño técnico, se divide en tres apartados importantes. Estos, al igual que en el análisis, se dividen en varios subapartados para gestionar mejor cada sección (Ver Figura 5).

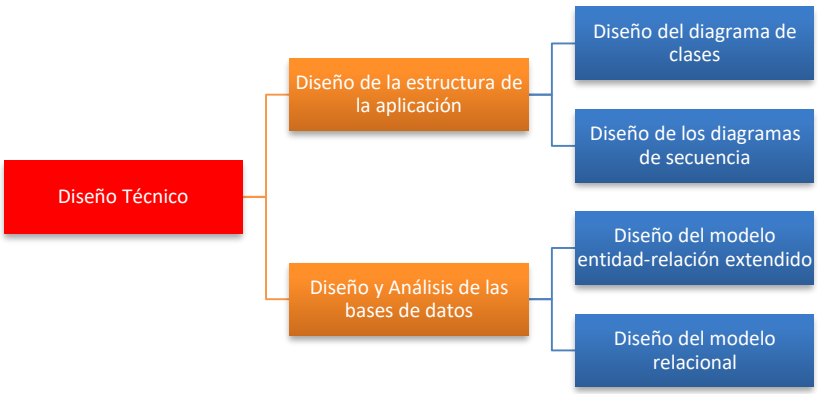


Figura 5: Esquema Diseño Técnico

2.3.1. Diseño de la estructura de la aplicación

- **Duración**: 40h
- **Descripción:** Diseño de diagramas, es decir, el diseño de los casos de uso, el diseño del modelo de dominio, el diseño del diagrama de clases y el diseño de los diagramas de secuencia.
- **Salidas/Entregables:** Diagramas, casos de uso y modelo de dominio.
- **Recursos necesarios:** Herramientas para la realización de diagramas (*Visual Paradigm*)

2.3.2. Diseño y Análisis de las bases de datos

- Duración: 25h
- **Descripción:** Diseño y análisis completo de las bases de datos, tanto el diseño del modelo entidad-relación extendido, como el diseño del modelo relacional.
- **Salidas/Entregables:** Diagramas de las bases de datos.
- **Recursos necesarios:** Herramientas para la realización de diagramas. (*VisualParadigm*, *Dia*).

2.4. **Diseño Gráfico**

El apartado encargado del diseño gráfico se divide en dos partes para una mejor gestión individual de cada una (Ver Figura 6).

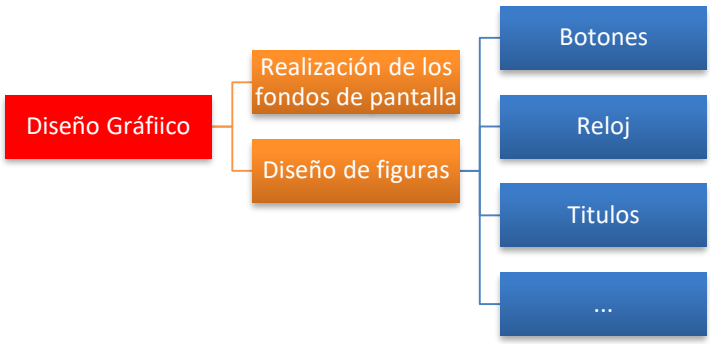


Figura 6: Esquema Diseño Gráfico

2.4.1. Realización de los fondos de pantalla

- **Duración**: 15h
- **Descripción:** Realización gráficamente de todos los fondos que se utilizarán en la aplicación.
- **Salidas/Entregables:** Fondos de la aplicación.
- **Recursos necesarios:** Herramientas de edición gráfica (*Photoshop*)

2.4.2. Realización de figuras

- **Duración**: 15h
- **Descripción:** Realización gráficamente de todos los componentes como botones, reloj, títulos, etc.
- **Salidas/Entregables:** Componentes gráficos restantes.
- **Recursos necesarios:** Herramientas de edición gráfica (*Photoshop*).

2.5. **Implementación**

El apartado principal del proyecto, donde crearemos el juego, se puede dividir en 3 partes, diferenciadas por el campo al que pertenecen (Ver Figura 7).

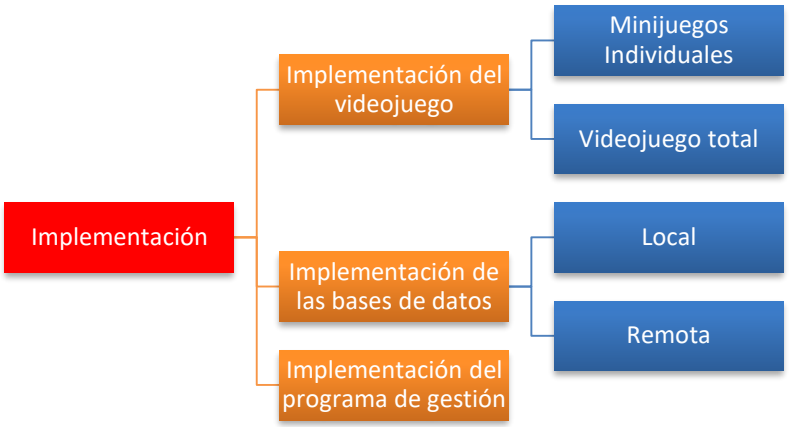


Figura 7: Esquema Implementación

2.5.1. Implementación del videojuego

- Duración: 300h
- **Descripción:** Programación de la aplicación completa, partiendo de los minijuegos, y por último el juego base.
- **Salidas/Entregables:** Videojuego base (sin base de datos).
- **Recursos necesarios:** Ordenador con Internet, herramientas elegidas.

2.5.2. Implementación de las bases de datos

- Duración: 18h
- **Descripción:** Implementación de todas las sentencias *SQL* para la creación de las bases de datos diseñadas en el paso 2.3.3, tanto en local como en remoto.
- **Salidas/Entregables:** Bases de datos remota y local e interacciones.
- **Recursos necesarios:** Ordenador con Internet, herramientas de bases de datos. (*MySQL*, *SQLite*, *Xampp* y *DB Browser*)

2.5.3. Implementación del programa de gestión

- Duración: 60h
- **Descripción:** Implementación del programa de gestión.
- Salidas/Entregables: Programa de gestión.
- **Recursos necesarios:** Ordenador con Internet, herramientas de entornos de java. (*Eclipse*)

2.6. **Pruebas**

Al igual que en la implementación, el apartado de pruebas se puede dividir también en tres apartados (Ver Figura 8).

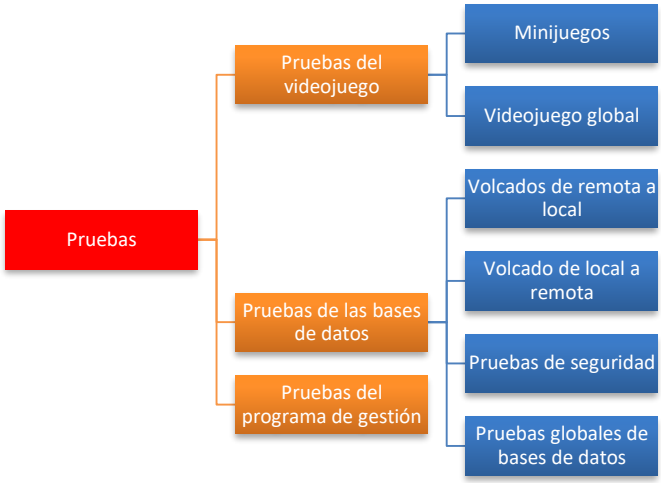


Figura 8: Esquema Pruebas

2.6.1. Pruebas del videojuego

- Duración: 18h
- **Descripción:** Prueba de las variedades que puedan existir dentro de los diferentes minijuegos y del videojuego en su conjunto.
- Salidas/Entregables: Pruebas del videojuego
- **Recursos necesarios:** Ordenador con Internet, herramientas de pruebas y diseño, *Smartphone* o *Tablet Android Studio* (para obtener los erróres).

2.6.2. Pruebas de las bases de datos

- Duración: 9h
- **Descripción:** Prueba de las bases de datos, desde el volcado de datos de una base a la otra y viceversa, hasta las pruebas de seguridad y globales.
- **Salidas/Entregables:** Pruebas de las bases de datos.
- **Recursos necesarios:** Ordenador con Internet, herramientas de bases de datos. (*MySQL*, *SQLite*, *Xampp* y *DB Browser*)

2.6.3. Pruebas del programa de gestión

- Duración: 9h
- **Descripción:** Pruebas del programa de gestión y su interacción con la aplicación
- **Salidas/Entregables:** Pruebas del programa de gestión (*JUnits*).
- **Recursos necesarios:** Ordenador con Internet, herramientas de entornos de java. (*Eclipse*)

2.7. **Documentación**

El último apartado, a pesar de desarrollarse durante todo el proyecto, se puede dividir en tres partes principales (Ver Figura 9).

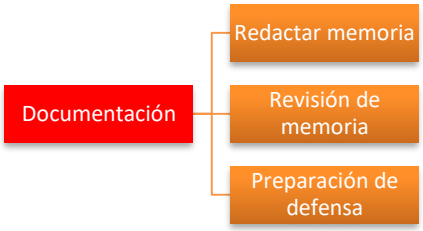


Figura 9: Esquema Documentación

2.7.1. Redactar memoria

- Duración: 200h
- **Descripción:** Redacción de la memoria completa.
- Salidas/Entregables: Memoria.
- **Recursos necesarios:** Ordenador con Internet, herramientas de creación de documentos (*Microsoft Word*).

2.7.2. Revisión de la memoria

- Duración: 8h.
- **Descripción:** Revisión de la memoria y entrega de la misma a los tutores para obtener opiniones y posibles errores.
- Salidas/Entregables: Memoria final
- **Recursos necesarios:** Ordenador con Internet, herramientas de creación de documentos (*Microsoft Word*).

2.7.3. Preparación de la defensa

- Duración: 8h
- **Descripción:** Preparación de la defensa del proyecto de fin de grado.
- **Salidas/Entregables:** Presentación realizada para la defensa.
- **Recursos necesarios:** Ordenador, herramientas de creación de documentos. (*Microsoft Office*).

3. Planificación temporal

Tras decidir las tareas que se van a realizar, hay que centrarse en la planificación temporal que se va a llevar para conseguir realizar el proyecto de la manera más eficiente posible. Para ello, antes de organizar los tiempos, vamos a crear una tabla (Ver Tabla 2) para tener resumidos todos los tiempos.

Tarea	Horas dedicadas (Estimación)
1. Planificación	33 horas
1.1. Reunión con los profesores	2 horas
1.2. Decisión de la aplicación	1 hora
1.3. Captura de Requisitos	20 horas
1.4. Diseño de interfaces	10 horas
2. Análisis	40 horas
2.1. Selección de la plataforma	3 horas
2.2. Selección de la herramienta	3 horas
2.3. Aprendizaje de la herramienta	12 horas
2.4. Definición de los distintos minijuegos	12 horas
2.5. Estudio de las leyes correspondientes	6 horas
2.6. Estudio de la información necesaria por la Logopeda	4 horas
3. Diseño Técnico	65 horas
3.1. Diseño de la estructura de la aplicación	40 horas
3.2. Diseño y análisis de las bases de datos	25 horas
4. Diseño Gráfico	20 horas
4.1. Realización de fondos	10 horas
4.2. Realización de figuras	10 horas
5. Implementación	378 horas
5.1. Implementación del videojuego	300 horas
5.2. Implementación de las bases de datos	18 horas
5.3. Implementación del programa de gestión	60 horas
6. Pruebas	36 horas
6.1. Pruebas del videojuego	18 horas
6.2. Pruebas de las bases de datos	9 horas
6.3. Pruebas del programa de gestión	9 horas
7. Documentación	216 horas
7.1. Redactar memoria	200 horas
7.2. Revisión de memoria	8 horas
7.3. Preparación de defensa	8 horas
TIEMPO TOTAL	**788 horas**

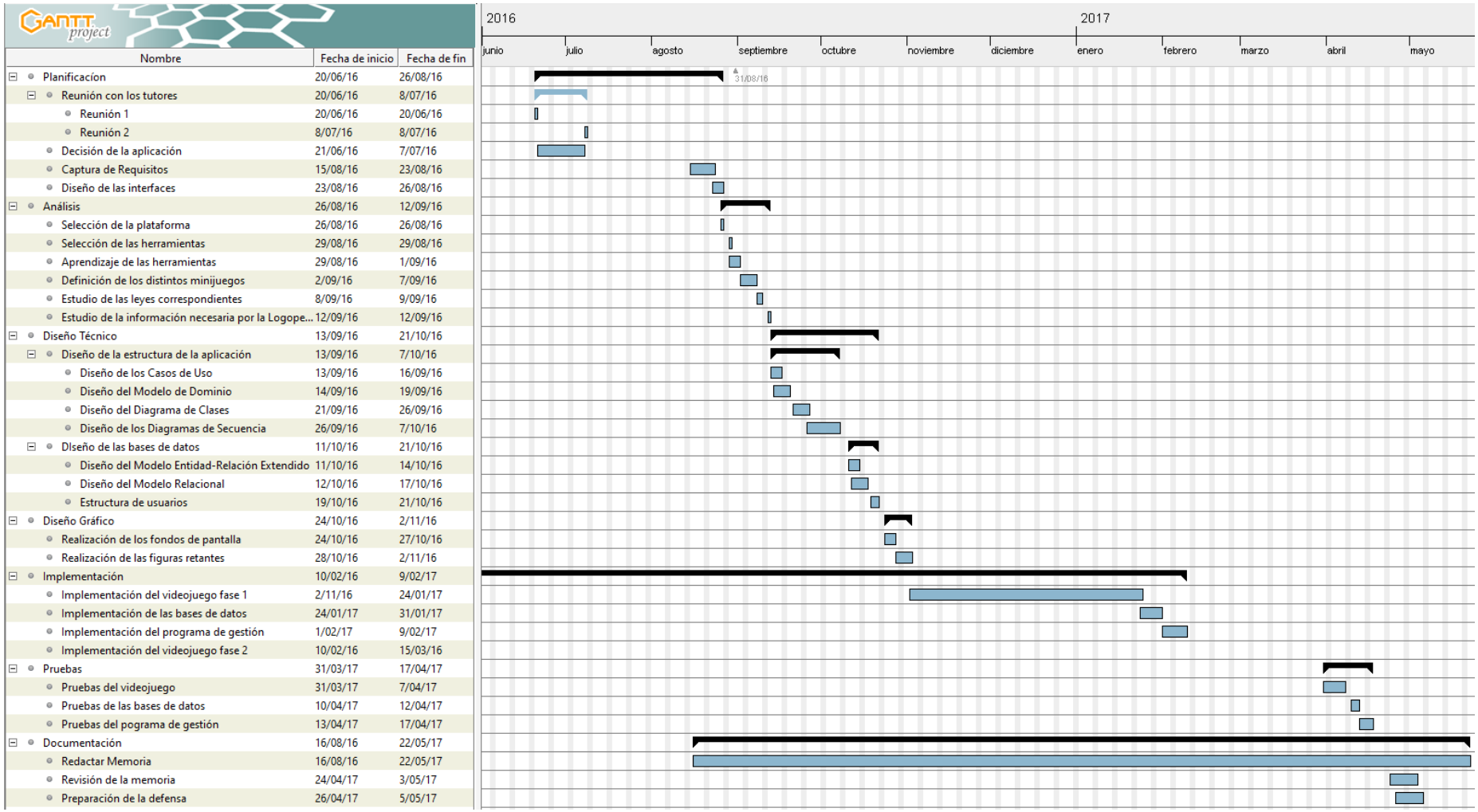
Tabla 2: Planificación temporal de las tareas

Una vez reunidos todos los tiempos, se ha realizado un diagrama de *Gantt* para representar las horas (Ver Figura 10), suponiendo un horario constante de 20 horas semanales de un único trabajador, separadas en 5 días, es decir, 4 horas diarias de lunes a viernes, y fin de semana de descanso. Este descanso se podría omitir en caso de necesitar terminar alguna tarea urgentemente. A pesar de tener constancia de los días festivos que tendría una empresa que realiza este tipo de trabajos, no los hemos tomado como días festivos, introduciéndolos en el diagrama.

Las tareas están repartidas equitativamente, por lo que en los días en los que se hacen más de una tarea, las horas de dicho día se reparten entre las tareas que se realizan. Además, como particularidad en el gráfico, las primeras tareas se realizan con un plazo de tiempo más amplio que el resto del proyecto. Esto es porque las reuniones se realizan antes de verano, para poder tener todo decidido en agosto. Por otro lado, el espacio entre las dos reuniones en el que se decide la aplicación no son horas reales. Esto significa que el tiempo dedicado a esta tarea es mucho menor y sin constancia, pudiendo ser un día una hora y al de una semana otra.

Otro dato a explicar es lo referente a la subtarea de “Redactar Memoria”. Esta subtarea transcurrirá durante todo el tiempo que dura el proyecto en ejecución. Esto quiere decir que la memoria se redactará y modificará a medida que se vaya avanzando, por lo que no tiene un plazo exacto, sino que se realizará durante “casi todo el proyecto”. Esta duración sería de 1h por día en esta tarea.

3.1. **Diagrama de *Gantt***



Nombre	Fecha de inicio	Fecha de fin
Planificacíon	20/06/16	26/08/16
Reunión con los tutores	20/06/16	8/07/16
Reunión 1	20/06/16	20/06/16
Reunión 2	8/07/16	8/07/16
Decisión de la aplicación	21/06/16	7/07/16
Captura de Requisitos	15/08/16	23/08/16
Diseño de las interfaces	23/08/16	26/08/16
Análisis	26/08/16	12/09/16
Selección de la plataforma	26/08/16	26/08/16
Selección de las herramientas	29/08/16	29/08/16
Aprendizaje de las herramientas	29/08/16	1/09/16
Definición de los distintos minijuegos	2/09/16	7/09/16
Estudio de las leyes correspondientes	8/09/16	9/09/16
Estudio de la información necesaria por la Logope...	12/09/16	12/09/16
Diseño Técnico	13/09/16	21/10/16
Diseño de la estructura de la aplicación	13/09/16	7/10/16
Diseño de los Casos de Uso	13/09/16	16/09/16
Diseño del Modelo de Dominio	14/09/16	19/09/16
Diseño del Diagrama de Clases	21/09/16	26/09/16
Diseño de los Diagramas de Secuencia	26/09/16	7/10/16
DIseño de las bases de datos	11/10/16	21/10/16
Diseño del Modelo Entidad-Relación Extendido	11/10/16	14/10/16
Diseño del Modelo Relacional	12/10/16	17/10/16
Estructura de usuarios	19/10/16	21/10/16
Diseño Gráfico	24/10/16	2/11/16
Realización de los fondos de pantalla	24/10/16	27/10/16
Realización de las figuras retantes	28/10/16	2/11/16
Implementación	10/02/16	9/02/17
Implementación del videojuego fase 1	2/11/16	24/01/17
Implementación de las bases de datos	24/01/17	31/01/17
Implementación del programa de gestión	1/02/17	9/02/17
Implementación del videojuego fase 2	10/02/16	15/03/16
Pruebas	31/03/17	17/04/17
Pruebas del videojuego	31/03/17	7/04/17
Pruebas de las bases de datos	10/04/17	12/04/17
Pruebas del pograma de gestión	13/04/17	17/04/17
Documentación	16/08/16	22/05/17
Redactar Memoria	16/08/16	22/05/17
Revisión de la memoria	24/04/17	3/05/17
Preparación de la defensa	26/04/17	5/05/17

Figura 10: Diagrama de Gantt

4. Gestión de Riesgos

En este apartado se comentarán los diferentes riesgos que se han visto como posibles, y su prevención y plan de contingencia.

4.1.1. Planificación Incorrecta

Descripción

Al ser la primera planificación que se realizan sobre un proyecto de una gran magnitud, seguramente todo lo planificado no sea correcto. Por esta razón, a pesar de que se estime un tiempo, su tiempo de dedicación real será bastante superior.

Prevención

La manera de prevenir este riesgo es realizando una planificación con una holgura amplia de tiempo. De esta forma, a pesar de no cumplir con los tiempos tal y como están previstos, la fecha de entrega será cumplida.

Plan de Contingencia

Si en caso de que la prevención no funcione, habrá que aumentar el número de horas según la fecha de entrega se vaya acercando.

Probabilidad

Es muy probable.

Impacto

El impacto será más alto cuanto más cerca esté la fecha de la entrega.

4.1.2. Sufrir fallo en el ordenador de trabajo o un virus

Descripción

Como es un proyecto informático, siempre va a existir la posibilidad de entrada de algún virus o la existencia de un problema informático (disco duro estropeado, por ejemplo). Esto causará una tremenda pérdida en caso de que no se haya realizado ninguna copia de seguridad.

Prevención

La manera de prevenir este riesgo es guardando todas las cosas que se hagan en diferentes servidores. En caso de la memoria y los diagramas, se guardará en *Google Drive* y *Dropbox*. En caso del código, se guardará en *Bitbucket*, y se accederá a él mediante *Git*.

Plan de Contingencia

Si se produjera este fallo, se podría seguir trabajando con el trabajo desde otro ordenador. El único problema podría existir en el código, si este no ha sido subido a la nube.

Probabilidad

Es probable.

Impacto

El impacto será más alto cuanto más cerca esté la fecha de la entrega (también, porque contra más avanzado esté el proyecto, más contenido se habrá desarrollado).

4.1.3. Borrado de la Base de Datos

Descripción

Como el servidor donde está alojada la base de datos no es un servidor propio, existe la posibilidad de borrarse tanto el usuario de acceso como las bases de datos creadas. Esto causaría la repetición de la creación de la base de datos.

Prevención

La única prevención existente frente a este problema, es la creación de los *scripts* de creación de la base de datos. De esta forma, la primera vez se ejecutarán dichos *scripts*, y en caso de error, se volverán a ejecutar (sin tener que reescribir la base de datos).

Plan de Contingencia

A pesar de solucionar el problema, se mantendría un contacto con las personas administradoras de la base de datos, para comentar el problema.

Probabilidad

Es poco probable.

Impacto

El impacto será medio porque la recreación de la base de datos se podría realizar en poco tiempo una vez ya construida la primera vez.

4.1.4. Actualización de Herramientas

Descripción

Como se están utilizando diferentes herramientas, que reciben continuas actualizaciones, podría ocurrir que una de esas actualizaciones provoque una inconsistencia en el código creado hasta el momento (parte del código queda inutilizable).

Prevención

La manera de prevenir este riesgo es dejando de actualizar las herramientas. Es decir, desde el comienzo del proyecto, hasta que finaliza el mismo, las diferentes herramientas que se utilicen no se actualizarán para prevenir cambios de este tipo.

Plan de Contingencia

Si se produjera este problema, se podría continuar con lo trabajado hasta el momento porque no variaría en nada. Una vez terminado, se podría actualizar y comprobar si sigue funcionando (para estar actualizado).

Probabilidad

Es probable.

Impacto

El impacto sería alto debido a que provocaría una gran pérdida de tiempo para corregir las inconsistencias surgidas.

4.1.5. Problemas Personales

Descripción

El hecho de sufrir algún problema personal a lo largo del proyecto.

Prevención

No existe una prevención exacta debido a que depende de qué tipo de problema ha surgido (es diferente romperse un brazo a una fiebre).

Plan de Contingencia

Una vez recuperado, dedicar más horas a las tareas retrasadas.

Probabilidad

Tiene una probabilidad media.

Impacto

El impacto depende del tipo de problema y del tiempo que conlleva recuperarse. De todas formas, este riesgo provoca un retraso en las fechas previstas para la terminación de las diferentes tareas.

5. Evaluación Económica

Tras realizar la planificación temporal, y obtener como resultado el diagrama de *Gantt*, el siguiente paso será el de presentar la evaluación económica del proyecto. Para ello, se ha dividido en varias partes y se calculará el coste económico mediante el cálculo del coste anual de cada apartado (porque el proyecto va a durar cerca de un año si se cuentan vacaciones).

Personal

Para la elaboración del proyecto, se ha necesitado el trabajo de un desarrollador. Buscando por internet, podemos observar que un programador cobra una media de 30 euros/hora, por lo que, si el proyecto va a llevarse a cabo en 788 horas, el coste personal será de 23640 euros. (Cuanto Gana, s.f.)

Hardware

Ordenador: Para programar el proyecto, se necesitará un ordenador de sobremesa valorado en unos 800 euros con una vida media de unos 8 años. Esto ordenador se utilizará durante todo el proyecto para tanto desarrollar el código, como la documentación.

Amortización Anual:

$$800€/8\ años\ = 100€/año$$

Tablet: Se necesita de una Tablet para realizar las diferentes pruebas relacionadas con el videojuego. Se parte de una Tablet de un coste de 250 euros, con una vida media de 3 años, y un uso único por parte del desarrollador.

Amortización Anual:

$$25€/3\ años = 83€/año$$

Software

Licencias de Pago: Licencia de Windows 10, obtenida en su época gratuita (no se contará en el coste final). La licencia del Visual Paradigm tiene un coste de 72 euros/año. A pesar de que se ha utilizado la licencia académica de la universidad, se tiene que tener en cuenta si fuera una empresa. Microsoft Word tiene un coste de 99 euros/años. Al igual que con Visual Paradigm, se ha utilizado la licencia gratuita de la universidad, pero en una empresa se tendría que tener en cuenta.

Licencias Gratuitas: Unity3D, Eclipse, Chrome, Sublime Text3, GanttProyect, etc. A pesar de que se ha utiliza la licencia gratuita de Unity3D, a la hora de vender el proyecto, habría que valorar obtener una licencia de pago. Aun así, en este caso, no se va a contar dicho coste debido a que se podría realizar con su versión gratuita.

Gastos	
Personal	23.640 €
Hardware	183 €
Software	171 €
TOTAL: 23.994 €	

Captura de Requisitos

En este apartado se presenta los casos de uso, con la jerarquía de actores y el modelo de dominio.

1. Casos de uso y jerarquía de actores

A continuación, se van a mostrar la jerarquía de actores (Ver Figura 11) y la representación del modelo de casos de uso (Ver Figura 12, Figura 13, Figura 14 y Figura 15). Junto con la representación de cada diagrama, se van a dar una pequeña descripción de cada actor o caso de uso individualmente.

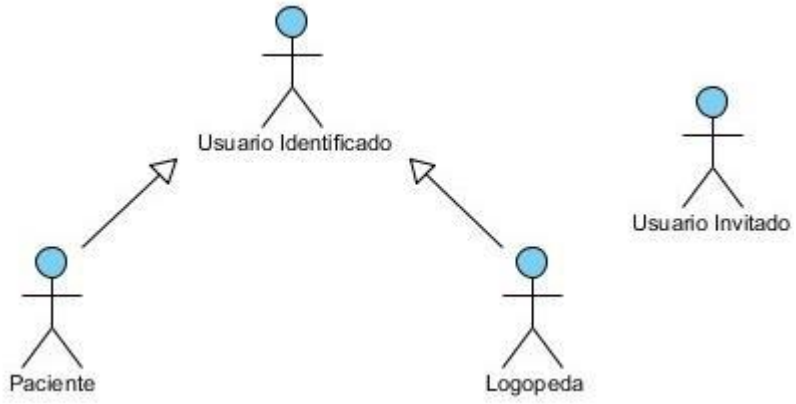


Figura 11: Jerarquía de Actores

Usuario Invitado

Este actor será el encargado de realizar aquellas funcionalidades que se pueden realizar sin haberse identificado.

Usuario Identificado

Este actor será el encargado de realizar aquellas funcionalidades que son comunes para el actor Logopeda y el actor Paciente. Previamente habrá tenido que identificar en el sistema.

Logopeda

Este actor será el que use un usuario identificado como Logopeda. Además, realizará aquellas funcionalidades que puede realizar el actor Usuario Identificado.

Paciente

Este actor será el que use un usuario identificado como Niño/Tutor. Además, realizará aquellas funcionalidades que puede realizar el actor Usuario Identificado.

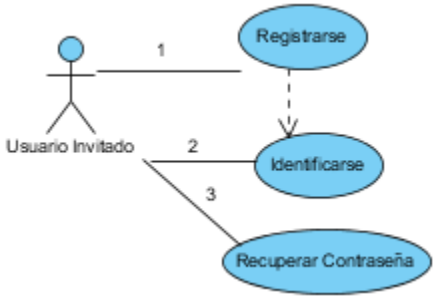


Figura 12: Modelo Casos de Uso Invitado

1. Registrarse

Permite al actor Usuario Invitado registrarse en el sistema como Logopeda, de tal forma que pueda acceder a sus funcionalidades. Una vez registrado, se dirigirá a la identificación.

2. Identificarse

Permite al actor Usuario Invitado identificarse en el programa (como Logopeda o Tutor) y en el videojuego para poder acceder a las funcionalidades de su actor.

3. Recuperar Contraseña

Permite al actor Usuario Invitado recuperar su contraseña.

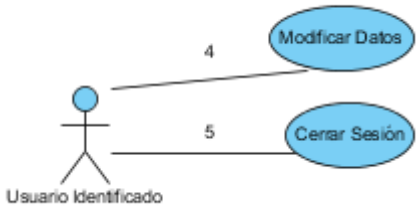


Figura 13: Modelo Casos de Uso Identificado

4. Modificar Datos

Permite al actor Usuario Identificado modificar sus datos personales.

5. Cerrar Sesión

Permite al actor Usuario Identificado cerrar la sesión con la que está conectado.

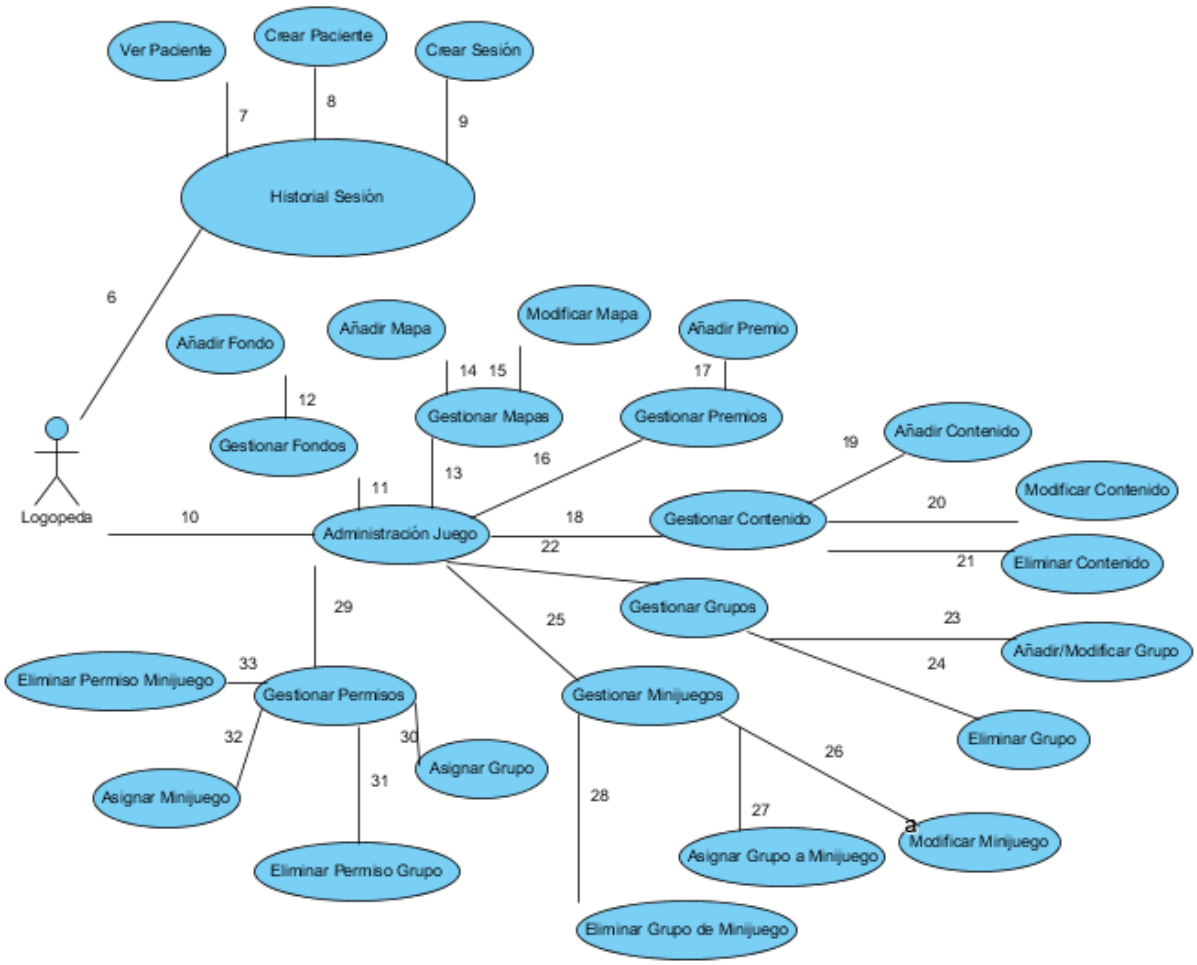


Figura 14: Modelo Casos de Uso Logopeda

6. Historial Sesión

Permite al actor Logopeda acceder a la ventana de la información de los pacientes.

7. Ver Paciente

Permite al actor Logopeda ver la información de un paciente.

8. Crear Paciente

Permite al actor Logopeda crear un nuevo paciente. Para ello tendrá que introducir sus datos.

9. Crear Sesión

Permite al actor Logopeda crear una nueva sesión de un paciente. Para ello se deberá introducir la fecha de la sesión, si la puede ver el tutor, y el comentario.

10. Administrar Juego

Permite al actor Logopeda acceder a la ventana de administración general de los juegos.

11. Gestionar Fondos

Permite al actor Logopeda acceder a la ventana donde se gestionan los fondos.

12. Añadir Fondo

Permite al actor Logopeda añadir un fondo nuevo. Para ello, se deberá seleccionar una imagen e introducir un nombre de fondo

13. Gestionar Mapas

Permite al actor Logopeda acceder a la ventana donde se gestionan los mapas.

14. Añadir Mapa

Permite al actor Logopeda añadir un mapa nuevo y sus botones. Para ello, se deberá seleccionar una imagen e introducir un nombre de mapa

15. Modificar Mapa

Permite al actor Logopeda modificar los botones de un mapa. Se podrá modificar la posición y tamaño de los botones, añadir nuevos y eliminar los existentes.

16. Gestionar Premios

Permite al actor Logopeda acceder a la ventana donde se gestionan los premios.

17. Añadir Premio

Permite al actor Logopeda añadir un premio nuevo. Para ello, se deberá seleccionar una imagen e introducir un nombre de premio

18. Gestionar Contenido

Permite al actor Logopeda acceder a la ventana donde se gestionan el contenido.

19. Añadir Contenido

Permite al actor Logopeda añadir contenido nuevo. Para ello se tendrá que introducir el nombre del contenido, el texto en castellano o euskera. También se puede introducir una imagen.

20. Modificar Contenido

Permite al actor Logopeda modificar un contenido creado.

21. Eliminar Contenido

Permite al actor Logopeda eliminar un contenido creado.

22. Gestionar Grupos

Permite al actor Logopeda acceder a la ventana donde se gestionan los grupos.

23. Añadir/Modificar Grupo

Permite al actor Logopeda añadir un grupo nuevo o añadir contenido a uno existente. Para ello, se deberá introducir un nombre del grupo, y seleccionar el contenido opción y contenido solución entre todo el contenido existente.

24. Eliminar Grupo

Permite al actor Logopeda eliminar un grupo creado.

25. Gestionar Minijuegos

Permite al actor Logopeda acceder a la ventana donde se gestionan los minijuegos.

26. Modificar Minijuego

Permite al actor Logopeda modificar la información y nombre de un minijuego.

27. Asignar Grupo a Minijuego

Permite al actor Logopeda asignar un grupo a un minijuego. Para ello, se deberá seleccionar el grupo que se quiere asignar entre los posibles.

28. Eliminar Grupo de Minijuego

Permite al actor Logopeda eliminar un grupo de un minijuego.

29. Gestionar Permisos

Permite al actor Logopeda acceder a la ventana donde se gestionan los permisos.

30. Asignar Grupo

Permite al actor Logopeda asignar un grupo a un paciente. Para ello, se deberá seleccionar el paciente al que se le quiere asignar el grupo.

31. Eliminar Permiso Grupo

Permite al actor Logopeda eliminar el permiso de un grupo a un paciente.

32. Asignar Minijuego

Permite al actor Logopeda asignar un minijuego a un paciente. Para ello, habrá que seleccionar el paciente al que se le quiere asignar el minijuego, los grupos máximos y mínimos que podrá visualizar y las vidas con las que comenzará.

33. Eliminar Permiso Minijuego

Permite al actor Logopeda eliminar el permiso de un minijuego a un paciente.

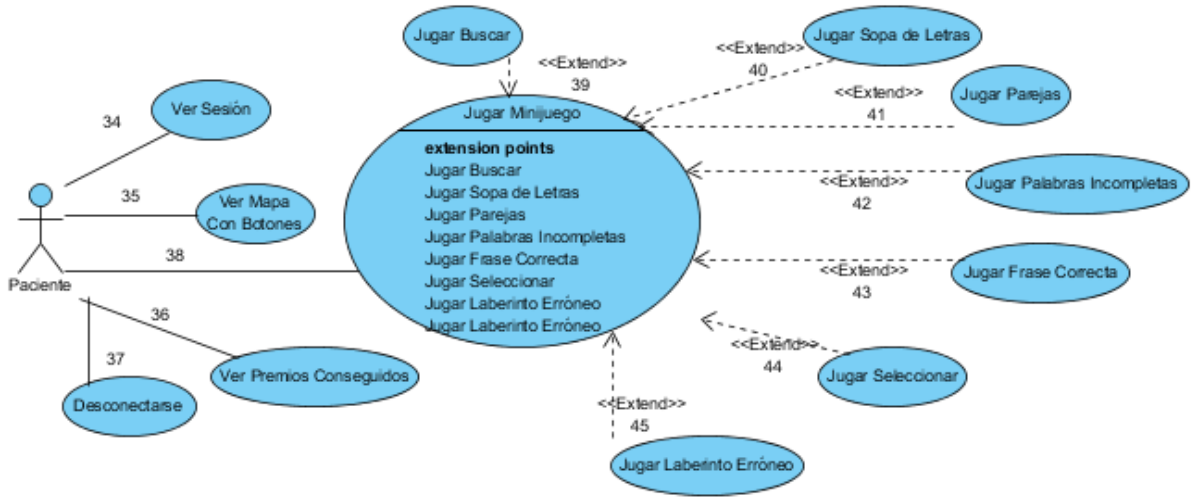


Figura 15: Modelo Casos de Uso Paciente

34. Ver Sesión

Permite al actor Paciente ver las sesiones que la logopeda le ha creado. Para ver alguna sesión, la logopeda deberá haber creado alguna sesión y haberla puesto como visible por el tutor.

35. Ver Mapas Con Botones

Permite al actor Paciente visualizar los mapas de juego, junto con sus botones o niveles (los niveles activados, los que se han completado y los futuros).

36. Ver Premios Conseguidos

Permite al actor Paciente visualizar los premios que ha conseguido. Estos premios se verán con su silueta si los ha conseguido, y la imagen en caso de no haberlo hecho.

37. Desconectarse

Permite al actor Paciente desconectarse del videojuego. Para ello, será necesario estar conectado a Internet.

38. Jugar Minijuego

Permite al actor Paciente poder jugar a cualquier minijuego. Para ello deberá pulsar en cualquier botón activo del mapa, y el minijuego que se juegue dependerá del minijuego que esté asignado.

39. Jugar Buscar

Permite al actor Paciente poder jugar al minijuego *Buscar*. Para ello, la logopeda se lo tendrá que haber asignado, junto con los grupos correspondientes.

40. Jugar Sopa de Letras

Permite al actor Paciente poder jugar al minijuego *Sopa de Letras*. Para ello, la logopeda se lo tendrá que haber asignado, junto con los grupos correspondientes.

41. Jugar Parejas

Permite al actor Paciente poder jugar al minijuego *Parejas*. Para ello, la logopeda se lo tendrá que haber asignado, junto con los grupos correspondientes.

42. Jugar Palabras Incompletas

Permite al actor Paciente poder jugar al minijuego *Palabras Incompletas*. Para ello, la logopeda se lo tendrá que haber asignado, junto con los grupos correspondientes.

43. Jugar Frase Correcta

Permite al actor Paciente poder jugar al minijuego *Frase Correcta*. Para ello, la logopeda se lo tendrá que haber asignado, junto con los grupos correspondientes.

44. Jugar Seleccionar

Permite al actor Paciente poder jugar al minijuego *Seleccionar*. Para ello, la logopeda se lo tendrá que haber asignado, junto con los grupos correspondientes.

45. Jugar Laberinto Erróneo

Permite al actor Paciente poder jugar al minijuego *Laberinto Erroneo*. Para ello, la logopeda se lo tendrá que haber asignado.

2. Modelo de Dominio

A continuación, en la Figura 16 se mostrará el Modelo de Dominio utilizado para organizar los diferentes datos sobre los que se han trabajado

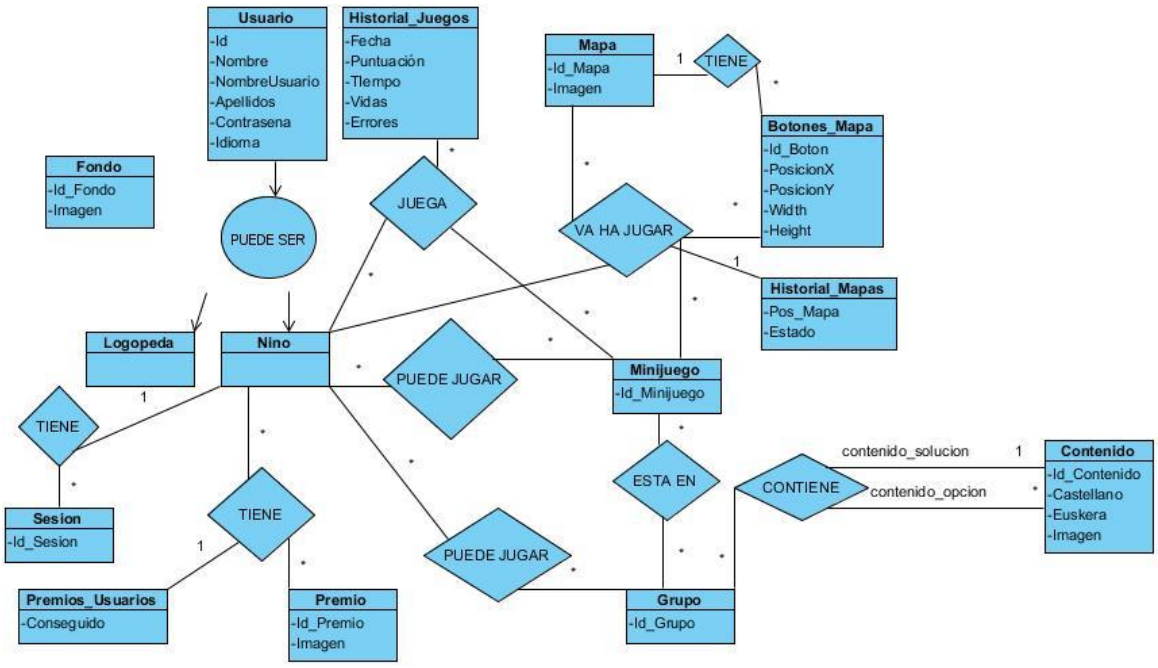


Figura 16: Modelo de Dominio

- Usuario: Es la entidad encargada de contener la información del usuario conectado. Debido a que de esta entidad van a heredar tanto la entidad Logopeda, como la entidad Nino, se ha puesto de una forma diferente a las relaciones.
- Logopeda: Es la entidad encargada de guardar la información de la Logopeda.
- Nino: Es la entidad encargada de guardar la información del paciente. A pesar de que no se le hayan puesto atributos (los que hereda de usuario solo), esta entidad tendrá una gran cantidad de atributos debido a la necesidad de guardar toda su información.
- Sesión: Esta entidad encargada de guardar las diferentes sesiones de cada paciente.
- Premio: Es la entidad encargada de guardar la imagen de cada premio.
- Premio_Usuarios: Entidad encargada de guardar para cada premio y cada usuario, si está conseguido o no.
- Contenido: Entidad encargada de guardar el contenido con el que se jugará en el videojuego.
- Grupo: Es la entidad encargada de agrupar el diferente contenido para utilizarlo en conjunto.
- Minijuego: Es la entidad encargada de guardar la información relacionada con los diferentes minijuegos.
- Historial_Juego: Es la entidad encargada de guardar los datos de las partidas o juegos jugados.
- Mapa: Es la entidad encargada de guardar la imagen de cada mapa.
- Botones_Mapa: Es la entidad encargada de guardar los botones de cada mapa y la posición de los botones en el mismo.
- Historial_Mapas: Es la entidad encargada de guardar el orden de los mapas que va a visualizar cada paciente, junto con el estado de cada botón (Actual, Pasado o Futuro).
- Fondo: Es la entidad encargada de guardar la imagen de cada fondo. No tiene relación con ninguna otra entidad.

Análisis y Diseño

En este apartado se explicará cómo se plantearon las diferentes partes del proyecto. Para ello, se ha dividido en cuatro partes: Programa de gestión, videojuego, base de datos remota, base de datos local.

1. Programa de Gestión

En esta parte se explicará cómo ha sido gestionado el programa de gestión de una forma general. Este programa será el utilizado por la logopeda para gestionar tanto los datos de los diferentes pacientes, como los diferentes aspectos del videojuego. Para ello, se va a mostrar el diagrama de clases del mismo. Debido al gran número de clases que posee este programa, estas se han dividido en varios diagramas de clases. Estos diagramas, a su vez, están relacionados con los elementos que utilizar la librería *MyBatis* utilizada para gestionar las sentencias y el acceso a base de datos. Esta librería será explicada más a fondo en el apartado de Desarrollo. Además, se ha utilizado el patrón *observador-observable.* De esta forma, se ha creado, para gestionar los diferentes datos., una clase gestor casi por cada ventana.

Por todo esto, se ha dividido este apartado en tantas partes como diagramas creados. De esta forma, se realizará una breve explicación de las clases que intervienen en cada uno de ellos.

1.1. **Ventanas**

A continuación, se explicarán brevemente las diferentes clases que intervienen en el diagrama de la Figura 17:

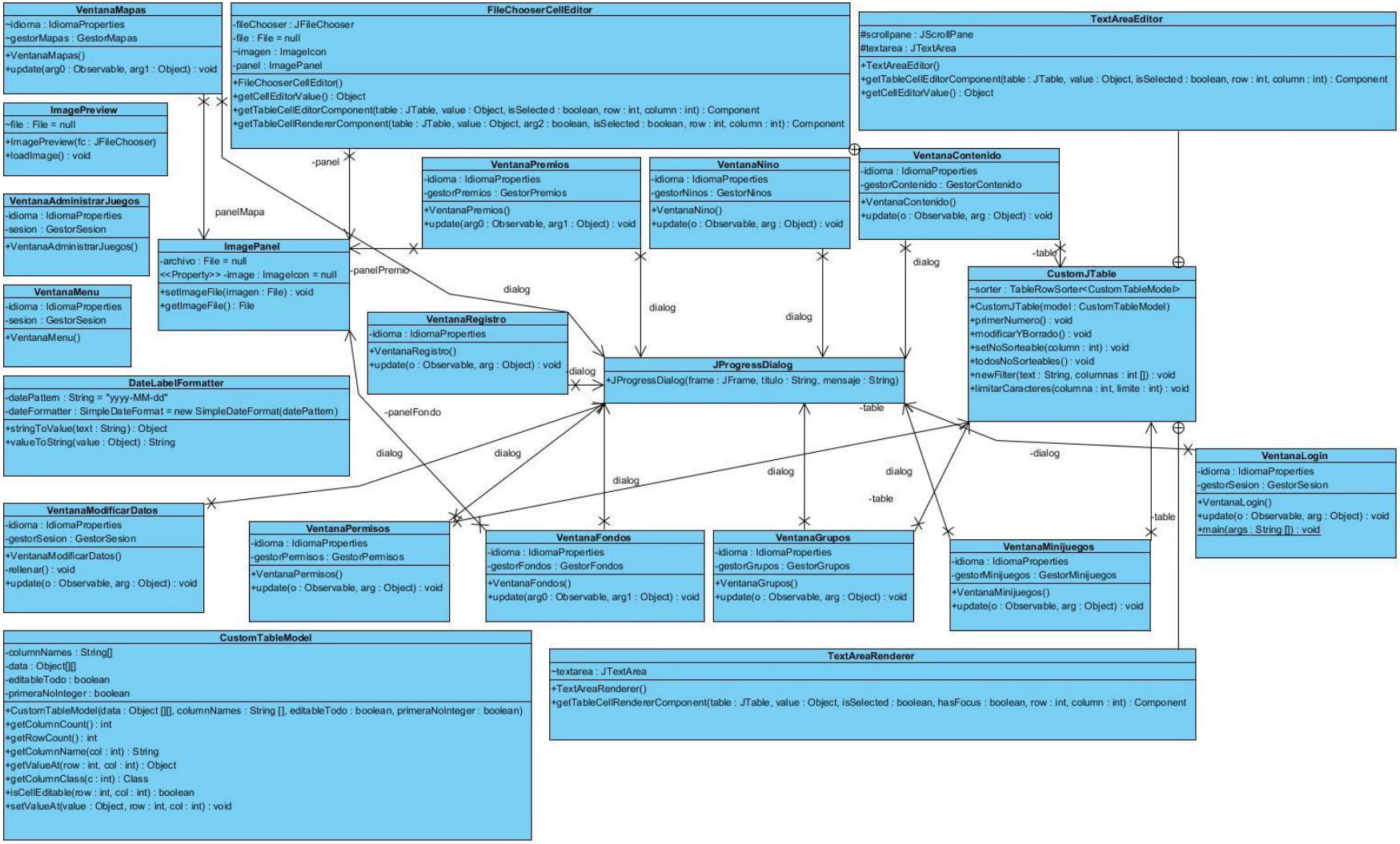


Figura 17: Diagrama de clases ventanas

- VentanaLogin: Clase encargada de mostrar la ventana de *Login*. Es la ventana principal (contiene el método *main*). Extiende de la clase *JFrame* e implementa la clase *Observer*.
- VentanaRegistro: Clase encargada de mostrar la ventana de *Registro*. Extiende de la clase *JFrame* e implementa la clase *Observer*.
- VentanaMenu: Clase encargada de mostrar la ventana de *Menú*. Extiende de la clase *JFrame* e implementa la clase *Observer*.
- VentanaModificarDatos: Clase encargada de mostrar la ventana de *ModificarDatos*. Extiende de la clase *JFrame* e implementa la clase *Observer*.
- VentanaNino: Clase encargada de mostrar la ventana de *Nino*. Extiende de la clase *JFrame* e implementa la clase *Observer*. Además, hace uso de un panel con *scroll* vertical para visualizar toda la información sobre el paciente.
- VentanaAdministrarJuegos: Clase encargada de mostrar la ventana de *AdministrarJuegos*. Extiende de la clase *JFrame* e implementa la clase *Observer*.
- VentanaFondos: Clase encargada de mostrar la ventana de *Fondo*. Extiende de la clase *JFrame* e implementa la clase *Observer*. Hace uso de la clase *ImagePanel* para visualizar la imagen de fondo en la pantalla y de la clase *ImagePreview* para personalizar el *filechooser*.
- VentanaMapas: Clase encargada de mostrar la ventana de *Mapa*. Extiende de la clase *JFrame* e implementa la clase *Observer*. Hace uso de la clase *ImagePanel* para visualizar la imagen de fondo en la pantalla y de la clase *ImagePreview* para personalizar el *filechooser*.
- VentanaPremios: Clase encargada de mostrar la ventana de *Premio*. Extiende de la clase *JFrame* e implementa la clase *Observer*. Hace uso de la clase *ImagePanel* para visualizar la imagen de fondo en la pantalla y de la clase *ImagePreview* para personalizar el *filechooser*.
- VentanaContenido: Clase encargada de mostrar la ventana de *Contenido*. Extiende de la clase *JFrame* e implementa la clase *Observer*. Hace uso de la clase *CustomTableModel* y CustomJTable para visualizar una tabla. Además, también utiliza la clase *ImagePanel* para visualizar la imagen del contenido en la celda correspondiente y de la clase *ImagePreview* para personalizar el *filechooser*.
- VentanaGrupos: Clase encargada de mostrar la ventana de *Grupo*. Extiende de la clase *JFrame* e implementa la clase *Observer*. Hace uso de la clase *CustomTableModel* y CustomJTable para visualizar una tabla.
- VentanaMinijuegos: Clase encargada de mostrar la ventana de *Minijuego*. Extiende de la clase *JFrame* e implementa la clase *Observer*. Hace uso de la clase *CustomTableModel* y CustomJTable para visualizar una tabla.
- VentanaPermisos: Clase encargada de mostrar la ventana de *Permiso*. Extiende de la clase *JFrame* e implementa la clase *Observer*. Hace uso de la clase *CustomTableModel* y CustomJTable para visualizar una tabla.
- JProgressDialog: Clase encargada de configurar el *JProgressDialog* (Ventana con una barra de progreso) para todas las ventanas. Esta clase extiende de la clase *JDialog*.
- CustomJTable: Clase encargada de configurar las tablas. Extiende de la clase JTable, y hace uso de la clase *CustomTableModel* para definir el modelo de la tabla. Además, tiene en su interior una subclase llamada *TextAreaRenderer*, utilizada para gestionar las áreas de texto en el interior de las celdas.
- CustomTableModel: Clase encargada de definir el modelo que utilizarán las tablas de cada ventana. Para esto, extiende de la clase *AbstractTableModel*.
- DateLabelFormatter: Clase encargada de definir el formato de fecha que se mostrará en los elementos (*JLabel*) correspondientes.
- ImagePanel: Clase encargada de dibujar o pintar la imagen que se le pase como parámetros. Para esto, extiende de la clase *JPanel*, y se le puede pasar una imagen (*ImageIcon*) o un archivo (*File*) como parámetro.
- ImagePreview: Clase encargada de visualizar el archivo seleccionado (en caso de que sea una imagen) a la hora de seleccionar una imagen con la clase *filechooser*.

1.2. **Mappers**

Las clases que se pueden apreciar en el diagrama de la Figura 18, son algunas de las clases utilizadas por la librería *MyBatis*. Se tratan de interfaces, por lo que la implementación de sus métodos se realizará en una clase. Además, cada *Mapper* se utilizará para acceder a los datos de la tabla del mismo nombre (Por ejemplo, la clase *GrupoMapper* será utilizada para acceder a los datos de la tabla *Grupo*).

Por otro lado, la clase *MyBatisUtil* será la encargada de seleccionar el archivo de configuración tanto de la base de datos, como de los *XML*.

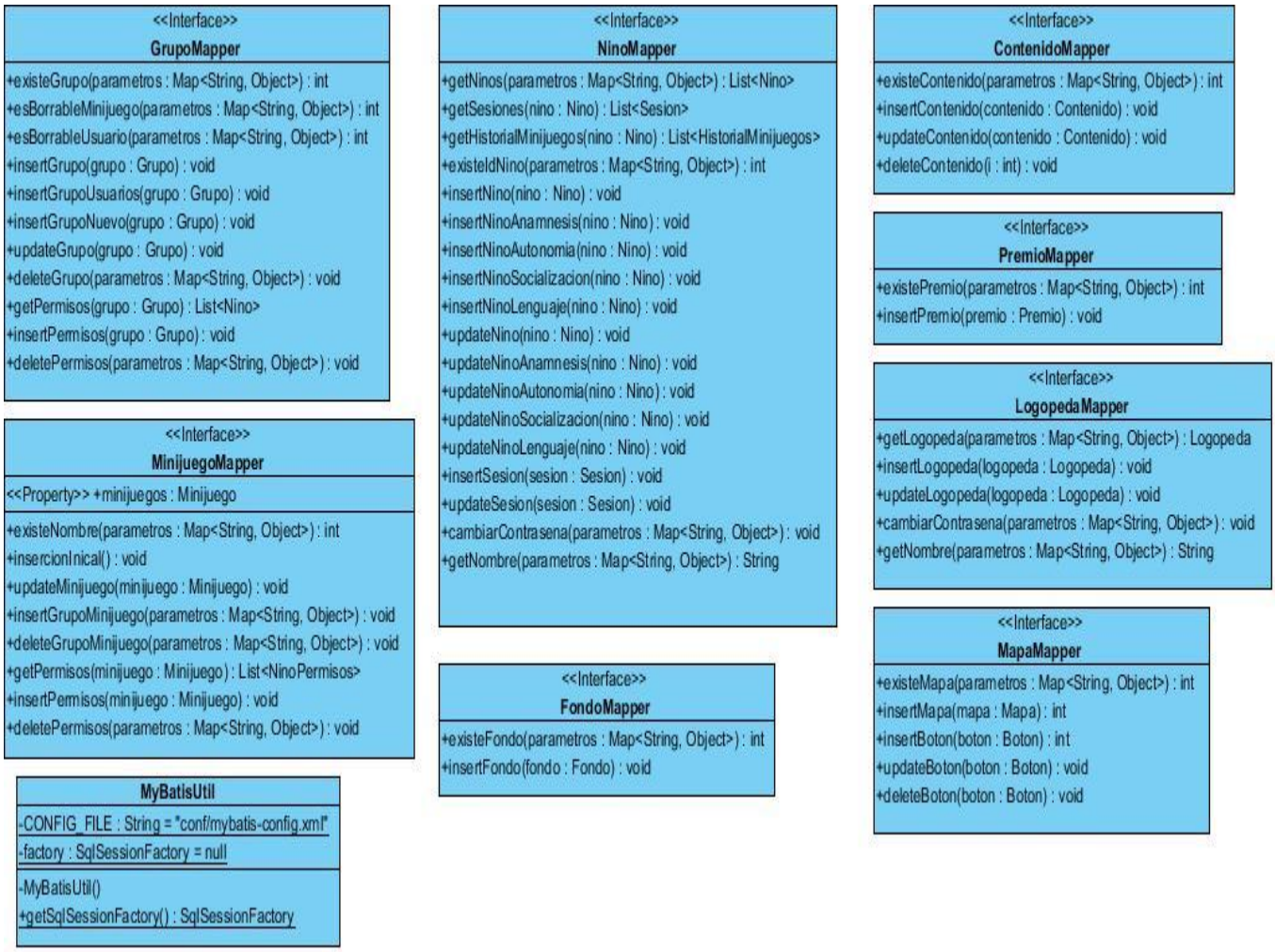


Figura 18: Diagrama de clases Mapper

1.3. **DAOs (Data Access Object)**

Las clases que se pueden apreciar en el diagrama de la Figura 19, son algunas de las clases utilizadas por la librería *MyBatis*. Se tratan de las clases que implementan las interfaces *Mapper*. Además, cada *DAO* se utilizará para acceder a los datos de la tabla del mismo nombre (Por ejemplo, la clase *GrupoDAO* será utilizada para, mediante la interfaz *GrupoMapper,* acceder a los datos de la tabla *Grupo*).

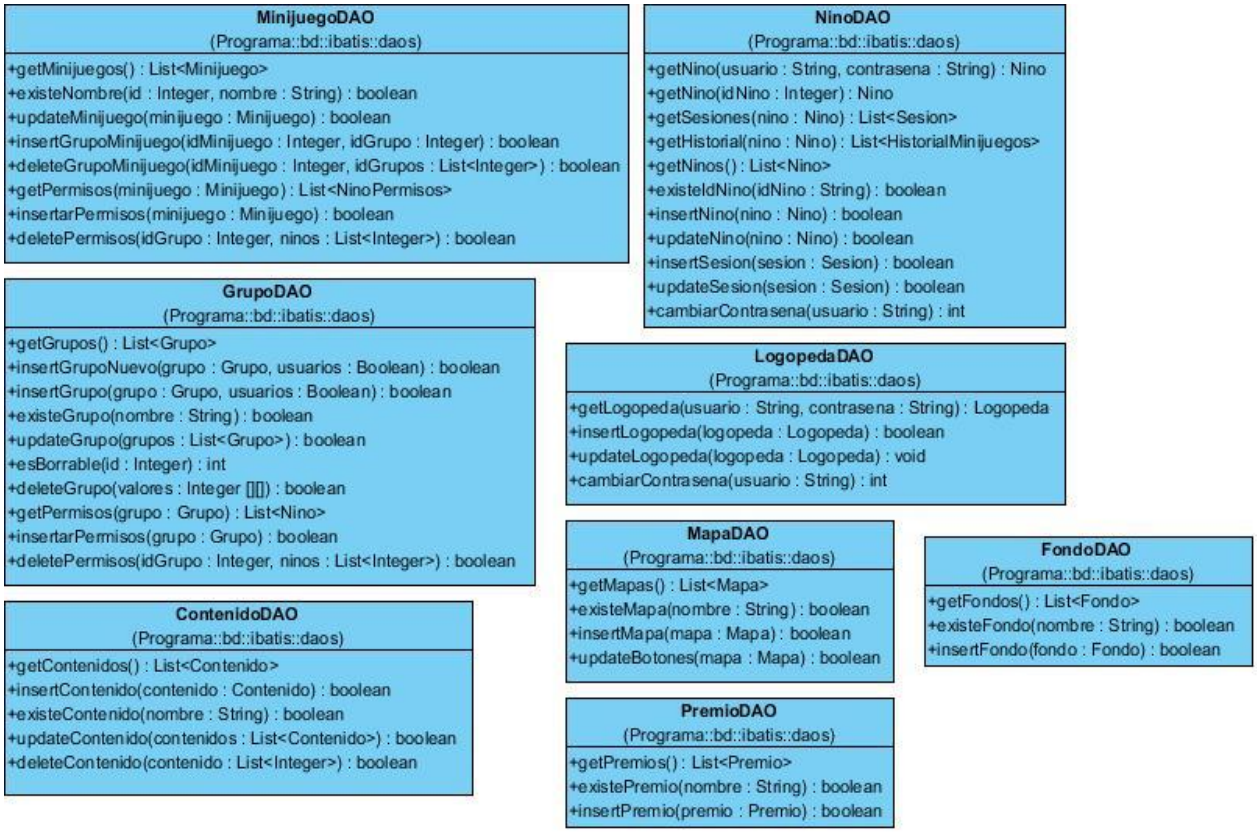


Figura 19: Diagrama de clases DAO

1.4. **Objetos**

Las clases relacionadas con el diagrama que se puede ver en la Figura 20, son las clases java relacionada con el modelo. Estas clases están relacionadas con cada tabla de la base de datos (con los atributos de cada clase referenciados a los atributos de cada tabla). De esta forma, estas clases estarán formadas por los atributos correspondientes, los *getters* y *setters* de dichos atributos, y las constructoras necesarias.

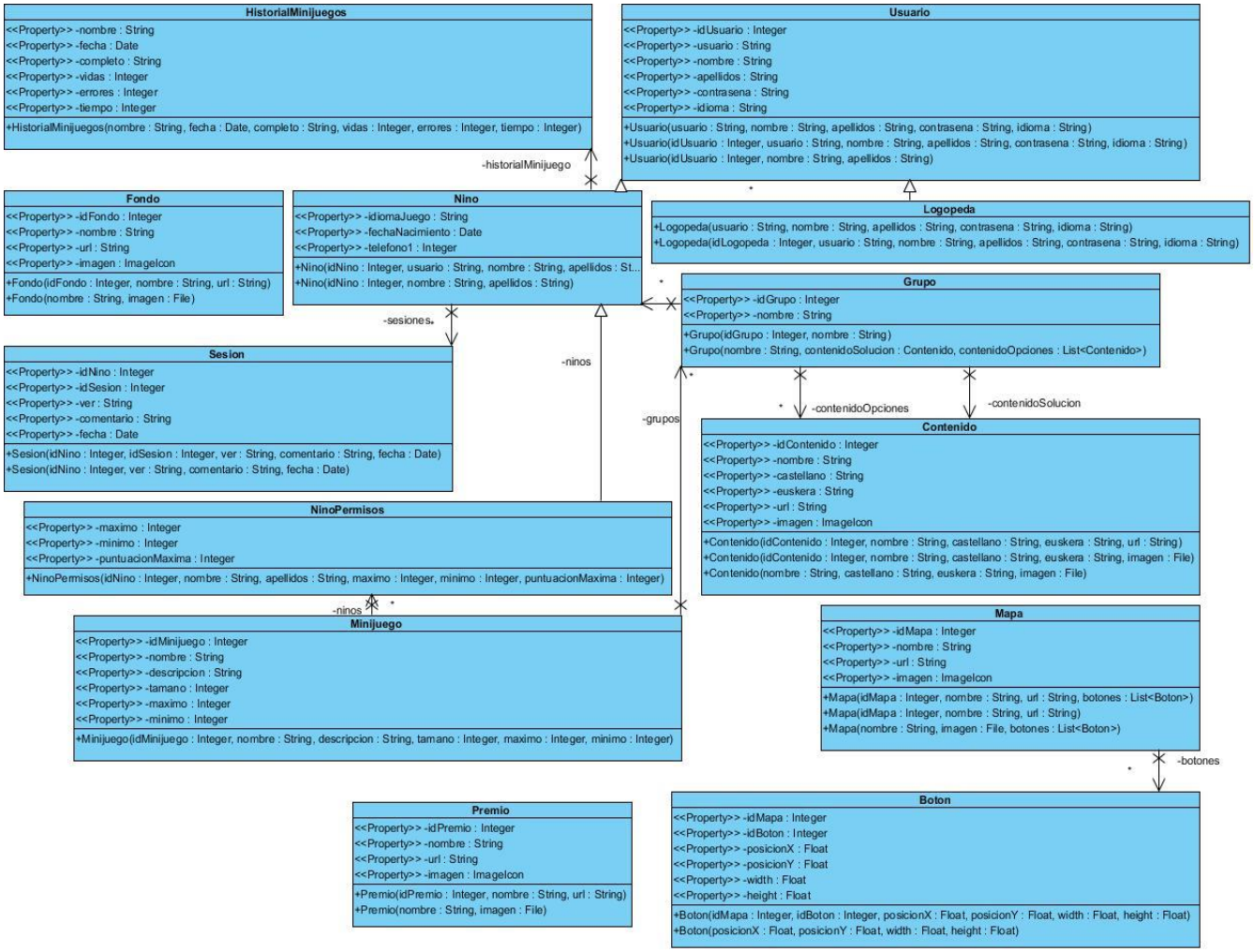


Figura 20: Diagrama de clases Objetos

1.5. **Controlador**

Por último, dentro del apartado del programa de gestión, tenemos el controlador (o clases de control). Estas clases, en su mayoría, está formadas por los gestores de las ventanas. Las diferentes clases que se pueden observar en la Figura 21 son las siguientes:

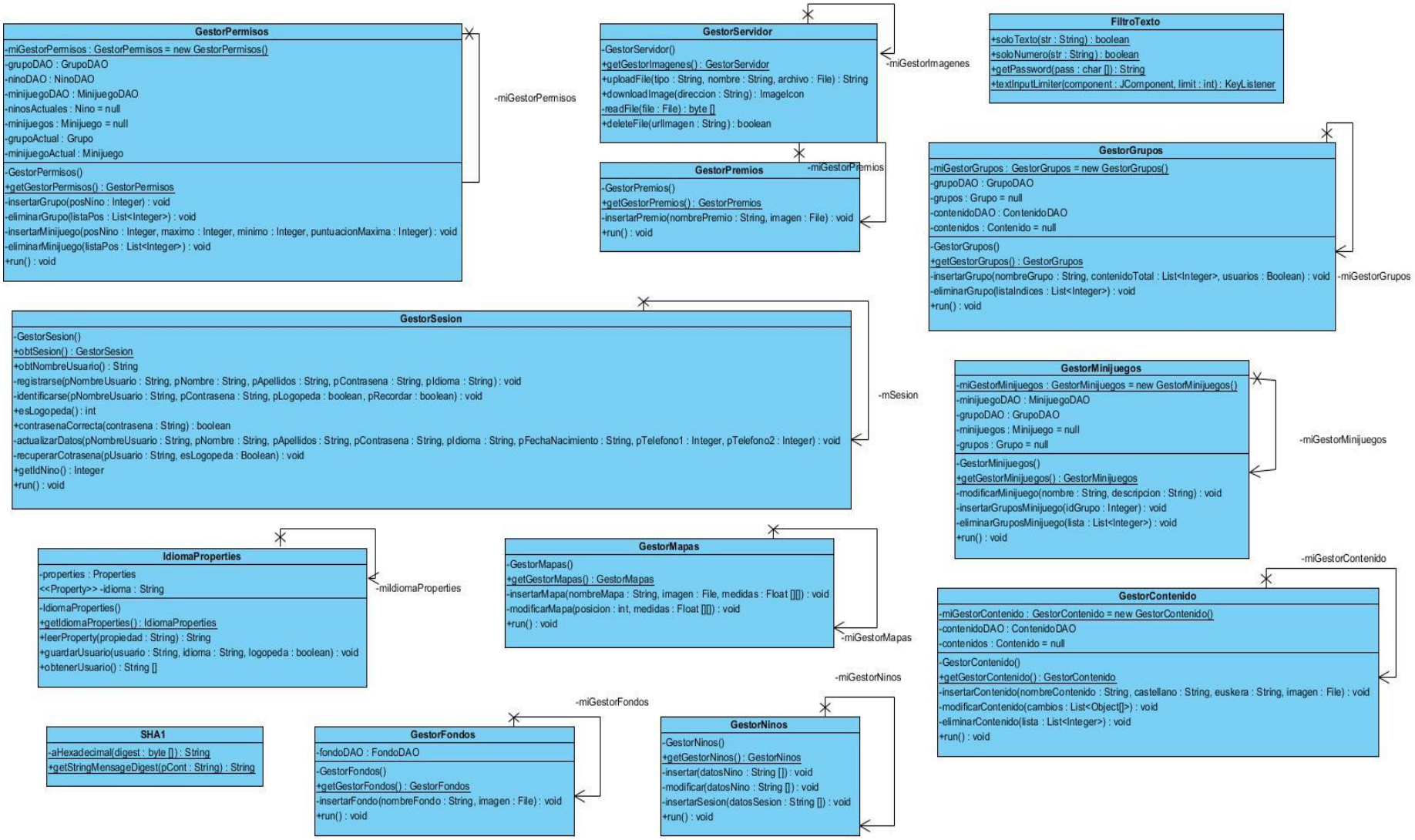


Figura 21: Diagrama de clases Modelo

- GestorSesion: Clase encargada de gestionar los datos del usuario conectado o usuario que se quiere registrar o conectar. De esta forma, será el responsable de verificar las credenciales y crear usuarios nuevos. Está relacionado con la clase *VentanaLogin*, *VentanaRegistro* y *VentanaModificarDatos*. Implementa el patrón *Singleton* y la clase *Runnable*. Además, extiende de la clase *Observable*. Utiliza los *DAO LogopedaDAO y NinoDAO*, así como sus clases objeto *Nino* y *Logopeda*. También utiliza la clase objeto *Usuario* para globalizar el usuario (Tanto la clase *Logopeda* como la clase *Nino* heredan de la clase *Usuario*).
- GestorNinos: Clase encargada de gestionar los datos de la clase *VentanaNino*. Implementa el patrón *Singleton* y la clase *Runnable*. Además, extiende de la clase *Observable*. Utiliza el *DAO NinoDAO*, así como su clase objeto *Nino*.
- GestorFondos: Clase encargada de gestionar los datos de la clase *VentanaFondos*. Implementa el patrón *Singleton* y la clase *Runnable*. Además, extiende de la clase *Observable*. Utiliza el *DAO FondoDAO*, así como su clase objeto *Fondo*.
- GestorMapas: Clase encargada de gestionar los datos de la clase *VentanaMapas*. Implementa el patrón *Singleton* y la clase *Runnable*. Además, extiende de la clase *Observable*. Utiliza el *DAO MapaDAO*, así como su clase objeto *Mapa*.
- GestorPremios: Clase encargada de gestionar los datos de la clase *VentanaPremios*. Implementa el patrón *Singleton* y la clase *Runnable*. Además, extiende de la clase *Observable*. Utiliza el *DAO PremioDAO*, así como su clase objeto *Premio*.
- GestorContenido: Clase encargada de gestionar los datos de la clase *VentanaContenido*. Implementa el patrón *Singleton* y la clase *Runnable*. Además, extiende de la clase *Observable*. Utiliza el *DAO ContenidoDAO*, así como su clase objeto *Contenido*.
- GestorGrupos: Clase encargada de gestionar los datos de la clase *VentanaGrupos*. Implementa el patrón *Singleton* y la clase *Runnable*. Además, extiende de la clase *Observable*. Utiliza los *DAO GrupoDAO* y *ContenidoDAO*, así como sus clases objeto.
- GestorMinijuegos: Clase encargada de gestionar los datos de la clase *VentanaMinijuegos*. Implementa el patrón *Singleton* y la clase *Runnable*. Además, extiende de la clase *Observable*. Utiliza los *DAO GrupoDAO* y *MinijuegoDAO*, así como sus clases objeto.
- GestorPermisos: Clase encargada de gestionar los datos de la clase *VentanaPermisos*. Implementa el patrón *Singleton* y la clase *Runnable*. Además, extiende de la clase *Observable*. Utiliza los *DAO GrupoDAO*, *NinoDAO* y *MinijuegoDAO*, así como sus clases objeto.
- GestorServidor: Clase encargada de realizar la conexión con el servidor para la descarga y subida de las imágenes. Implementa el patrón *Singleton*. Además, para realizar las conexiones al servidor, utiliza la clase *HttpURLConnection*.
- FiltroTexto: Clase encargada de gestionar las entradas de texto de cada campo. Se gestiona si es número, si es texto (sin números) y el tamaño permitido (mediante *KeyListener*). Para poder llamarse desde cualquier clase, sin tener que instanciarla, todos los métodos son estaticos.
- IdiomaProperties: Clase encargada de obtener el texto de cada elemento en el idioma correspondiente. Implementa el patrón Singleton. Además, mediante la clase *Preferences* se ha gestionado el guardado de sesión.
- SHA1: Clase encargada de cifrar el contenido de un texto mediante el algoritmo SHA1.

2. Videojuego

En esta parte se explicará cómo se ha gestionado el videojuego (la aplicación para dispositivo móvil con la que los pacientes realizarán los ejercicios). Para realizar una mejor explicación, al igual que con el programa de gestión, se utilizarán diagramas de clases. En este caso, se mostrarán todas las clases unidas en un mismo diagrama debido a que el número total de clases es bastante inferior (Ver Figura 22). Esto es debido a que, para acceder a la base de datos, se utiliza una única clase. Además, no existen clases separadas para realizar el patrón *observador-observable*, si no que todo lo realiza la misma clase. Por todo esto, a continuación, se va a llevar a cabo una breve descripción de cada clase:

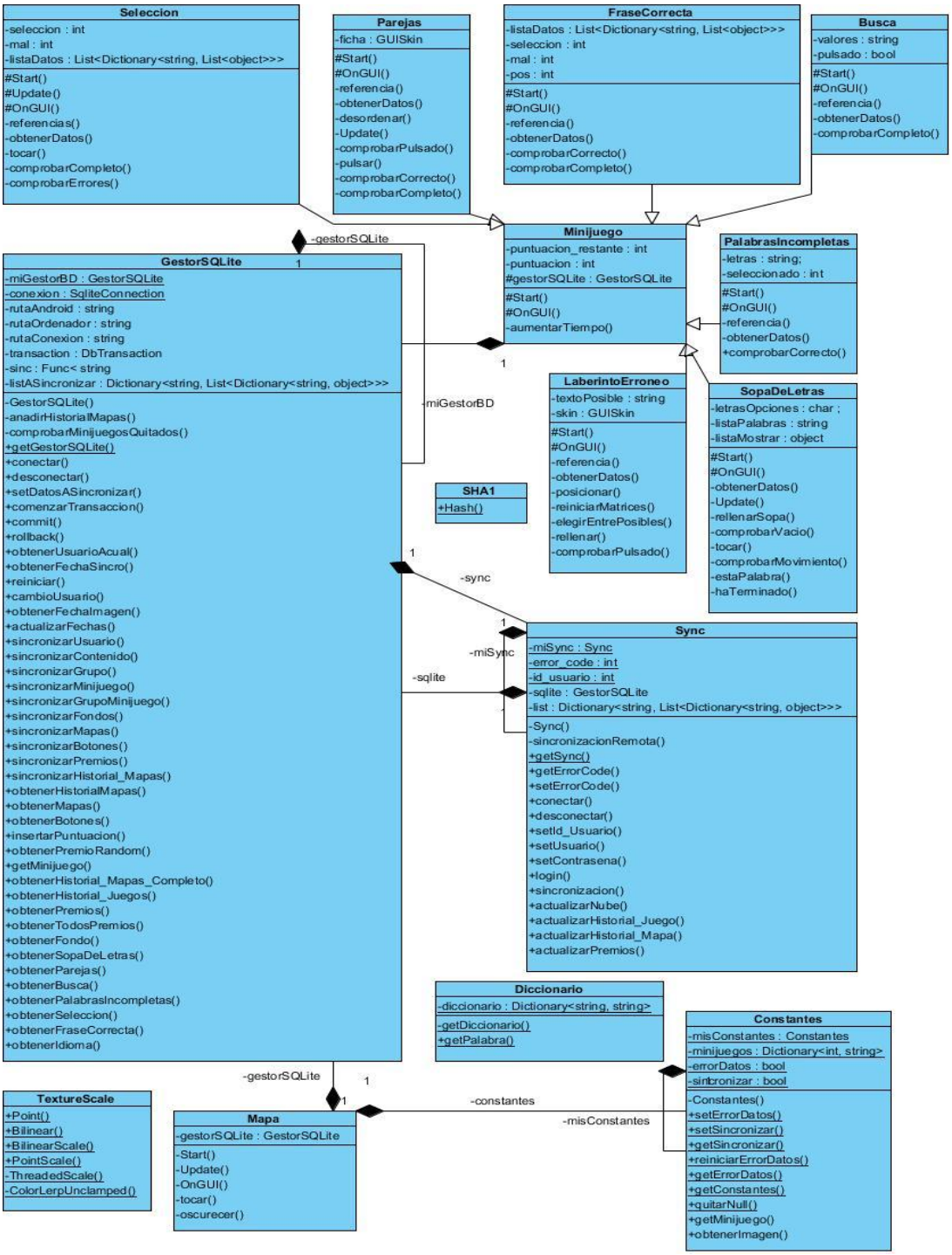


Figura 22: DIagrama de clases Videojuego

- Constantes: Clase encargada de realizar acciones utilizadas entre varias clases. Entre estas funciones, está el obtener las imágenes del servidor (mediante *WebClient*), guardar la relación de cada identificador de minijuego con su respectivo objeto y gestionar cuando es necesaria la sincronización y cuando ha dado esta un error.
- Diccionario: Clase encargada de seleccionar en todo momento el texto en el idioma indicado. Para ello, utiliza la clase *Dictionary* (*Map*).
- GestorSQLite: Clase encargada de lo que se refiere a la conexión con la base de datos local. Entre sus funciones, está la de insertar datos, obtener datos, conectarse, desconectase y crear transacciones (por si es necesario hacer *rollback* por la existencia de algún error).

- Login: Clase encargada de visualizar y gestionar lo relacionado con el *Login*. Es la clase principal, la cual es llamada nada más iniciarse el videojuego.
- Mapa: Clase encargada de la gestión de todas las acciones que se pueden realizar en la ventana del mapa. Dentro de estas acciones, se encuentra el moverse por el mapa, ver los premios conseguido, cambiar el idioma, desconectarse y jugar a cualquier minijuego.
- SHA1: Clase encargada de cifrar la contraseña introducida en la ventana de Login mediante el algoritmo *SHA1*. Esto se realiza para aumentar la seguridad y para poder comprobar las credenciales (en la base de datos, la contraseña está guardad mediante este cifrado).
- Sync: Clase encargada de la conexión contra la base de datos remota. Debido a esto, es la que tiene el mayor control de la sincronización de los datos de la base de datos remota a la local.
- TextureScale: Clase encargada de cambiar el tamaño de la clase *Texture2D* (clase de *Unity3D*) una vez formada ya la imagen.
- Minijuego: Clase encargada de gestionar todas las acciones que se generan de la misma forma en todos los minijuegos. Entre estas acciones, se encuentra el mostrar los elementos básicos de los minijuegos (título, botón de opciones, botón de ayuda y vidas), gestionar las acciones a realizar al terminar el minijuego, gestionar el tiempo y las opciones.
- Busca: Clase encargada de gestionar lo relacionado al minijuego 'Busca'. Esta clase hereda de la clase Minijuego, por lo que hará todas las acciones que tiene dicha clase.
- FraseCorrecta: Clase encargada de gestionar lo relacionado al minijuego 'Frase Correcta'. Esta clase hereda de la clase Minijuego, por lo que hará todas las acciones que tiene dicha clase.
- LaberintoErroneo: Clase encargada de gestionar lo relacionado al minijuego 'Laberinto Erróneo'. Esta clase hereda de la clase Minijuego, por lo que hará todas las acciones que tiene dicha clase.
- PalabrasIncompletas: Clase encargada de gestionar lo relacionado al minijuego 'Palabras Incompletas'. Esta clase hereda de la clase Minijuego, por lo que hará todas las acciones que tiene dicha clase.
- Parejas: Clase encargada de gestionar lo relacionado al minijuego 'Parejas'. Esta clase hereda de la clase Minijuego, por lo que hará todas las acciones que tiene dicha clase.
- Seleccion: Clase encargada de gestionar lo relacionado al minijuego 'Selección'. Esta clase hereda de la clase Minijuego, por lo que hará todas las acciones que tiene dicha clase.
- SopaDeLetras: Clase encargada de gestionar lo relacionado al minijuego 'Sopa de Letras'. Esta clase hereda de la clase Minijuego, por lo que hará todas las acciones que tiene dicha clase.

3. Base de Datos Remota

En esta parte se explicará cómo ha sido gestionada la base de datos remota en este proyecto. Esto, se antes de crear la base de datos se diseñó como serían sus tablas y relaciones (Ver Figura 23). Para evitar el uso de un alias en las consultas, se decidió utilizar prefijos en los atributos. Estos prefijos serán la primera letra del nombre de la tabla. Estas tablas se explicarán brevemente a continuación:

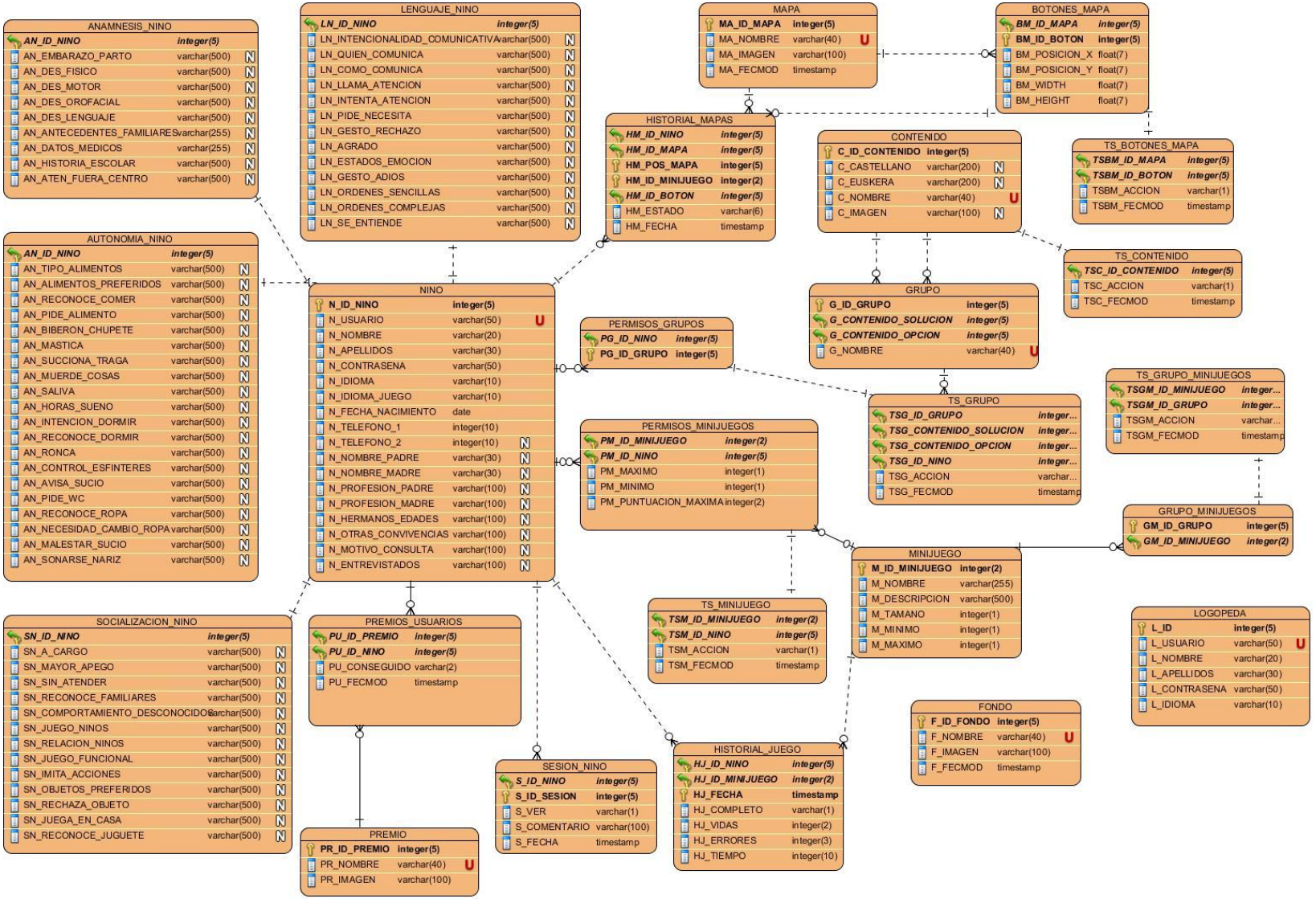


Figura 23: Diagrama de tablas Remotas

- Nino: Se encuentran los datos de acceso e información de los pacientes.
- Ananmesis_Nino: Se encuentra parte de la información personal de cada paciente.
- Autonomia_Nino: Se encuentra parte de la información personal de cada paciente.
- Socializacion_Nino: Se encuentra parte de la información personal de cada paciente.
- Lenguaje_Nino: Se encuentra parte de la información personal de cada paciente.
- Logopeda: Se encuentran los datos de acceso e información de la logopeda.
- Contenido: Contiene el contenido del minijuego (cada palabra en castellano, euskera y su imagen si se le asigna una).
- Grupo: Contiene la relación entre los contenidos. Esta tabla es la encargada de agrupar los diferentes contenidos en uno (por ejemplo, para agrupar la palabra completa y la palabra con la letra que falta para el minijuego 'Palabras Incompletas').
- Minijuego: Contiene el nombre, descripción, tamaño máximo de cada grupo (por ejemplo, en el juego 'Separar' será 4, y en la 'Sopa de Letras', 1), cantidad mínima y máxima de grupos (ambos atributos utilizados para seleccionar por la logopeda cuantos grupos le van a aparecer a un paciente a la hora de jugar). Estos tres últimos atributos, son atributos fijos para cada minijuego.
- Permisos_Minijuegos: Contiene el valor de los atributos máximo y mínimo mencionados en la tabla 'Minijuego' para cada paciente y cada minijuego. Además, será la tabla encargada de mostrar qué minijuegos puede jugar cada paciente.
- Permisos_Grupos: Es la tabla encargada de mostrar con qué grupos puede jugar un paciente.
- Grupo_Minijuego: Tabla encargada de asociar los grupos con los minijuegos (No todos los grupos se tienen que poder ver en todos los minijuegos).
- Premio: Contiene el nombre e imagen de cada premio.
- Premios_Usuarios: Es la tabla encargada de dar permisos de los premios a los pacientes, y de guardar el estado de los premios (conseguido o no).
- Fondo: Contiene el nombre del fondo (para identificarlo) y su imagen.
- Mapa: Contiene el nombre del mapa (para identificarlo) y su imagen.
- Botones_Mapa: Contiene la posición y tamaño de los botones, junto con el identificador numérico del mapa al que pertenecen.
- Historial_Mapas: Tabla encargada de guardar los botones que puede cada paciente ver (junto con el mapa al que corresponden), los minijuegos que cada botón tiene asignado, y el estado de dichos botones ('Pasado', 'Actual' o 'Futuro' si es un botón de un mapa futuro). La necesidad de esta tabla viene debido a que, si un mismo paciente juega en diferentes dispositivos, este tiene que poder tener los mismos datos y mapas en ambos dispositivos.
- Historial_Juego: Contiene la información del resultado de cada paciente en cada minijuego jugado (fecha en la que se ha jugado, si se ha completado, las vidas que le han quedado, el número de errores cometidos y el tiempo tardado en terminar).
- Tablas_TS: Estas tablas, acompañadas de un prefijo referente a algunas tablas mencionadas anteriormente (*TS_Grupo* por ejemplo), son las encargadas de la sincronización. Todas contienen la clave primaria de la tabla a la que hacen referencia, y un atributo de estado y fecha de modificación. El estado será 'A', 'B' o 'M', dependiendo si la última acción realizada sea 'Añadir' 'Borrar' o 'Modificar'. De esta forma, aunque un dato desaparezca de la tabla original, en estas tablas seguirá teniendo constancia de su existencia para poder eliminarse de cada dispositivo al sincronizarse.

4. Base de Datos Local

En esta última parte nos encontramos con la base de datos local. Esta base de datos tiene una gran similitud con la base de datos remota explicada anteriormente (Ver Figura 24). Un dato a tener en cuenta es que esta base de datos no es relacional, es decir, las tablas no dispondrán de relación entre ellas (no tendrán *foreign key*). Esto se ha realizado de esta forma para prevenir errores de inexistencia de datos a la hora de realizar la sincronización (porque se realiza de forma asíncrona contra todas las tablas a la vez). Además, como en los anteriores apartados, a continuación, se explicarán las diferentes tablas que intervienen.

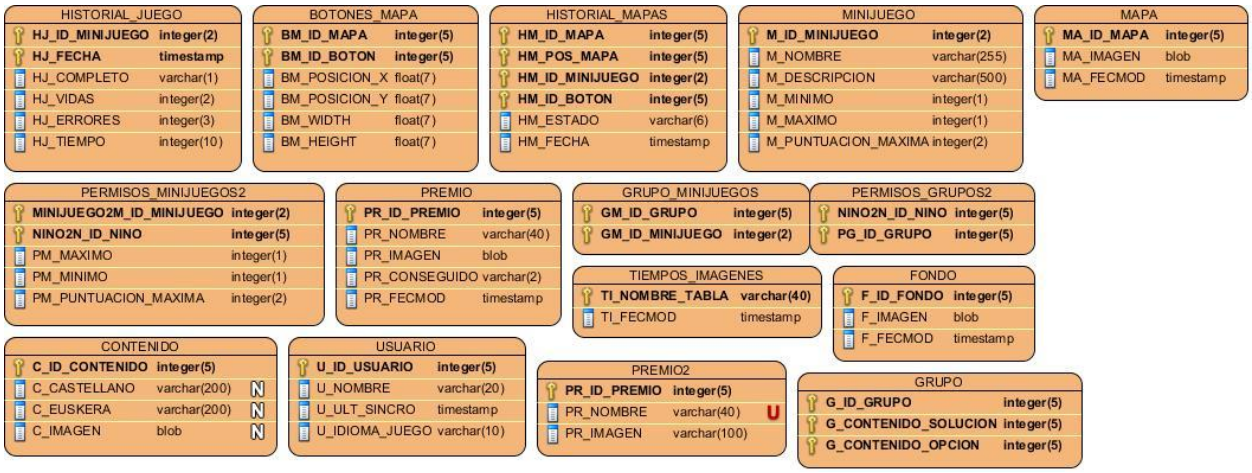


Figura 24: Diagrama de tablas Local

- Usuario: Tabla encargada de guardar el usuario actual y la última fecha en la se ha sincronizado.
- Contenido: Tabla con las mismas características que la tabla de Contenido de la base de datos remota.
- Minijuego: Tabla similar a la tabla de Minijuego de la base de datos remota. La única diferencia es que el atributo tamaño en esta tabla no está, y los atributos máximo y mínimo contendrán los valores para esos atributos, pero de la tabla Permisos_Minijuegos de la base de datos remota para el este usuario. Además, se ha añadido el atributo puntuación_maxima de esta misma tabla.
- Grupo: Tabla con las mismas características que la tabla de Grupo de la base de datos remota.
- Grupo_Minijuegos: Tabla con las mismas características que la tabla de Grupo_Minijuegos de la base de datos remota.
- Premio: Con esta tabla ocurre algo similar a la de Minijuego, pero en este caso combina las tablas Premio y Premios_Usuarios de la base de datos remota.
- Fondo: Tabla con las mismas características que la tabla de Fondo de la base de datos remota.
- Mapa: Tabla con las mismas características que la tabla de Mapa de la base de datos remota.
- Botones_Mapa: Tabla con las mismas características que la tabla de Botones_Mapa de la base de datos remota.
- Historial_Mapa: Tabla con las mismas características que la tabla de Historial_Mapa de la base de datos remota.
- Historial_Juego: Tabla con las mismas características que la tabla de Historial_Juego de la base de datos remota. Está tabla, a diferencia de las anteriores, no se sincronizará contra la base de datos remota. Con esto se quiere decir que no se volcarán los datos de la misma tabla de la base de datos remota a esta, si no que se realizará al revés (los datos de local se copiarán a la remota). Esto se hace porque esta tabla contiene los resultados de los minijuegos (cada juego o ejercicio individual), los cuales no hacen falta en el propio minijuego.
- Tiempos_Imagenes: Esta tabla es la encargada de guardar los tiempos de sincronización. Estos tiempos son los referentes a las tablas de Fondo y Mapas. Esto se ha hecho así, y no con los tiempos de la última sincronización, para agilizar las sincronizaciones en caso de que un usuario se desconecte (para no tener que cargar todos los fondos y mapas de nuevo, los cuales tardan mucho en sincronizar debido a las imágenes de pantalla completa).

Desarrollo

El desarrollo del proyecto comenzó por la investigación de qué información guarda un logopeda sobre un paciente. Esta información no sólo era importante para diseñar la base de datos, sino que también para saber qué datos poner para modificar o añadir en un paciente. Este tipo de información se llama Anamnesis, y contiene toda la información del paciente (tanto familiar, como física o social) (Consejería de Educación del Principado de Asturias).

Una vez obtenido el formato de la información, el siguiente paso era obtener los tipos de minijuegos diferentes que se iban a desarrollar. Para esto, se buscaron diferentes tipos de ejercicios en libros de educación, como los libros de la editorial *SM*. De esta forma, se obtuvieron inicialmente 21 minijuegos.

Una vez terminada la investigación, y tras haber realizado la captura de requisitos y el diseño de los diferentes diagramas, el proyecto se volcó completamente en realizar la implementación. Esta implementación se comenzó por la parte del videojuego. Más tarde se desarrollaron las bases de datos, para implementarlas luego en el videojuego y concluir en el programa de gestión. El motivo principal de empezar por el videojuego, y luego volver a él más tarde, y no por la base de datos e implementar el videojuego del tirón, fue para evitar la construcción de la base de datos numerosas veces. Con esto último se quiere decir que, a pesar de haber diseñado tanto la implementación como las bases de datos en la tarea de Diseño Técnico, a la hora de implementar los minijuegos surgía la duda de qué elementos de la base de datos iba a necesitar cada minijuego en realidad. Estos elementos solo se iban a saber una vez implementado cada minijuego, por lo que se decidió primero crear estos, y luego realizar la base de datos. Por todo esto, a continuación, se explicará cómo fue el desarrollo de cada parte. En caso del videojuego, se dividirá, como ya se ha mencionado, en dos fases.

1. Videojuego Fase 1

En esta primera fase del videojuego, debido a que es la primera toma de contacto con *Unity3D*, se explicará tanto el funcionamiento del programa, como los fundamentos básicos de lenguaje de programación utilizado enfocado a *Unity* (*C#*), y la descripción de cómo ha sido el desarrollo de cada minijuego (cada juego individual dentro de la aplicación móvil o videojuego). Por lo tanto, antes de explicar cómo se ha hecho, se deberá saber cómo es la herramienta que hemos utilizado.

1.1. **Unity 3D**

Unity 3D se divide en escenas. Cada escena contendrá una parte de videojuego. De esta forma, los elementos que existan en una escena no podrán interaccionar con otra, al igual que no podrán existir dos escenas ejecutándose a la vez. A pesar de que en nuestro caso podríamos haber hecho todo en una misma escena, nos ha resultado más útil el utilizar dos. La primera contiene únicamente el menú de *Login*. La segunda, por otro lado, está formada tanto por el mapa, como por los minijuegos.

Por otro lado, los objetos que pueden existir en cada escena se llaman *GameObject*. Estos objetos pueden ir desde un objeto vacío, el cual nos sirva para que se ejecute nuestro código, hasta un terreno sobre el cual todos los demás objetos interactúen. Todos estos elementos se listan en una ventana dentro del propio programa, además de verlos en una ventana en 3D (Ver Figura 25). A pesar de esta última característica, nosotros sólo contendremos dos *GameObjects*, la cámara (obligatoria para todas las escenas) y un *GameObject* que contendrá únicamente nuestra clase.

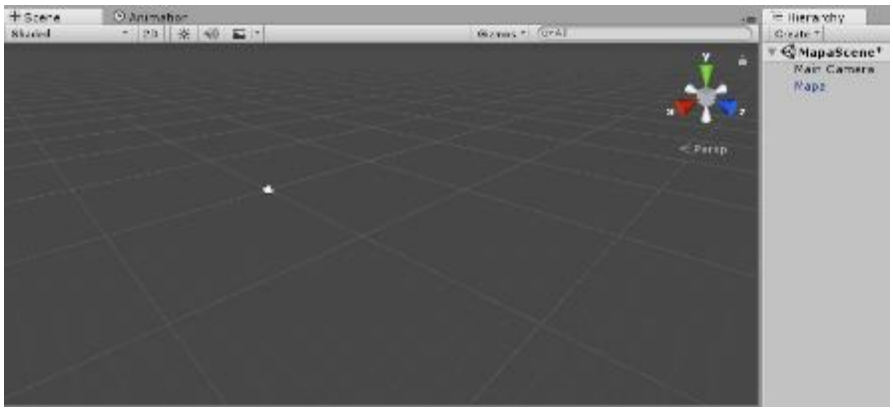


Figura 25: Ventana Escena

Por último, en cuanto se refiere a la interfaz, también existe una ventana con la funcionalidad de ver los archivos que veremos desde fuera del programa. Estos archivos serán los que se encuentren en la carpeta *Assets* del proyecto (esta carpeta se crea en el momento de crea un nuevo proyecto). Gracias a esta funcionalidad, podremos tanto ver los objetos que creemos o descarguemos desde fuera de Unity 3D (imágenes, sonidos, etc.), hasta los propios archivos y carpetas. A pesar de parecer de gran utilidad, no se podrá acceder al contenido que veamos en esta carpeta mediante código. Para poder hacer esto, deberemos crear una carpeta con el nombre de "Resources" en la raíz de la carpeta de *Assets*, y meter en su interior el contenido que queramos obtener mediante programación (Ver Figura 26). Entre todo el contenido que existe, vamos a utilizar los siguientes tipos:

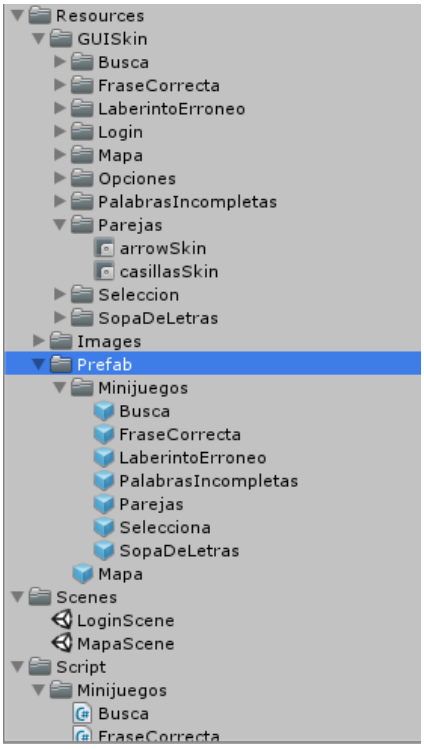


Figura 26: Ventana Project

- *GameObject*: También llamados *Prefabs* cuando se encuentra fuera de la escena, es cualquier elemento que pueda existir en la escena. (Elementos con el icono de un cubo)
- *Scene*: El archivo encargado de guardar el contenido de cada escena. (Elementos con el icono de Unity)
- Texture2D: El formato para las imágenes en 2D que menos espacio ocupa y mejor se moldea. También existe un elemento parecido *Sprite*, pero este último está más enfocado para animaciones en 2D.
- Script: Archivos con extensión ".cs" en los cuales se encontrará una clase C# en nuestro caso (también se podría *Javascript* y *Boo).*
- *GUISkin*: Archivo de configuración del formato que tendrán los diferentes *GUIs* (Interfaz Gráfica de Usuario), los cuales se verán más adelante. Esta configuración va desde la posibilidad de solo poner imágenes, solo texto, o ambos y la posición de los elementos, hasta la opción de cambiar el color de letra y el fondo al pasar el ratón por encima, o al pulsar, o al soltar el botón, etc. (Son los archivos con la terminación Skin – no es obligatoria esta terminación).

1.2. **C#**

El segundo apartado que habría que tratar, como ya se ha adelantado anteriormente, es el tema del lenguaje de programación utilizado. En este caso, se explicará no solo el propio lenguaje, sino cómo funciona la API de Unity 3D integrada en el. Sabiendo esto, a continuación, se explicarán los métodos y funciones utilizadas principalmente, y como funciona o qué hace cada una.

Cuando creamos un nuevo script se nos escribirá automáticamente dentro del propio archivo algunos métodos y librerías utilizadas. Todas las clases extienden de *MonoBehaviour* (al poner ":" –dos puntos- a la derecha del nombre de la clase, en c# se interpreta como extiende o hereda). Esta clase es la clase base de la que derivan todos los *scripts* de Unity 3D. Debido a esta herencia, también nos encontramos dos métodos predeterminados, *Start*, y *Update*. El primero método será llamado en el momento en el que el objeto que contenga este *script* es llamado o ejecutado. Por otro lado, el *Update* se podría decir que es el método principal. Este método es llamado o ejecutado cada *frame* o fotograma. Debido a esto, este método sería el encargado de comprobar si se pulsa alguna tecla, de mover un objeto por la pantalla, etc.

Además de los dos métodos mencionados, también existe otro método muy utilizado llamado *OnGUI*. Este método se encarga de dibujar o mostrar imágenes o interfaces (botones, cuadros de texto, etc.). Al igual que pasa con el *Update*, es llamado cada *frame*. Debido a la complejidad de este método (comprueba las posiciones de pantalla para posicionar los diferentes *GUIs*, comprobación de los elementos a posicionar (imágenes, texto o ambos), formato de la interfaz, control de botones, etc.), a la hora de realizar los diagramas de secuencia omitiremos su contenido (simplemente diremos que lo llamamos dentro de un bucle, al igual que el *Update*).

Como podemos ver, estos tres métodos mencionados son interfaces, es decir, tenemos que implementar lo que queramos que hagan al ser llamados o ejecutados. Por otro lado, tenemos también las clases que existen en la librería *Unity Engine*, la cual nos viene importada por defecto. Una de estas clases se llama *Resources*. Esta clase será encargada de relacionarnos con el contenido de la carpeta de *Assets*. Como hemos dicho anteriormente, y gracias al comando *Load("objeto")* o *LoadAll("ruta_carpeta")*, cargaremos el objeto o todos los objetos que existan en la *ruta_carpeta*. Estos datos se guardarán o en un *array* (lista) o en un objeto. En caso de querer obtener un objeto de un tipo específico, podremos especificarlo mediante *Load<Texture2D>("ruta_objeto")* o *LoadAll<Texture2D>("ruta_carpeta")*. Según lo mencionado anteriormente, los archivos que queramos obtener tendrán que estar en la carpeta de *Resourcres*, por lo que la ruta tendrá que ser desde dicha ruta (Ej: "*Resources.Load<GUISkin> ("GUISkin/FraseCorrecta/skin");*").

Por otro lado, en lo referente a la base de datos interna se ha utilizado *SQLite*, la cual, se puede crear la base de datos esta manualmente con cualquier programa de bases de datos (en este caso se ha utilizado *DB Browser*). En este caso, se ha utilizado el programa *DB Browser* para generar la base de datos y sus tablas. Esta base de datos hay que guardarla en la carpeta *StreamingAssets*, dentro de la carpeta de *Assets*. Esta carpeta se utiliza para guardar los datos que no se van a instanciar dentro de Unity3D (como ficheros). Entre estos datos encontraremos nuestra base de datos. El motivo de guardarla en esta carpeta, y no en el directorio de *Resources*, como los demás archivos a los que se acceden por código, es por la conexión y la copia. Para conectarse a una base de datos interna desde un dispositivo móvil *Android*, hay que crearla en la memoria interna del dispositivo. Esta creación o copia se debe hacer una vez instalado el juego en el dispositivo. Por otro lado, para copiar archivos, estos tienen que estar en la carpeta anteriormente mencionada. Esto ocurre porque es la única carpeta a la que se puede obtener mediante rutas relativas (“*Application.persistentDataPath+‘/database.sqlite’*”). La diferencia de estos dos directorios se descubrió debido a que, en un principio, para comprobar que cada minijuego funcionaba como debía se realizaba mediante el propio Unity. El problema surgió a la hora de generar el instalador de la aplicación (*apk*) para probarla en dispositivos reales. Este problema apareció debido a que ni la librería que se utilizaba de *sqlite*, ni la localización de la base de datos funcionaban correctamente, dando un error de inexistencia de la base de datos. Para solucionar esto, se tuvo que investigar más a fondo los tipos de carpeta y librerías existentes para utilizar bases de datos internas con dispositivos móviles en Unity (en el ordenador, la librería utilizada en un principio funcionaba a la perfección).

A diferencia que a un proyecto basado en Java, en Unity no existe ningún archivo de propiedades (*.properties* de java). Estos archivos sirven para acceder a texto, por ejemplo, teniendo localizado todo el texto en el mismo archivo. Debido a esto, la forma en la que se puede realizar una función similar es creando una clase con un único elemento de tipo *Dictionary* (equivalente al *HashMap* de java, el cual es una lista con claves y valores, a la cual se accede más rápido mediante las claves). Este elemento tendrá que ser inicializado en una constructora, metiéndole todo el texto que queramos ver en la aplicación. En este caso, al tratarse de una aplicación multiidioma, en vez de ser un elemento de tipo *Dictionary*, es un *array* de *Dictionary* de tal forma que la primera posición del *array* contendrá el diccionario en castellano, y el segundo en euskera.

1.3. **Minijuego**

Para realizar el videojuego, se empezó, como se ha mencionado anteriormente, por los minijuegos (juegos individuales). Esta elección la lleve a cabo debido a que era la parte más difícil dentro del propio videojuego. Además, una vez hechos todos, el paso de la escena del mapa con los niveles a cada minijuego sería más sencillo y global en vez de individualizado.

Debido a que todos los minijuegos iban a tener una estructura similar (título, botón de opciones y botón de ayuda), era recomendable tener una clase encargada de realizar dichas acciones. Para esto, se creó la clase minijuego, de la cual extenderán todos los minijuegos creados. Esta clase, como ya se ha mencionado, era la encargada de visualizar el título del minijuego, el botón de ayuda y el botón de opciones, junto con todas sus funcionalidades. Además de estas interfaces, esta clase también se encarga de gestionar el tiempo y las vidas que le quedan al usuario (mostradas como llaves).

Para lograr esto, primero se creó una lista de objetos de tipo Rect, los cuales son inicializados mediante la posición en la que se quieren poner relativa al tamaño de la pantalla (Ver Figura 27). Además, se guardará tanto la anchura, como la altura del título y la altura restante (altura total – altura del título) en variables protegidas para poder acceder a ellas. Estas inicializaciones se realizan para evitar el constante uso de memoria para calcularlas durante la representación de los objetos en pantalla.

```
rectOpciones = new Rect[14];
rectOpciones [0] = new Rect (0, 0, Screen.width, Screen.height);
rectOpciones [1] = new Rect (Screen.width / 3, Screen.height / 12, Screen.width / 3, Screen.height / 9); //title
rectOpciones [2] = new Rect (5 * Screen.width / 6, Screen.height / 10, Screen.width / 10, Screen.height / 9);
rectOpciones [3] = new Rect (Screen.width / 3, Screen.height / 10, Screen.width / 3, 8 * Screen.height / 10);
```

Figura 27: Inicialización de Rect

Una vez logrado posicionar los diferentes elementos, el siguiente paso es el de inicializar las vidas (de la misma forma que los demás elementos), y de introducir el tiempo. En el caso de las vidas, existe una lista de enteros (puntuación_restantes), la cual será de tamaño tres, e inicialmente con el mismo valor en las tres posiciones. Esta variable servirá para cambiar el número de vidas disponibles. Esto se hace comprobando si para cada llave a pintar, su posición en la lista es mayor de cero o no (en este caso, no se pintará). En el caso del tiempo, debido a la falta de una clase *Timer* en *C#*, se ha utilizado el método *InvokeRepeating*. Este método llama a la función pasada como primer parámetro, desde el segundo pasado como segundo parámetro, con unas repeticiones del tercer parámetro (Ver Figura 28). En este caso, el método que es llamado solo aumentará la variable tiempo en uno en caso de que el minijuego no haya terminado ni esté en pausa. Además, para evitar posibles "trampas" con el tiempo (poner en pausa cuando va a cambiar de segundo), la variable tiempo contendrá los milisegundos transcurridos.

```
InvokeRepeating ("aumentarTiempo", 0f, 0.01f);
```

Figura 28: InvokeRepeating

Por último, se ha añadido un método para restar puntuación (la cual restará un punto a la última posición del array con un número mayor a 0) y el menú de opciones. Este menú contiene un botón para continuar, otro botón para reiniciar el nivel, y un último botón para abandonar el minijuego. A su vez, al estar este menú activo, el contenido del minijuego desaparecerá y el tiempo dejará de aumentar como ya se ha mencionado. Además de todo esto, y de cara a su utilización en los minijuegos, se ha creado también un método para oscurecer las imágenes. Esto se hace variando a 0.2 el campo *alpha* de la gama *rgb* del color, el cuál es el nivel de transparencia.

Una vez hecha la clase Minijuego, lo siguiente era realizar cada minijuego. Para esto, en un principio se hicieron 15 minijuegos diferentes, seleccionados de los 21 que se habían elegido anteriormente (Esta subselección se hizo porque muchos de estos 21 minijuegos eran similares). De estos 15, debido a diferentes problemas que ya se mencionarán más adelante, se optaron por realizar 7. A continuación se explicará detalladamente en qué consiste cada minijuego y como se ha implementado cada uno, los cuales, como ya se ha dicho, estarán heredando la clase *Minijuego,* y, por tanto, al comienzo de los métodos *Start*, *Update* y *OnGUI* explicados anteriormente, harán una llamada al mismo método de Minijuego mediante "base." y el nombre del método (el equivalente al *super()* de *Java*).

Debido a que la base de datos se creó una vez terminada la primera parte de la programación del videojuego, a la hora de crear los minijuegos se trataron todos los campos como texto, en vez de texto e imágenes (a pesar de que a continuación se explique cada minijuego con las imágenes integradas). Se decidió hacerlo así en vez de metiendo imágenes a mano porque la utilización de texto o imágenes en cualquier componente *GUI* es la misma. Esto es porque estos componentes se forman mediante la introducción de su *Rect* (posición y tamaño en pantalla), el contenido y un estilo. El contenido es el encargado de decir que es lo que se va a ver, y puede ser de tipo texto, textura o *GUIContent*. Este último es la combinación de imagen y texto. Además, si se utilizaba imágenes desde el principio se complicaba el trabajo debido a que se tenían que crear o buscar, y cargarlas en el *script*, lo cual luego se tendría que deshacer para cargarlo de base de datos y sería trabajo en vano.

1.3.1. Buscar

El objetivo principal de este minijuego es la de buscar y seleccionar las palabras o letras que están en una caja de contenido, y que coincidan con la que se encuentra encima de dicho cuadro. (Ver Figura 29)

Figura 29: Busca

Para lograr este objetivo, el primer paso fue, al igual que con la clase padre, posicionar todos los elementos. Esto se hizo de la misma forma, pero en diferentes posiciones y haciendo uso de los atributos creados anteriormente (*width*, *altura_titulo* y *altura_restante*). En este paso, la única diferencia respecto a la clase minijuego es que la altura y anchura de cada elemento dependerá del número de elementos que haya, los cuales, variarán respecto a los datos del usuario que se gestionarán más adelante.

El siguiente paso será crear una matriz de texto (*string*) y otra de booleanos, de tamaño de la tabla que se creará (altura X anchura). La primera matriz se inicializará mediante una elección aleatoria (mediante la clase *Random*) de entre las posibilidades. En el caso de la segunda, será inicializado a false. Estas dos matrices serán las responsables de determinar tanto lo que pone en cada casilla, como si se ha pulsado correctamente una casilla o no. Además de las matrices, se guardará en una variable de tipo *string* la opción correcta.

Por último, dentro del método *OnGUI* se dibujarán todos los elementos. Estos elementos estarán compuestos por cajas (*GUI.Box*) o botones (*GUI.Button*), dependiendo de si ha sido pulsado o no (esto se comprobará mediante la matriz *pulsado*). En caso de los botones, al pulsarse se comprobará si su contenido es el mismo que el de la variable correcto. En caso afirmativo, se comprobará si todos los elementos correctos han sido pulsados (recorriendo la matriz *pulsado*). En caso negativo, se llamará a la función “restar puntuación” creada en la clase *Minijuego*, la cual quitará un punto a la puntuación restante en caso de que esta sea mayor a 0. Por otro lado, en el caso de las cajas, se comprobará si es correcto o no el valor que contiene, y utilizará el GUISkin (estilo) correspondiente (verde para correcto y rojo para incorrecto). Todos estos elementos solo se mostrarán, como ya se ha mencionado anteriormente, si la variable “opcionesOn” de la clase padre (Minijuego) está a *true*.

1.3.2. Frase Correcta

El objetivo de este segundo minijuego es el de seleccionar para cada apartado la frase que mejor encaje con la imagen mostrada. (Ver Figura 30)

Figura 30: Frase Correcta

En este caso, el contenido del minijuego está formado tanto por imágenes como por texto. Las imágenes, al igual que las frases de cada fase, estarán almacenadas en un *array* de *Map* (en este caso, de la clase *Dictionary*). A su vez, este *Map* contendrá como clave, en modo *string*, si se trata de una opción, del enunciado (la imagen) o de la solución. En el caso del valor, será un *array* de objetos. El motivo de hacerlo de objetos, en vez de *string*, es porque esta lista podrá contener tanto imágenes (en el caso del enunciado), como texto (en caso de las frases) o como número (en caso de la solución). En caso de la solución y del enunciado, su lista será de un único elemento, pero en caso de las opciones, esta será el número de opciones que contiene la fase. Las frases de las diferentes fases, al igual que la solución, fueron introducidas a mano en un principio, ya que no existía todavía una conexión contra la base de datos. Por otro lado, el campo de la solución contendrá la posición del elemento de opciones que es el correcto.

Una vez introducidos los datos, el siguiente paso es el de la creación de los *Rects*. Esto se hizo al igual que en el minijuego *Buscar*, pero posicionando los nuevos tamaños y posiciones de los elementos en sus correspondientes sitios.

En este caso, la escena está formada por una caja, que contiene la imagen; un área donde se situarán las diferentes frases, y dos botones en la parte inferior. Las frases, a su vez, serán botones. Estos botones, al ser pulsados, se añadirá su posición a una matriz de "seleccionados", y en caso de existir algún elemento en la posición de la fase en la matriz "mal", este se eliminará. La posición será la correspondiente al *array* inicial, y a la que obtiene dentro del *array* de opciones (dentro del *Map*). Junto con esto, cada elemento se pintará en la función *OnGUI* con el *GUIStyle* correspondiente (rojo si está mal, verde si está seleccionado y transparente si no lo está).

Por último, están los botones inferiores. El primero botón se encargará de cambiar a la anterior fase en caso de que no sea la primera. Por otro lado, el segundo botón se encargará de lo contrario, avanzar a la siguiente fase. Además, esté segundo botón también será el encargado de comprobar si las soluciones elegidas son correctas. Para ello, habrá que posicionarse en la última fase (donde nos aparecerá con el nombre de "Terminar" en vez de "Siguiente" si en todas las fases se ha seleccionado alguna opción) y pulsarlo. Una vez hecho esto, se comprobará si hay alguna fase con algún error.

Para comprobar la opción seleccionada en todas las fases, primero se pondrá a -1 las posiciones de todos los elementos del *array* "mal". Una vez hecho esto, se pondrá la variable de la posición actual con el valor -1 y se recorrerá el *array* donde estaban guardadas las posiciones de los elementos seleccionados, comprobando si dicha posición es la misma que la del campo solución del *Map*. En caso negativo, se introducirá en la matriz de elementos incorrectos, se pondrá a -1 de la matriz de seleccionados, y se introducirá en la variable de la posición actual la posición de la fase en la que está el error (Ver Figura 31). Esta variable será la encargada de mostrarnos en todo momento el contenido de la fase en la que nos encontramos.

```
int comprobarCorrecto ()
{
    for (int i = 0; i < mal.Length; i++) {
        mal [i] = -1;
    }
    for (int i = 0; i < seleccion.Length; i++) {
        if (seleccion [i] == -1) {
            if (pos == -1)
                pos = i;
        } else {
            if (seleccion [i] != (int)listaDatos [i] ["Solucion"] [0]) {
                mal [i] = seleccion [i];
                seleccion [i] = -1;
                if (pos == -1)
                    pos = i;
            }
        }
    }
    return pos;
}
```

Figura 31: Comprobar Frase Correcta

Una vez terminada la comprobación, en caso de que dicha variable siga con el valor -1 se llamará al método terminar de la clase padre al igual que en el minijuego *Buscar*. En caso contrario, se posicionará al jugador en la posición de la fase de la variable y se llamará a método mencionado anteriormente de *restarPuntuacion*.

1.3.3. Laberinto Erróneo

En este tercer minijuego, tenemos el objetivo de llegar a la llave por un laberinto en el que, en un principio, solo se ve la casilla inicial. Para visualizar las casillas, hay que ir tocando las casillas en las que la flecha apunte en la dirección que pone en la palabra que se encuentra en la casilla. (Ver Figura 32)

Figura 32: Laberinto Erróneo

Debido a que este minijuego casi no necesita de ningún dato externo (solo el tamaño del laberinto), es el único minijuego que realmente se realizó completamente en esta primera fase. Para lograr este objetivo, lo primero que se necesita es obtener el tamaño del camino. Para esto, se obtiene un número aleatorio entre el tamaño al cuadrado entre 3, y el tamaño al cuadrado entre 4 (Ver Figura 33). Además, también se introducirán en un *array* las diferentes direcciones (en el idioma que esté el usuario) y en otro *array* las flechas.

```
tamanoCamino = UnityEngine.Random.Range (Mathf.FloorToInt (tamano * tamano / 3), Mathf.RoundToInt (tamano * tamano / 4));
```

Figura 33: Tamaño Camino

Una vez obtenido el camino, comienza la posición de las flechas y palabras sobre el tablero. Para ello, mediante un bucle, se irá obtenido la posición siguiente del camino aleatoriamente. Esta aleatoriedad se realizará entre las opciones posibles que existan en el momento. Para obtener estas opciones, se comprobará si la posición actual es un borde o no, y si las casillas de alrededor están ocupadas. Por otro lado, si no existe ninguna opción posible, se reiniciará este bucle y se volverá a intentar calcular de nuevo el camino. Además, dependiendo a que casilla se mueva, se elegirá la palabra y la flecha correspondientes, o la llave o casa. En caso de ser la casilla inicial, se pondrá a *true* su posición en una matriz de las casillas activas. Por el contrario, si se trata de la casilla objetivo, se guardará en una variable su posición. Por último, en caso de que la casilla destinada para la llave (la última casilla) sea una casilla colindante con la inicial (su casilla superior, inferior, de la derecha o de la izquierda), se reiniciará el bucle. En caso de reiniciarse el bucle, se reiniciará tanto la casilla solución (poniéndola a -1) como todas las matrices creadas (activos, camino y contenido).

Una vez obtenido el camino, se procederá a dar valor a todas las casillas restantes. Para esto, se les dará un valor aleatorio tanto en imagen como en texto. En caso de que una casilla tenga el mismo valor (la flecha apunta en la dirección de la palabra), se volverá a calcular dicha casilla. Esto se hace para evitar posibles confusiones a la hora de jugar, de elegir las casillas correctamente, pero no ser parte del camino correcto.

Con el contenido de cada casilla ya posicionado, se dibujará en pantalla el laberinto, mostrando únicamente aquellas casillas en las que su posición en la matriz activos sea *true*, los cuales se mostrarán como botones (*GUIButton*). Las demás casillas, las que aparecen vacías, se mostrarán como cajas (*GUIBox*), las cuales no se pueden ser pulsadas (Esto se ha hecho para prevenir posibles fallos involuntarios). En caso de pulsarse en una casilla mostrada (un botón), se comprobará si dicha casilla es una solución o no. En caso afirmativo, al igual que en los otros minijuegos, se llamará al método *terminarJuego*. En caso contrario, se comprobará si es la casilla inicial o la imagen se corresponde a la palabra (obteniendo y comprobando la posición de ambos en los *arrays* iniciales de *textoPosible* y *imagenesPosibles*). En caso negativo, se restará la puntuación de la misma forma que en los otro minijuegos. En caso afirmativo, se mostrará la casilla superior, inferior y laterales, en caso de que no estén en el borde.

1.3.4. Palabras Incompletas

En el cuarto minijuego, hay que rellenar las letras que faltan en cada palabra. Para saber qué palabra es, encima de esta se muestra una imagen de la misma. (Ver Figura 34)

Figura 34: Palabras Incompletas

Antes de comenzar a realizar los mismos pasos que en los anteriores minijuegos, en este minijuego habrá que descomponer los elementos. Como las palabras que se obtendrán de la base de datos serán palabras completas, antes de incluso decidir los tamaños y posiciones de los diferentes elementos, deberemos descomponer las palabras en sus letras y convertirlas en mayúsculas (esto se hace para que todas las letras sean del mismo tipo). Para esto, al igual que en el minijuego "Frase Correcta", crearemos un *array* de *Map*. En este caso, las claves de cada *Map* serán 'Palabra' (contendrá un *array* de letras, formando la palabra correcta), 'Imagen' (contendrá un *array* de un elemento, con la imagen de la palabra), 'Mostrar' (contendrá un *array* con las letras que se van a mostrar) y 'Relleno' (contendrá una lista de booleanos para comprar si es una letra a modificar, originalmente con'_' escrita, o es una letra fija).

Una vez se tenga todas las palabras, al igual que en anteriores minijuegos, se han creado los diferentes *Rect* para posicionar los diferentes elementos en pantalla. Para hacer esto, se han dividido los elementos en 2, para que la mitad de las palabras e imágenes se encuentren en la primera final, y la segunda mitad en la segunda. Por otro lado, todas las letras posibles a introducir están guardadas en un *array*. Además, el tamaño de todas las letras (de las palabras y las propias letras a elegir) dependerán del tamaño de las palabras. De esta forma, se cambiará el tamaño de los estilos utilizados.

Adentrándonos en la función *OnGUI*, se mostrará un botón en caso de que la letra de la palabra sea true en la clave 'Relleno' del *Map*. Por otro lado, se mostrará una caja en caso contrario. Además, en caso de que la variable llamada *seleccionado* (la que contendrá la posición del elemento seleccionado – x = la palabra, y = la letra) sea distinta a -1, se mostrarán los botones laterales de las letras y la letra de la palabra correspondiente con un fondo verde (cambio de estilo). Por otro lado, al completar todas las palabras se mostrará un botón para comprobar la solución.

A la hora de comprobar la solución, se compara cada letra de cada palabra del apartado 'Mostrar' del *Map*, con el apartado 'Palabra'. En caso de que sean diferentes, se añade la posición en un *array* llamado 'mal'. Una vez terminada la comprobación, si este *array* está vacío, es decir, no ha habido ningún error, se llamará a la función *terminarJuego*, al igual que en los anteriores minijuegos. En caso contrario, se restará la puntuación, y a la hora de pintar se dibujará dicha letra con un color rojo de fondo (cambio de *GUIStyle*). Debido a esta comparación, las palabras no dispondrán de ningún acento y serán mostradas todas en mayúsculas (pueden estar guardadas con acentos y minúsculas, pero se transformarán a nuestras necesidades).

1.3.5. Parejas

En este otro minijuego, hay que seleccionar las cartas o casillas que estén blancas para hacer parejas. Las parejas pueden ser de imagen y texto, o de texto y texto. (Ver Figura 35)

Figura 35: Parejas

Referente al tema de la implementación, este minijuego ha resultado el más sencillo de todos, ya que no tiene gran cantidad de contenido.

Antes de gestionar los datos, se ha obtenido el tamaño de las filas y de las columnas. Esto se hace para dejar todos los elementos ordenados y quepan lo mejor posible en la pantalla. Para lograrlo, se comprueba si el total de datos (los datos multiplicados por 2, ya que hay dos casillas por dato) es de tamaño cuadrático. Para esto, se multiplica la longitud del *array* de datos por 2, y se comprueba si su raíz cuadrada tiene resto o no. En caso afirmativo, se pone tanto al tamaño de la fila como el tamaño de la columna dicha raíz cuadrada. En caso contrario, se pone a la fila el número entero máximo entre el tamaño total entre su raíz cuadrada, y la raíz cuadrada del tamaño total. En el caso de la columna, se pone el mínimo. (Ver Figura 36)

```
int tamano = listaDatos.Length * 2;
if (Mathf.Sqrt (tamano) % 1 == 0) {
    tamanoX = (int)Mathf.Sqrt (tamano);
    tamanoY = (int)Mathf.Sqrt (tamano);
} else {
    int numSqrt = (int)Mathf.Sqrt (tamano);
    int numDiv = Mathf.CeilToInt (tamano / numSqrt);
    tamanoX = Mathf.Max (numSqrt, numDiv);
    tamanoY = Mathf.Min (numSqrt, numDiv);
}
```

Figura 36: Parejas Calcular tamaño

A la hora de gestionar los datos, estos son introducidos en una matriz de *Map*. El motivo por el cual es una matriz, y no un *array*, como en los anteriores minijuegos, es porque en este caso, hay que desordenar los elementos, por lo que, si se ponen en el mismo *Map* no se podría desordenar. Por lo cual, en este caso, las claves serán 'Valor' (contendrá la imagen o el texto a mostrar) e 'Identificador' (atributo para relacionar los diferentes componentes de la matriz).

El siguiente paso es el de obtener las medidas de los *Rect*, las cuales dependerán del tamaño de fila y de columna calculado anteriormente.

Como se ha dicho en los párrafos anteriores, el siguiente paso es el de desordenar. Para realizar esto, se ha creado una nueva matriz del mismo tipo que la de los datos, pero con el tamaño de fila y columna hallados anteriormente. A continuación, se recorrerá esta nueva matriz, y en cada campo, se seleccionará aleatoriamente un campo de la otra matriz. Para esto se seleccionará dos números aleatorios de entre sus medidas (x e y de la matriz). Si el elemento seleccionado es vacío (*null*) en el campo 'Valor', se volverá a intentar. Una vez añadido el elemento, se pondrá a null el campo valor de la matriz anterior en la posición de la que se ha cogido los datos. En el caso de ser una imagen, se transformará a las medidas que tiene que ocupar mediante la clase *TextureScale* (Ver Figura 37)

```
TextureScale.Point ((Texture2D)aux [i] [j] ["Valor"], (int)rect [i] [j].width - 20, (int)rect [i] [j].height - 20);
```

Figura 37: Redimensionar imagen

Por último, tenemos, como en los anteriores minijuegos, la función *OnGUI*. En este caso, se comprobará si cada elemento se está mostrando (marcado por una matriz de booleanos). Si es el caso, se dibujará el elemento en una caja. Por otro lado, se comprobará si el elemento está pulsado o no. En caso afirmativo, se pondrá la imagen o texto correspondiente. En caso contrario, se pondrá un texto vacío (""). Lo último que queda es saber qué ocurre cuando pulsamos sobre una casilla. Una vez pulsado, se comprueba el estado de la casilla y puede pasar diferentes casos:

- La casilla estaba pulsada → Se despulsa, es decir, deja de ser visible lo que había. (borra su posición de una matriz 'pulsado').
- Había alguna casilla pulsada (Guardada su posición en la matriz 'pulsado') → Se comprueba si tienen el mismo identificador. En caso de ser diferente, se restará puntuación como en los anteriores minijuegos, y se guardará el tiempo actual del minijuego en una variable. Este tiempo se utilizará para, mediante el método *Update* (que se ejecutaba cada *frame*) comprobar si ha pasado un segundo. Cuando pase el segundo, se dejarán de ver ambas casillas y se borrarán sus posiciones de la matriz 'pulsado'. En caso de ser iguales, se pondrá a *true* sus posiciones en la matriz 'mostrado', se oscurecerá tanto la imagen como el texto con la función oscurecer de la clase Minijuego mencionada anteriormente, y se comprobará si se ha completado el minijuego (comprobando si esta matriz tiene todos los campos a *true*). En caso de haber terminado, se llamará a la función *terminarJuego* de la clase padre.
- No había ninguna casilla pulsada → Se guarda la posición de la casilla en la matriz 'pulsado'.

1.3.6. Selecciona

En el penúltimo minijuego, el objetivo es el de seleccionar, para cada apartado, la imagen que mejor encaje con cada frase que se muestra. (Ver Figura 38)

Figura 38: Selecciona

Este minijuego se puede decir que es similar al minijuego Frase Correcta. Por eso, se decidió hacerlo de una forma diferente, es decir, utilizar un *scroll* (desplazamiento) para ver las diferentes fases en vez de botones.

La obtención de los datos es similar al minijuego mencionado en el párrafo anterior, con la diferencia de en vez de utilizar como enunciado una imagen y como opciones texto, se hace al revés.

En este caso, el tema complicado es el del *scroll*. Para explicarlo, primero se comentará la función *OnGUI*.

En *Unity3D*, para poner *scroll* a trozo de ventana, es necesario crear un área (*GUILayout.BeginArea(rectArea)).* Dentro de esta área se deberá introducir un *GUILayout*. Este tipo de elemento es donde se introducirán todas las vistas que queramos que tengan *scroll*. Además, se creará una vista con *scroll* vertical (*GUILayout.BeginScrollView*). Esta función necesita como parámetros tanto la posición en el *scroll* (scrollPosition), como la anchura y largura que va a ocupar y si se puede expandir en caso de que el contenido sea de mayor tamaño (En la Figura 39 se ha puesto en dos líneas para una mejor visualización).

```
scrollPosition = GUILayout.BeginScrollView (scrollPosition, GUILayout.Width (rectArea.width),
GUILayout.Height (rectArea.height), GUILayout.ExpandWidth (false), GUILayout.ExpandHeight (false));
```

Figura 39: BeginScroll

A continuación, se han creado grupos verticales (*GUILayout.BeginVertical(skin.window)*) y horizontales (*GUILayout.BeginHorizontal (skin.window)*) para organizar todos los elementos. Estos dos grupos son necesarios, ya que a los elementos dentro de un área no se les puede dar una posición, sino que solo un tamaño (y se adaptan al espacio). Por lo que ésta es la única forma de cuadrar los elementos para mostrarlos donde los necesitamos. Los demás elementos, como los *GUIStyles* y comportamientos al pulsar un botón (meter la posición en el *array* de selección) son iguales que en el otro minijuego similar, por lo que no se va a volver a explicar. Por último, y ya fuera del *scroll*, se encuentra el botón de comprobar. Este botón sólo se verá, al igual que en el minijuego *Frase Correcta*, cuando todas las fases tienen algún elemento seleccionado.

El otro tema a mencionar es el *scroll*. A diferencia de en el ordenador, en los dispositivos móviles es un tanto incómodo el tener que hacer *scroll* sobre una vista arrastrando su *scrollBar*. Por esto, se ha tenido que implementar un método, el cual es llamado desde el *Update*, y se encarga de esta ardua tarea. Este método, llamado *tocar*, comprueba si se ha tocado la pantalla. En caso de tocarla, se obtendrá su fase (*TouchPhase*), y se comprobará si se mueve el dedo en alguna dirección. En caso de moverse, el estado cambiará de *TouchPhase.Began* a *TouchPhase.Moved*. En este momento de moverse, se comprobará cuanto se ha movido, y en caso de ser mayor a 40, se moverá a la siguiente fase o a la anterior fase, dependiendo de qué movimiento se realice. Para realizar este movimiento, se le suma al *scrollPosition* (que es la posición del *scroll*) la altura del área de *scroll* + 20. Este 20 es una cantidad utilizada para no solaparse las fases entre sí. (Ver Figura 40)

```
void tocar ()
{
    foreach (Touch touch in Input.touches) {
        if (touch.phase == TouchPhase.Began) {
            origen = new Vector2 (touch.position.x, Screen.height - touch.position.y);
            movido = false;
        } else if (touch.phase == TouchPhase.Moved) {
            if (!movido && touch.position != origen) {
                movido = true;
                if (touch.position.y > origen.y + 40 && pos < datos.Count - 1) {
                    scrollPosition.y += (rectArea.height + alturaScroll - 20);
                    pos++;
                } else if (touch.position.y < origen.y - 40 && pos > 0) {
                    scrollPosition.y -= (rectArea.height + alturaScroll - 20);
                    pos--;
                }
            }
        } else if (touch.phase == TouchPhase.Ended) {
            movido = false;
        }
    }
}
```

Figura 40: Selecciona tocar

1.3.7. Sopa de letras

El último minijuego se trata de una sopa de letras. Su objetivo es el de buscar en la sopa de letras las palabras que corresponden a las imágenes o palabras que aparecen en los laterales. (Ver Figura 41)

Figura 41: Sopa de Letras

Entre todos los minijuegos, puede que este haya sido el más complicado de implementar debido a que, para poder arrastrar y seleccionar las diferentes letras por la pantalla, las letras no podían ser botones, si no que tenían que controlarse los toques en pantalla (*touch*), al igual que en el minijuego *Seleccionar*.

Al igual que en otros minijuegos, este minijuego se comienza por la obtención de datos. Estos datos, es decir, las imágenes y el texto, se guardarán en dos *arrays*. Para los elementos de los laterales, se utilizará el *array* 'listaMostrar'. Como este array tiene que estar formado tanto por imágenes como texto, tiene que ser de tipo *Object*. Además, la elección de mostrar imágenes o texto depende de si los datos que tiene contienen imágenes o no. En caso de contener ambos, se realizará una elección aleatoria para decidir cuál de los datos mostrar. Por otro lado, está el array de las palabras que se van a encontrar en la sopa de letras. En este caso, será un *array* de *string*. El texto de este *array* será convertido a mayúsculas para coincidir con las letras de la propia sopa de letras.

Una vez gestionados los datos, el siguiente paso, al igual que en los otros minijuegos, es la creación de los *Rects* y rellenar la Sopa de Letras. En este caso, los *Rects* son la sopa de letra, y los cuadros donde van colocados las imágenes o textos solución. Antes de rellenar la sopa de letras, hay que definir su tamaño. Este será el tamaño de la palabra más larga. Por último, en este tema, se encuentra el relleno de la sopa de letras. Para hacer esto, primero se han introducido en una matriz de caracteres todas las palabras de una forma aleatoria, de tal forma que se puedan solapar entre ellas (una misma letra puede servir para más de una palabra), y puedan estar posicionadas en todas las direcciones (horizontal, vertical y diagonal). La posición se calculará mediante la obtención de dos números (x e y) aleatorios entre 0 y el tamaño. Más tarde, se comprobarán las direcciones en las que puede estar la palabra (dependiendo de los límites y de las letras ya posicionadas), y se seleccionará una aleatoria entre las posibles. Una vez posicionadas las palabras, el siguiente paso es el de rellenar el resto de la matriz. Para ello, al igual que en otras ocasiones, se utilizar la aleatoriedad para coger las letras de un *array* de letras creado inicialmente.

El siguiente paso es dibujarlos mediante la función *OnGUI*. En este caso, no existe ninguna anomalía ni caso especial, ya que, como veremos a continuación, todo se gestiona cuando se toca la pantalla. La única cosa a mencionar sería que no existe ningún botón. Todos los elementos son cajas, y en caso de la sopa de letras, cada casilla tiene su propio *GUIStyle*. Con su propio *GUIStyle* se quiere decir que se ha creado una matriz de tipo *GUIStyle* para indicar cada casilla de la sopa de letras que estilo tendrá (Ver Figura 42).

```
GUI.Box (rectList [i] [j], texto [i] [j], skinMatrix [i] [j].box);
```

Figura 42: Sopa de Letras OnGUI

Por último, nos encontramos con la parte más complicada de esta primera fase, el controlar que parte de la pantalla se toca y cómo actuar en cada movimiento. Para ello, lo primero que se comprueba nada más se toca la pantalla es si es una casilla. Esto se comprueba en el *TouchState.Begin* (mencionado en el minijuego *Seleccionar*). Para comprobar esto, y que casilla es, se utiliza la matriz de *Rects* de la sopa de letras creada anteriormente. Se utiliza el tamaño y posición de cada casilla para comprobar si se el punto de la pantalla que se ha tocado está entre la posición de la casilla y la posición + el tamaño (tanto en altura como anchura tiene que coincidir). Tras identificar la casilla, el siguiente paso es el de comprobar que se ha salido de la casilla tras entrar en el *TouchState.Moved*, es decir, tras mover el dedo por la pantalla. Para esto se comprobará de forma similar al de presionar, pero comprobando si sigue siendo la misma casilla al final.

Una vez sabido que se ha movido de casilla, se volverá a entrar en el método *presionar*, pero esta vez, además de comprobar si está en una casilla (porque esto ya se supone al comprobar que se ha cambiado) se comprobará la dirección que se está tomando. Para esto, se comprobará si existe algún elemento en la matriz del recorrido (posiciones de las casillas tocadas). En caso de no existir, estaríamos en el estado que acabamos de tocar la pantalla, por lo que se añadiría esta casilla a la matriz de recorrido, se pondrá su posición en la matriz actual (matriz para saber cuál es la última casilla seleccionada) y cambiaría la matriz de skin en esta posición por la correspondiente a una casilla seleccionada ("*skinMatrix[i][j]=skinList[1]*"). En caso de que la matriz de recorrido no estuviera vacía, se comprobará si esta nueva casilla está en la matriz, obteniendo su posición si es así (si está, quiere decir que estamos borrando la casilla o echando hacia atrás el recorrido). En caso de que exista, se eliminará la última casilla de la matriz y se cambiará el *GUISkin* al que tenía antes dicha casilla. Además, se cambiará la casilla actual a la anterior. En caso contrario, se comprobará si sigue la misma dirección (para no hacer giros a la hora de seleccionar). Esta comprobación se hace observando las posiciones de las últimas casillas de la matriz recorrido. De esta forma, se comprueba si el último movimiento es horizontal, vertical o diagonal y si coincidía con el realizado. Si todo ha ido bien, realizará la misma acción que cuando no hay ningún elemento en la matriz recorrido.

Por último, cuando se deje de pulsar la pantalla se comprobará si la palabra formada por las letras de las casillas seleccionadas existe entre las opciones. Para esto, se formará la palabra, y se comprobará con las existentes en el array de *listaPalabras*. En caso de existir, se pondrá como que está hecha dicha palabra, se oscurecerá la imagen o el texto con la función de la clase Minijuego, y se cambiará el estilo por el referente a correcto. Además, se añadirá las casillas a una nueva matriz booleana para, en caso de seleccionar dichas letras de nuevo, al soltar el botón no se borre su estilo. Como es de esperar, en este paso se comprobará si se han encontrado todas las palabras mediante el *array listaPalabras*, y en dicho caso, se llamará a la función *terminarJuego* del padre. En caso de no existir, se restará puntuación y se volverá a cambiar los estilos de las casillas por los que tenían antes (haciendo uso de esa matriz mencionada).

2. Base de Datos

Una vez diseñada y organizada la parte conflictiva de la base de datos (la parte de cómo gestionar los datos de los minijuegos), se decidió diseñar la base de datos. Esta parte, debido a que se tenían dos bases de datos (remota y local), se dividió en dos.

2.1. **Remota**

Para realizar la base de datos remota, se ha utilizado una base de datos *MYSQL*. Respecto a esta parte, se encontraron ciertas limitaciones. Una de ellas fue en la información almacenada del paciente. Esta información no se sabía de qué longitud podría ser, por lo que se puso su tamaño reservado en base de datos lo más grande posible. Como el tipo de dato sería VARCHAR, no importaba el hecho de tener espacio sin utilizar, ya que este tipo de dato se adapta a la información que contiene en su interior. El problema surgió en el caso contrario, en el tamaño máximo. Esta longitud máxima se fijó a 500 para evitar la falta de espacio en caso de que se quisiera introducir mucha información en cualquier campo. Por otro lado, en una primera instancia, en el diagrama entidad relación se puso toda la información del paciente en la misma tabla, para evitar un número elevado de ellas. Por tanto, debido al elevado número de registros de información (60) y al tamaño máximo de dichos registros (500), al realizar una consulta de todos los campos devolvía un error de tamaño. Este error se podría solucionar modificando el tamaño máximo de las consultas de la base de dato en la configuración del servidor, pero debido a que en la base de datos utilizada no se disponían de los derechos de administrador, esta opción se descartó. La siguiente opción fue la de dividir dichos datos en varias tablas. Estas tablas se cuadraron con las tablas del documento de Anamnesis (Nino, Anamnesis_Nino, Autonomía_Nino, Socialización_Nino y Lenguaje_Nino).

Ha de mencionarse que en las tablas que contienen imágenes, en esos campos se guarda la dirección de la imagen en el servidor en vez de la imagen. Esta decisión se tomó debido a que, a la hora de sincronizar los datos de la base de datos remota a la local, se demoraba mucho tiempo en descargar u obtener los datos en las tablas con imágenes. En cambio, con las direcciones URL, el tiempo era muy inferior.

Por otro lado, a la hora de diseñar la base de datos se encontró un problema con la relación entre las tablas *Grupos_Minijuegos* y Grupos. Debido a que la tabla *Grupo* tiene como clave primaria el *id_grupo*, el *id_contenido_solucion* e *id_contenido_opciones*, a la hora de querer agrupar en Grupos_Minijuegos por el *id_grupo* no dejaba. Esto es debido a que la clave extranjera (*foreign key*) de una tabla tiene que ser la clave primaria completa de otra. En este caso, no podía ser solo el *id_grupo*. Como solución, se optó por eliminar la clave extranjera entre estas tablas (por eso no aparece ni en el diagrama Entidad-Relación Extendido ni en el diagrama de tablas) y crear un *trigger* para controlar cuando se eliminaba un grupo de la tabla *Grupo* (para eliminarlo también de la tabla *Grupos_Minijuegos*).

Junto con todas estas tablas, también se han creado varios *Triggers* o disparadores. Estos *triggers* se ocuparán de introducir los datos correspondientes en las tablas TS cuando se realice una acción sobre las tablas originales. Por ejemplo, cuando un grupo se modifique, se activará su *trigger* (T_GRUPO_M) y se actualizará tanto la acción (a 'M') como la fecha de modificación (a la actual) del dato correspondiente en la tabla TS_GRUPO (Ver Figura 43).

```
CREATE TRIGGER T_GRUPO_M
BEFORE UPDATE ON GRUPO
FOR EACH ROW
BEGIN
    UPDATE TS_GRUPO
    SET TSG_ACCION = 'M',
        TSG_FECMOD = SYSDATE()
    WHERE TSG_ID_GRUPO = OLD.G_ID_GRUPO
      AND TSG_CONTENIDO_SOLUCION = OLD.G_CONTENIDO_SOLUCION
      AND TSG_CONTENIDO_OPCION = OLD.G_CONTENIDO_OPCION;
END $$
```

Figura 43: Trigger

Por otro lado, debido también a los derechos de administrador, no se han podido gestionar los derechos de acceso de la base de datos. Dentro de estos derechos, se pretendía crear dos usuarios, uno de logopeda y otro de paciente. El primero tendría acceso a todos los datos y podría realizar cualquier operación. En caso del segundo, sólo tendría acceso de modificación (*Insert* o *Update*) a las tablas que se modifican en el videojuego (Historial_Mapas, Historial_Juegos y Premios_Usuarios). En cuanto a los accesos de consulta de tablas, sólo tendría acceso a las vistas, funciones y procedimientos. Gracias a esta restricción, evitaría el poder ver campos protegidos por la Ley de Protección de Datos en caso de conocer la contraseña de este usuario. Además, en el caso del Login o comprobación de usuario y contraseña en el videojuego se hará mediante la función *Login*. Esta función, que tiene como parámetros dos cadenas de texto (usuario y contraseña), comprobará si dichos datos existen combinados. En dicho caso, devolverá el identificador numérico del usuario. En caso contrario, devolverá un error -2.

Por último, pero no por ello menos importante, se encuentra el procedimiento de actualización de la tabla Historial_Mapa. Debido a la posibilidad de tener el videojuego por un mismo usuario en varios dispositivos, pueden llegar a existir problemas de sobrescribir unos datos por otro más antiguos (por ejemplo, si en un dispositivo no se completa un mapa, pero tampoco se actualiza. Entonces en otro dispositivo donde dicho mapa se ha conseguido completar, se sincroniza. Pero si el primer dispositivo se sincronizaría en este momento, se sobrescribiría el historial, dando como resultado que este usuario no se ha pasado el mapa). Este procedimiento comprobará para cada dato que va a insertar, que no exista o que el estado de cada botón guardado en la base de datos no sea *'Pasado'* (si el estado es pasado, nos da igual cuál sea más nuevo). En este segundo caso, se comprobará también si el estado que introducimos es más nuevo que el que reside en la base de datos.

2.2. **Local**

En lo que se refiere a la base de datos local, ésta iría en cada dispositivo móvil, y sería la base de datos desde la que se cogerían los diferentes datos en el videojuego. Se ha utilizado una base de datos *SQLITE*, y los datos serán insertados mediante una sincronización, la cual se explicará en el apartado de la segunda fase del videojuego. Debido a esto, esta base de datos tendrá similares características y tablas a la base de datos remota. Una de las diferencias globales reside en las tablas que contienen imágenes, donde, en este caso, las imágenes se guardan en formato *BLOB* en vez de *VARCHAR* (se guarda la imagen en vez de su dirección URL). Además, para gestionar y visualizar si se estaba insertando correctamente los diferentes datos, se ha utilizado el programa *DB Browser*.

En este caso, esta base de datos no contiene ningún *trigger*. Esto es debido a que los datos que se van introducir, en su mayoría, proceden de la base de datos remota, por lo que no se requiere de ninguna acción en su inserción, modificación o eliminación. Además, a pesar de que deberían estar relacionadas entre sí, estas tablas no tienen ninguna clave extranjera (*foreign key*). Esto fue decidido hacerlo así, debido a que la sincronización iba a ser asíncrona (para obtenerse los datos de todas las tablas al mismo tiempo), por lo que podía ser que se introdujera los datos en las tablas con *foreign key* antes que en las tablas con las que está relacionada dicha *foreign key*, produciendo errores.

Por último, se han creado varias vistas para evitar la repetición de las mismas sentencias SQL en varias partes del código. Estas vistas, en su mayoría, están formadas por la agrupación de varias tablas (por ejemplo, los botones del historial_mapa). Pero hay dos vistas que están en función de los datos existentes. Estas vistas son V_Minijuegos y V_Contenido. La primera vista se encarga de obtener los datos de los minijuegos que se pueden jugar, es decir, que tienen suficientes grupos asignados para poder jugar (Ver Figura 44).

```
CREATE VIEW V_MINIJUEGOS AS
    SELECT M_ID_MINIJUEGO, M_NOMBRE, M_DESCRIPCION, M_PUNTUACION_MAXIMA
    FROM MINIJUEGO
    WHERE (SELECT COUNT(*)
            FROM GRUPO_MINIJUEGOS
            WHERE GM_ID_MINIJUEGO = M_ID_MINIJUEGO) >= M_MINIMO
           OR M_ID_MINIJUEGO IN(1, 3);
```

Figura 44: V_Minijuego

En lo referente a la segunda vista, esta se encarga de devolver todo el contenido referente a un grupo (tanto solución como opciones). Además, se encarga de devolver únicamente el texto del idioma que tiene el usuario asignado para jugar (Por ejemplo, si el usuario tiene asignado por la logopeda para jugar con las palabras en euskera, solo obtendrá dichas palabras) (Ver Figura 45).

```
CREATE VIEW V_CONTENIDO AS
    SELECT G_ID_GRUPO, GM_ID_MINIJUEGO,
            CS.C_ID_CONTENIDO AS ID_CONTENIDO_SOLUCION, CS.C_IMAGEN AS IMAGEN_SOLUCION,
            (CASE WHEN (SELECT U_IDIOMA_JUEGO FROM USUARIO) ='euskera' THEN CS.C_EUSKERA
                ELSE CS.C_CASTELLANO
            END) AS  TEXTO_SOLUCION,
            CO.C_ID_CONTENIDO AS ID_CONTENIDO_OPCION, CO.C_IMAGEN AS IMAGEN_OPCION,
            (CASE WHEN (SELECT U_IDIOMA_JUEGO FROM USUARIO) ='euskera' THEN CO.C_EUSKERA
                ELSE CO.C_CASTELLANO
            END) AS  TEXTO_OPCION
    FROM GRUPO, GRUPO_MINIJUEGOS, CONTENIDO CS, CONTENIDO CO
    WHERE G_ID_GRUPO = GM_ID_GRUPO
      AND CS.C_ID_CONTENIDO = G_CONTENIDO_SOLUCION
      AND CO.C_ID_CONTENIDO = G_CONTENIDO_OPCION;
```

Figura 45: V_Contenido

3. Programa de Gestión

Una vez diseñada y creada la base de datos, el siguiente paso se decidió que fuera el de gestionar el programa de gestión. Como se ha mencionado anteriormente, este programa ha sido hecho en el lenguaje de programación *Java*, y será el encargado de permitir a la logopeda gestionar toda la información de cada paciente, junto con los datos de los diferentes minijuegos (los grupos, mapas, contenidos, permisos, etc.). Para lograr esto, todas las ventanas creadas (a excepción de las de los dos menús) contendrán una clase que extiende de la clase *JFrame* (la cual se llamará 'Ventana*', donde * será el nombre de cada ventana – Por ejemplo, VentanaMinijuegos) y una clase Gestora (llamada 'Gestor*', donde * será el nombre de cada gestor – Por ejemplo, GestorMinijuegos). Gracias a estas dos clases java, y al patrón *Observer* (el cual permite a una clase estar en "vigilancia" de otra, y saber cuándo esta cambia para actuar) se ha conseguido implementar la relación modelo vista en este programa. Además, al ser un programa multidioma (castellano y euskera), todo el texto visible en el mismo no está puesto directamente sobre el código, sino que está gestionado por la clase *IdiomaProperties* que se explicará más adelante.

Por otro lado, en las ventanas en las que aparezcan tablas, estas tablas están creadas mediante un modelo de tabla personalizado (*CustomTableModel*). Gracias a este modelo, se ha podido personalizar la tabla. Esta personalización se refiere desde a editar qué columnas y filas se pueden editar en cada tabla, hasta cómo se van a mostrar los datos en cada columna. Además, junto que este modelo, también se ha utilizado una tabla personalizada (*CustomJTable*) para terminar de personalizar dichas tablas con los filtros de ordenación, o la gestión de los datos dentro de las celdas (por ejemplo, para restringir el tamaño de los *textAreas* dentro de las celdas).

Una cosa que se va a explicar ahora es la ventana de carga. Al acceder a cualquier ventana en la que se necesite una carga de datos (todas a excepción de los menús, *ModificarDatos*, *Login* y *Registro*), mientras se cargan dichos datos, se visualizará una ventana con una barra de carga (Ver Figura 46). Una vez que dicha carga termine, se le notificará a la ventana correspondiente, y aparecerá en la pantalla la ventana con sus datos.

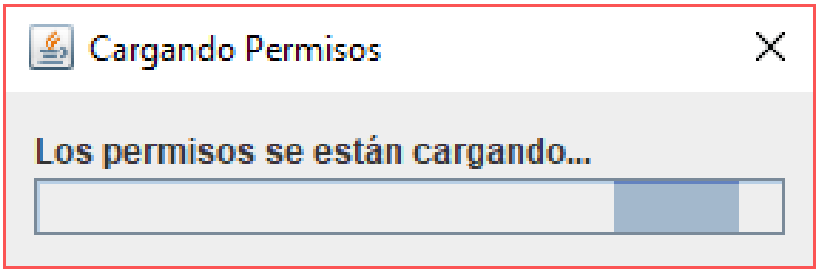


Figura 46: Barra de Carga inicial

A diferencia de lo visto a lo largo de la carrera, en este caso, el acceso a base de datos se realizará de forma diferente. Para hacer esto, se ha realizado gracias a la librería *MyBatis* (conocida mientras se realizaban prácticas de empresa), la cual se explicará a continuación. Después de esto, se explicarán detenidamente todas las ventanas.

3.1. **MyBatis**

Para conocer qué hace esta librería, se podría coger la definición que le dan sus creadores en su página web, "*MyBatis* es un *framework* de persistencia que soporta SQL, procedimientos almacenados y mapeos avanzados. *MyBatis* elimina casi todo el código JDBC, el establecimiento manual de los parámetros y la obtención de resultados. *MyBatis* puede configurarse con *XML* o anotaciones y permite mapear mapas y *POJOs* (*Plain Old Java Objects*) con registros de base de datos" (Maven, 2017). De esta forma, nos evitamos el uso de cursores y sentencias preparadas al acceder a base de datos. En su defecto, se usarán clases *Java*, *mappers* y archivos *XML*. Además, el método de conexión a la base de datos (usuario, contraseña, host y drivers) es configurado mediante un archivo de configuración. Este archivo es guardado dentro del proyecto, aunque, se podría guardar dicho archivo en un lugar fuera del programa (en un *PHP,* por ejemplo), y hacer el programa más seguro. Esto no se ha podido realizar por falta de tiempo

Como se ha dicho en el anterior párrafo, *MyBatis* se compone principalmente de clases *mappers*, *Java* y *XML*, por lo que, a continuación, se va a explicar cada uno de estos elementos:

- *Mappers*: Los *mappers* son las interfaces encargadas de relacionar el código Java con los archivos XML. A su vez, estas interfaces contendrán tantas sentencias como el archivo XML al que están asociados tiene, y con el mismo nombre y parámetros. Estos parámetros la mayoría de veces suelen ser de tipo *Map* o de una clase Java (Ver Figura 47).

```
public interface GrupoMapper
{
    public List<Grupo> getGrupos();

    public int existeGrupo(Map<String, Object> parametros);

    public int esBorrableMinijuego(Map<String, Object> parametros);
    public int esBorrableUsuario(Map<String, Object> parametros);
    public void insertGrupo(Grupo grupo);
    public void insertGrupoUsuarios(Grupo grupo);
    public void insertGrupoNuevo(Grupo grupo);
    public void updateGrupo(Grupo grupo);
    public void deleteGrupo(Map<String, Object> parametros);


    public List<Nino> getPermisos(Grupo grupo);
    public void insertPermisos(Grupo grupo);
    public void deletePermisos(Map<String, Object> parametros);

}
```

Figura 47: Interfaz MyBatis

- *Java*: En lo referente a las clases java, esta librería utiliza dos tipos de "clases". Por un lado están las de cada elemento (Sería las clases 'Grupo' que se ven en la Figura 47), y por otro los *DAO*:
    - Clases Java: Las clases java serán clases de cada elemento, es decir, una clase por cada elemento o tabla que se tiene en la base de datos. De esta forma, esta clase contendrá su constructora, sus atributos (que serán aquellos que recogeremos en las sentencias SQL) y sus *getters* y *setters*.
    - *DAO*: Los *DAO* serán las clases encargadas de utilizar los *mappers* mencionados anteriormente. Estos *DAOs* controlarán que elementos van a ser enviados al *XML*, que ocurre en caso de error (*commit* y *rollback*), y qué hacer con los datos resultantes. Para ello, se tiene que abrir una conexión SQL (*SqlSession*), obtener su *mapper*, y realizar las operaciones pertinentes mediante llamadas a los métodos del *mapper* obtenido. En la Figura 48 podemos observar cómo se realizarían una *select* y una *insert*.

```java
public class GrupoDAO {

    public List<Grupo> getGrupos() throws Exception
    {
        SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();

        try {
            GrupoMapper mapper = sqlSession.getMapper(GrupoMapper.class);
            return mapper.getGrupos();
        }
        finally {
            if(sqlSession != null) sqlSession.close();
        }
    }

    public boolean insertGrupoNuevo(Grupo grupo, Boolean usuarios)
    {
        SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();

        try {
            GrupoMapper mapper = sqlSession.getMapper(GrupoMapper.class);
            mapper.insertGrupoNuevo(grupo);
            if(usuarios) mapper.insertGrupoUsuarios(grupo);
            sqlSession.commit();
            sqlSession.close();
            return true;
        } catch(Exception e){
            e.printStackTrace();
            if(sqlSession != null)
            {
                sqlSession.rollback();
                sqlSession.close();
            }
            return false;
        }
    }
```

Figura 48: DAOGrupo

- XML: Los archivos *XML* son los archivos principales de esta librería. Estos documentos permiten definir las diferentes sentencias *SQL* de forma dinámica, convirtiendo sentencias de numerosas líneas en un número muy reducido de ellas. Además, evita los problemas de inserción o selección de datos con parámetros como *SQL Injection*. Para lograr esto, cada *XML* se compone de dos partes, la primera, será el resultado de la sentencia (*resultMap*), y la segunda la sentencia.
    - *resultMap*: Un resultMap es el resultado de una sentencia SQL (en la mayoría de casos, de una *select*). Este *resultMap* estará referenciado a una clase java. De esta forma, cada parámetro que obtengamos de la sentencia irá destinado a un atributo de una clase java. Entre los atributos que necesita, están el 'id', que será el nombre por el cual se le identifique, el 'type', que es el encargado de decir a que clase java corresponde el resultMap, y *autoMapping,* que es el encargado de decir si los diferentes parámetros se van a *mappear* automáticamente (opcional). Para construir los *resultMap*, en este programa, se han utilizado las siguientes etiquetas (mostradas mediante el *resultMap* de los grupos) (Ver Figura 49):
        - *constructor*: Dentro de esta etiqueta irán los parámetros de la constructora (En el mismo orden que la constructora).
        - idArg: Esta etiqueta es la encargada de definir cada parámetro dentro de la constructora. Se utilizar el atributo '*column*' para seleccionar la columna, y el atributo '*javaType*' para decir de qué tipo java es el parámetro.
        - *association*: Esta etiqueta es la encargada de asociar a un atributo de una clase otra clase (otra clase creada por nosotros). En este caso, se trata de la clase Contenido. Se utilizan los atributos '*property*' para decir que atributo dentro de la clase es, '*javaType*' para, al igual que en la etiqueta *idArg*, definir el tipo de java, y '*column*' para seleccionar la clave primaria por la cual se va a seleccionar el contenido del interior.

- *collection*: Es similar a la etiqueta *association*, a diferencia de que, en este caso, se introducirá una lista en vez de un único elemento (la lista de contenido en este ejemplo). En este caso, el atributo '*javaType*' será la clase lista (*ArrayList*), y el atributo '*ofType*' será el tipo de la lista.

```
<resultMap type='Grupo' id='GrupoResultset' autoMapping="true">
    <constructor>
        <idArg column="G_ID_GRUPO" javaType="java.lang.Integer"/>
        <idArg column="G_NOMBRE" javaType="java.lang.String"/>
    </constructor>
    <association property="contenidoSolucion" javaType="Contenido" column="G_CONTENIDO_SOLUCION" >
        <constructor>
            <idArg column="G_CONTENIDO_SOLUCION" javaType="java.lang.Integer"/>
            <idArg column="s_NOMBRE" javaType="java.lang.String"/>
            <idArg column="S_CASTELLANO" javaType="java.lang.String"/>
            <idArg column="S_EUSKERA" javaType="java.lang.String"/>
            <idArg column="S_IMAGEN" javaType="java.lang.String"/>

        </constructor>
    </association>
    <collection property="contenidoOpciones" javaType="ArrayList" ofType="Contenido" column="G_CONTENIDO_OPCION" >
        <constructor>
            <idArg column="G_CONTENIDO_OPCION" javaType="java.lang.Integer"/>
            <idArg column="O_NOMBRE" javaType="java.lang.String"/>
            <idArg column="O_CASTELLANO" javaType="java.lang.String"/>
            <idArg column="O_EUSKERA" javaType="java.lang.String"/>
            <idArg column="O_IMAGEN" javaType="java.lang.String"/>
        </constructor>
    </collection>
</resultMap>
```

Figura 49: ResultMap

- Sentencia: La etiqueta de la sentencia dependerá de que sentencia SQL sea. Los atributos utilizados son iguales para los tres tipos de sentencias, los cuales son el 'id', que será el nombre de la sentencia que aparecerá en el mapper; 'resultType', que es el tipo de resultado que devuelve (normalmente de tipo 'int' o el id de un resultMap), y 'parameterType', que es el atributo que define el tipo de parámetros que se le van a pasar (en su mayoría, o un Map, o la propia clase Java, o un int). Además, dentro de la consulta, se pueden utilizar diferentes etiquetas para facilitar la construcción de las mismas. Todas estas etiquetas son opcionales, y a pesar de que se puedan utilizar en todas las sentencias, vamos a explicarlos de manera individual según las hemos utilizado. Por esto, a continuación, explicaremos cada sentencia SQL:
  - *Select*: Para la construcción de las sentencias *select*, se ha utilizado, principalmente, una única etiqueta (Ver Figura 50):
    - *Where*: Etiqueta encargada de definir el apartado del *where* en una sentencia. En caso de que su interior esté vacío, el where no se "escribirá" a la hora de ejecutar la sentencia.

```
<select id='esBorrableUsuario' resultType='int' parameterType='int'>
    SELECT COUNT(1)
    FROM PERMISOS_GRUPOS
    <where>
        PG_ID_GRUPO = #{idGrupo}
    </where>
</select>
```

Figura 50: Select

- *Insert*: Para la construcción de las sentencias *insert*, se han utilizado, principalmente, dos etiquetas (Ver Figura 51):
  - *selectKey*: Etiqueta encargada de realizar una subsentencia SQL para obtener una clave o un valor a introducir (normalmente es la *primary key autoincremental* de la tabla). A su vez, está compuesta por tres atributos, *keyProperty* (Sobre qué parámetro se va a escribir el resultado), *resultType* (El tipo de resultado que se va a obtener) y *order* (Cuándo se va a ejecutar la sentencia. AFTER -> Después de ejecutar la sentencia original. BEFORE -> Antes de ejecutar la sentencia original).
  - *foreach*: Etiqueta encargada de escribir en la sentencia SQL partes iguales, recorriendo un bucle (para recorrer un arrayList, por ejemplo). Para esto, contiene 6 tipos de atributos: 'collection' (el nombre del atributo que contiene la lista), 'item' (El nombre que se va a utilizar para el ítem dentro del bucle), 'index' (El número de iteraciones que se van a hacer. Con el parámetro index, se dice que se recorra toda la lista), 'open' (El carácter o caracteres que se pondrán cuando inicie el bucle), 'separator' (El carácter o caracteres que se pondrán cuando entre cada repetición o entrada en el bucle – cada fila) y 'close' (El carácter o caracteres que se pondrán cuando termine el bucle).

```
<insert id='insertGrupoNuevo' parameterType='Grupo'>
  <selectKey keyProperty="idGrupo" resultType="int" order="BEFORE">
    SELECT GREATEST(IFNULL(MAX(TSG_ID_GRUPO), 0), IFNULL(MAX(G_ID_GRUPO),0))+1 FROM TS_GRUPO, GRUPO
  </selectKey>
    INSERT INTO GRUPO (G_ID_GRUPO, G_NOMBRE, G_CONTENIDO_SOLUCION, G_CONTENIDO_OPCION)
    VALUES
    <foreach collection="contenidoOpciones" item="contenidoOp" index="index" open="(" separator="),(" close=")">
        #{idGrupo}, RTRIM(#{nombre}), #{contenidoSolucion.idContenido}, #{contenidoOp.idContenido}
    </foreach>
</insert>
```

Figura 51: Insert

- *Update*: Para la construcción de las sentencias *insert*, se han utilizado, principalmente, dos etiquetas (Ver Figura 52):
  - *set*: Etiqueta similar a la etiqueta *where*, pero en este caso del apartado 'set' de una *update*.
  - *If*: Etiqueta encargada de comprobar si un dato de entrada es lo que esperamos. Utiliza úncamente el atributo '*test*', donde se pone la condición. Un ejemplo de esta etiqueta, es para comprobar si un campo se ha introducido.

```
<update id='updateLogopeda' parameterType='Logopeda'>
    UPDATE LOGOPEDA
    <set>
        L_USUARIO= #{usuario},
        L_NOMBRE= #{nombre},
        L_APELLIDOS = #{apellidos},
        <if test="contrasena != null">
            L_CONTRASENA = #{contrasena},
        </if>
        L_IDIOMA = #{idioma}
    </set>
    <where>
        L_ID_LOGOPEDA= #{idUsuario}
    </where>

</update>
```

Figura 52: Update

- *Delete*: En el caso de la *delete*, se ha utilizado, principalmente, una etiqueta, pero no se explicará porque ya se ha hecho anteriormente (Ver Figura 53).

```
<delete id="deletePermisos" parameterType='Map'>
    DELETE FROM PERMISOS_GRUPOS
    <where>
        PG_ID_GRUPO = #{idGrupo}
        AND PG_ID_NINO= #{idNino}
    </where>
</delete>
```

Figura 53: Delete

Por último, para obtener los datos de entrada, se pone el nombre del atributo entre los caracteres '#{' y '}'.

3.2. **IdiomaProperties**

Como se ha mencionado al inicio de este apartado, todo el texto que se visualiza en el programa no está puesto directamente en el código, sino que está gestionado por esta clase.

Para lograr esta gestión, se utilizan dos archivos de extensión '*.properties*'. Estos archivos, actualmente colocados en la carpeta *Sources*¸ se llaman *español.properties* y *euskera.properties*. Este tipo de archivo se utiliza para almacenar datos mediante claves, por lo tanto, ambos archivos tendrán las mismas claves, y variará su valor (Ver Figura 54). La localización de estos dos archivos en un principio fue la misma que la clase *IdiomaProperties*, pero a la hora de probar el "*.jar"* se encontró el error de que no se encontraba dicha ruta. Por tanto, se tuvo que crear una nueva carpeta, almacenar en esta los archivos, y cambiar el método de acceso a los archivos.

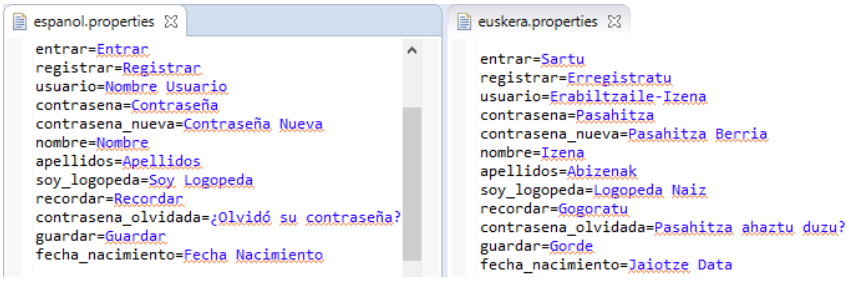


Figura 54: Properties

Además de estos dos archivos, también se ha utilizado la clase *IdiomaProperties*. Esta clase, utiliza el patrón *Singleton* para existir una única instancia de la misma. Por otro lado, tiene guardado el idioma actual del usuario en una variable, para poder acceder a él desde cualquier clase (y no tener que obtener antes el usuario). En lo referente a la obtención del texto en el idioma correspondiente, se utiliza la clase Properties para ello. Esta clase, lee un *input* (un archivo abierto para lectura), y coge su propiedad. Esta propiedad será lo que hemos llamado clave anteriormente (Ver Figura 55).

```
input = ClassLoader.getSystemResourceAsStream(idioma+".properties");
properties.load(input);

texto = properties.getProperty(propiedad);
```

Figura 55: Obtener Texto en Idioma

Además del idioma, esta clase también es la encargada de gestionar las preferencias de usuario. Estas preferencias son lo relacionado a guardar los datos del usuario (nombre de usuario y si es logopeda o tutor) para evitar introducirlo cada vez que se quiera acceder. El motivo de realizar esta acción en esta clase y no en la del gestor de login, fue porque en un principio se utilizó, al igual que con el idioma, un fichero properties para guardar estos datos (entonces se reutilizaba el código). El problema se encontró al crear el ".jar", porque los ficheros existentes dentro de un ".jar" no son modificables, por lo que se decidió utilizar la clase *Preferences* para guardar y obtener estos datos. Esta clase, una vez instanciada, se utilizará de la misma forma que un *HashMap*, es decir, se utilizará el método *put* para insertar un valor en una clave, y *get* para obtenerlo (Ver Figura 56). Esta clase no se almacena en el propio ".jar", sino que en la carpeta del usuario.

```
Preferences prefs = Preferences.userNodeForPackage(IdiomaProperties.class);
prefs.put("usuario", usuario);
prefs.put("idioma", idioma);

idioma = prefs.get("idioma", "espanol");
```

Figura 56: Preferences

3.3. **ImagePanel**

Clase encargada de mostrar imágenes en pantalla. Esta clase extiende de la clase *JPanel*, por lo que se trata de un contenedor (en este caso, de imágenes). Para realizar esta función, se le tiene que pasar a esta clase la imagen mediante un archivo (*File*) o una clase imagen (*ImageIcon*). El motivo de poder utilizar ambos es debido a que depende de donde proceda objeto (Si procede del servidor, será una imagen. Si procede del ordenador, será un archivo porque será necesaria su dirección), se utilizará uno u otro. Una vez obtenido, en caso de ser un archivo, se obtendrá a imagen que contiene. Por último, tanto la imagen como el fichero tienen sus correspondientes *getters* y *setters* para poder acceder a ellos.

3.4. **GestorServidor**

Esta clase es la encargada de subir, descargar y eliminar imágenes del servidor. En un principio, esta clase no se pensó realizar porque las imágenes se guardaron en la base de datos (como BLOB). El problema vino en que parar obtener dichas imágenes a la hora de sincronizar en el videojuego, se demoraba mucho tiempo (con solo 3 imágenes podía estar 2 minutos). Debido a esto, y tras consultarlo con los tutores, se decidió guardar las imágenes en el servidor y guardar su ruta en la base de datos (lo que redujo el tiempo a un par de segundos, dependiendo del tamaño de las imágenes).

Por otro lado, esta clase implementa el patrón *Singleton* para existir una única instancia de la misma. Además, consta de tres métodos principales. Estos métodos se explicarán a continuación, y para la realización de sus acciones, también se han creado dos archivos PHP (*Hypertext Preprocessor*), los cuales se encuentra alojados en el mismo servidor que la base de datos ("galan.ehu.eus").

UploadFile

Método encargado de subir una imagen al servidor. Para ello, se le pasan como parámetros el tipo de archivo que es (extensión), el nombre del archivo y el archivo en sí. Una vez obtenidos, se crea la URL al PHP, y se codifican los parámetros para enviarlos mediante POST al servidor (Ver Figura 57). Entre estos parámetros, se le añade la contraseña del usuario y el nombre de usuario para evitar la creación de imágenes por cualquier persona ajena. Estos se obtienen mediante el *GestorSesion*.

```
URL url = new URL("http://galan.ehu.eus/avelez012/WEB/TFG/uploadImage.php");

StringBuffer param = new StringBuffer("usuario=").append(URLEncoder.encode(usu[0], "UTF-8"));
param.append("&contrasena=").append(URLEncoder.encode(usu[1], "UTF-8"));
param.append("&tipo=").append(URLEncoder.encode(tipo, "UTF-8"));
param.append("&nombre=").append(URLEncoder.encode(nombre, "UTF-8"));
param.append("&imagen=").append(URLEncoder.encode(Base64.encode(readFile(archivo)), "UTF-8"));
```

Figura 57: Parámetros Upload

En el caso de la imagen, antes de enviarla, se convierte a un array de bytes y se codifica en Base64 (esto último se hace para que sea más fácil y seguro su transporte a través de la red). Esto se realiza mediante la función readFile, la cual leerá la imagen como *FileInputStream* , obtendrá sus bytes de 1024 en 1024, y los introducirá en un *array* de *bytes* de salida (*ByteArrayOutputStream*) (Ver Figura 58). Por último, se obtendrá el *array* de *bytes* de este último *array*.

```
FileInputStream fis = new FileInputStream(file);
byte[] buffer = new byte[1024];
bos = new ByteArrayOutputStream();
for (int len; (len = fis.read(buffer)) != -1;) {
    bos.write(buffer, 0, len);
}
fis.close();
```

Figura 58: Read File

Una vez codificados todos los parámetros, se realiza la conexión. Para la conexión se le ha puesto un tiempo límite de respuesta o *timeout* de 5000 segundos. Además, el lenguaje utilizado será el del ordenador (obtenido mediante *Locale*). Cuando se reciba una respuesta del servidor, se comprobará que dicha respuesta tenga el código 200 (Código de estado de HTML que significa OK). Si es así, se leerá el mensaje de respuesta y se obtendrá la dirección de la imagen que se está devolviendo.

DownloadImage:

La función de este método consiste en obtener de una dirección web (pasada como parámetro) una imagen. Para ello se utiliza un buffer (espacio en memoria temporal para el almacenamiento) de imágenes (*BufferedImage*) (Ver Figura 59).

```
url = new URL(direccion);
BufferedImage img = ImageIO.read(url);
imagen = new ImageIcon(img);
```

Figura 59: DownloadImage

DeleteFile:

En el caso de la función *delete*, funcionará de forma similar a la función *uploadFile*. En este caso, su objetivo es el de eliminar una imagen en vez de crearla, por lo que como parámetros irán el usuario, la contraseña y la dirección URL donde está alojada la imagen que queremos eliminar (Ver Figura 60).

```
URL url = new URL("http://galan.ehu.eus/avelez012/WEB/TFG/deleteImage.php");

StringBuffer param = new StringBuffer("usuario=").append(URLEncoder.encode(usu[0], "UTF-8"));
param.append("&contrasena=").append(URLEncoder.encode(usu[1], "UTF-8"));
param.append("&url=").append(URLEncoder.encode(urlImagen, "UTF-8"));
```

Figura 60: Parámetros Delete

Por otro lado, en este caso, no se requiere la obtención de datos. Aun así, se verificará que todo ha ido correcto y que se devuelve tanto el código 200 como un JSON (*JavaScript Object Notation*). Esto último se comprobará observando si empieza la respuesta por el carácter '{'.

UploadImage.php

Este archivo PHP será el encargado de guardar la imagen pasada como parámetro en el servidor. Para ello, primero se comprobará que se ha pasado todos los parámetros requeridos. Una vez verificado, obtendrá dichos parámetros, y en caso de la imagen, la decodificará en Base64 mediante la función *base64.decode*.

Una vez obtenido todos los datos, se conectará a la base de datos y se comprobará que las credenciales son correctas (usuario y contraseña). A continuación, se obtendrá la fecha actual en milisegundos, se añadirá al nombre de la imagen, y se guardará la imagen en la carpeta "Img", la cual está disponible en la misma ruta que el PHP (Ver Figura 61). Por último, se devolverá la dirección de la imagen. Si en cualquier momento existe un error, o no entra en una condición, se devolverá un error en formato JSON.

```
$fecha = new DateTime();
$ruta = "Img/".$nombre.'_'.$fecha->getTimestamp().'.'.$tipo;
$fp = fopen($ruta, 'w');
$ruta = "http://galan.ehu.eus/avelez012/WEB/TFG/".$ruta;
fwrite($fp, $imagen);
if(fclose($fp)){
    print_r($ruta); // Image uploaded
```

Figura 61: UploadImage.php

DeleteImage.php

En el caso del archivo PHP de eliminación de imágenes, se hará un proceso similar al de subir la imagen. La diferencia mayor será que en vez de añadir una imagen, se eliminará el fichero que exista en la dirección pasada como parámetro (Ver Figura 62).

```
if (file_exists($url)) {
   unlink($url);
```

Figura 62: DeleteImage.php

3.5. **Login**

Esta ventana es la ventana principal, y la encargada de comprobar las acreditaciones para poder entrar en el sistema. Está compuesta por dos campos de texto (*JTextField*), dos botones seleccionables (*JCheckBox*) un texto (*JLabel*) clickable y dos botones (*JButton*).

Como se ha mencionado al comienzo de esta sección, todas las ventanas se rigen bajo el patrón de *observador-observable*. Por esto, todas las acciones sobre esta ventana las controlará su gestor (*GestorSesion*). En este caso, el gestor es el encargado de guardar la sesión. De esta forma, se controlará en todo momento quien es el usuario conectado, y si es logopeda o un tutor.

Por otro lado, todos los gestores están creados mediante el patrón *Singleton*. De esta forma, únicamente podrá existir una única instancia del mismo. Asimismo, en la constructora privada de todos los gestores se inicializarán todos los *DAO* que dicha clase utilizará, los cuáles, en este caso son *LogopedaDAO* y *NinoDAO*.

En lo referente a la utilización de la ventana, el usuario podrá seleccionar si es logopeda o no, y si quiere recordar sus datos para la próxima vez que se conecte (nombre de usuario y si es logopeda). Una vez pulsado el botón de *Login*, se comprobarán los datos y si son correctos, pasará a la pantalla del *Menú*. En caso de no serlo, aparecerá un mensaje de error. Para comprobar estos datos, se introducirán en el gestor y se ejecutará mediante Hilos (*Thread*). Para conseguir esto, los gestores implementan la clase *Runnable*, la cual, tiene un método llamado *run* que se le llama una vez iniciado el hilo. Debido a que se ejecuta mediante hilos, y no directamente llamando a una función, es necesaria la carga de datos de la vista al gestor. En esta carga de datos, se le mandará los datos introducidos o marcados por el usuario, y el dato de una variable constante para saber el gestor que acción realizar (Ver Figura 63).

```
gestorSesion.setDatos(usuario, contrasena, checkLogopeda.isSelected(), checkRecordar.isSelected());
dialog = new JProgressDialog(VentanaLogin.this, idioma.leerProperty("login.titulo_conectandose"),
        idioma.leerProperty("login.mensaje_conectandose"));
setEnabled(false);
new Thread(gestorSesion).start();
```

Figura 63: Thread

Por otro lado, mientras el hilo se está ejecutando, la pantalla se quedará bloqueada y aparecerá una ventana de carga con un mensaje de lo que está ocurriendo (Ver Figura 64).

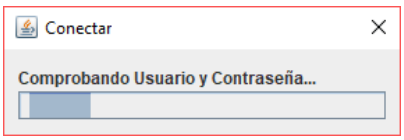


Figura 64: Dialog

Una vez hecho, y ejecutado el hilo, se llamará al método *run*, y dependiendo de la acción que se le haya dicho que hacer, hará llamará a un método u otro (Ver Figura 65).

```
@Override
public void run() {
    if (accionActual == accionLogin) {
        identificarse(String.valueOf(datos[0]), String.valueOf(datos[1]), (boolean) datos[2], (boolean) datos[3]);
    } else if (accionActual == accionRegistro) {
        registrarse(String.valueOf(datos[0]), String.valueOf(datos[1]), String.valueOf(datos[2]),
                String.valueOf(datos[3]), String.valueOf(datos[4]));
    } else if (accionActual == accionRecuperarContrasena) {
        recuperarCotrasena(String.valueOf(datos[0]), (boolean) datos[1]);
    } else if (accionActual == accionModificacion) {
        actualizarDatos(String.valueOf(datos[0]), String.valueOf(datos[1]), String.valueOf(datos[2]),
                datos[3] == null ? null : String.valueOf(datos[3]),
                datos[4] == null ? null : String.valueOf(datos[4]),
                datos[5] == null ? null : String.valueOf(datos[5]),
                datos[6] == null ? null : Integer.valueOf(String.valueOf(datos[6])),
                datos[7] == null || datos[7].equals("") ? null : Integer.valueOf(String.valueOf(datos[7])));
    }
}
```

Figura 65: Run

En este caso, debido a que el usuario puede ser logopeda o tutor (*nino*), la clase mediante la cual se va a guardar el usuario en el gestor va a ser la clase *Usuario*. De esta forma, de esta clase van a heredar tanto *Logopeda* como *Nino*, y dependiendo de cuál sea, se utilizará *LogopedaDAO* o *NinoDAO*. Para comprobar si el usuario existe, se comprobará si hay constancia en la base de datos de un registro con el usuario introducido, y la contraseña introducida pero cifrada mediante *SHA1* (esto se hace como medida de seguridad en caso de acceso a los datos de la base de datos). Por último, en caso de que se nos devuelva un usuario (logopeda o niño), se informará a la clase *IdiomaProperties* del idioma de dicho usuario. Si se ha marcado la casilla recordar, se guardará sus datos mediante la misma clase que el idioma (y mediante los métodos explicados anteriormente). Además, se le notificará a la vista que se ha terminado (junto con una acción estática). En caso contrario, o en caso de error (por imposibilidad de conexión, por ejemplo), únicamente se notificará a la ventana que se ha terminado con su correspondiente acción estática con los métodos estáticos de la clase Observable *notifyObservers* y *setChanged* (Ver Figura 66).

```
            setChanged();
            notifyObservers(accionLoginCorrecto);
        } else {
            setChanged();
            notifyObservers(accionLoginIncorrecto);
        }
    } catch (Exception e) {
        setChanged();
        notifyObservers(accionCargarError);
```

Figura 66: Notificación de cambios

Una vez que la ventana obtenga la notificación (mediante el método update implementado al implementar la clase *Observer*), según la acción que el gestor haya notificado, se mostrará un mensaje, o se cambiará a la ventana del menú. En la Figura 67 podemos observar las acciones tomadas a cabo para la carga inicial, Login incorrecto y error.

```
if ((int) arg == GestorSesion.obtSesion().accionLoginCorrecto) {
    gestorSesion.deleteObserver(VentanaLogin.this);
    new VentanaMenu();
    dispose();
} else if ((int) arg == gestorSesion.accionLoginIncorrecto) {
    JOptionPane.showMessageDialog(contentPane, idioma.leerProperty("error.datos_incorrectos"));
} else if ((int) arg == gestorSesion.accionCargarError) {
    JOptionPane.showMessageDialog(contentPane, idioma.leerProperty("error_carga"), idioma.leerProperty("error"),
            JOptionPane.ERROR_MESSAGE);
```

Figura 67: Update

Por último, están el botón de registrar (el cual dará la posibilidad de cambiar a la ventana de registro), y el *JLabel* de recordar contraseña. Este *JLabel* tiene implementado un *OnClickListener* para realizar alguna acción al hacer *click* sobre él. En dicho caso, se realizará algo similar que al hacer login, pero con otra acción. Para cambiar la contraseña, se introducirá automáticamente el nombre del usuario cifrado mediante SHA1.

3.6. **Registro**

La segunda ventana a realizar fue la del Registro. Esta ventana, formada por 4 campos de texto, un selector (*JComboBox*) y dos botones (Ver Figura 68).

Figura 68: Ventana Registro

En este caso, esta ventana es similar a la de *Login*. Además, hacen uso de la misma clase gestora (*GestorSesion*). Lo único que diferencia a la anterior ventana es el *JComboBox*. Debido a que en base de datos no es muy recomendable la inserción de caracteres no ASCII, a pesar de que se puedan seleccionar los textos "Español" o "Euskera" (o sus derivados en euskera), realmente, lo que se guarda en la base de datos será "español" o "euskera" y da igual el idioma en el que el usuario esté visualizando la pantalla. Además, una vez pulsado el botón de registro, si el nombre de usuario no existe, se cambiará a la pantalla de *Login* (en caso de existir, saldrá un error). Por último, en cualquier ventana que aparezca el botón con el símbolo '<' volverá a la pantalla anterior (esto no se volverá a mencionar, para no sonar repetitivo). En este caso, a la de *Login*.

3.7. **Menú**

Esta ventana será la encargada de dar al usuario las diferentes opciones que tiene. Por esto, está únicamente formada por botones. Además, como esta aplicación se puede utilizar tanto por la logopeda, como por los tutores, tiene dos tipos de interfaces (Ver Figura 69 y Figura 70).

Figura 69: Menú Logopeda

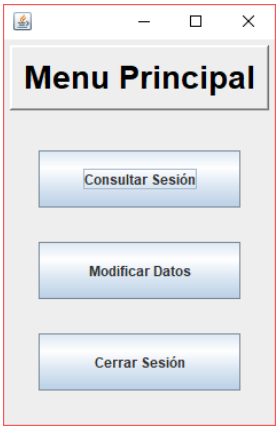


Figura 70: Menú tutor

Debido a lo mencionado en el párrafo anterior, lo primero que se comprobará en esta pantalla será si el usuario es logopeda o tutor. Solo se mostrará el botón de 'Administrar Juegos' si es logopeda. Además, en caso de que no exista usuario conectado (por cualquier posible fallo), se le dirigirá a la ventana de *Login*. (Esta comprobación la hará en todas las ventanas a partir de esta) Como esta ventana no necesita de acciones contra la base de datos, no necesita de una clase gestora.

Por último, a la hora de pulsar cada botón, nos dirigirá a su pantalla:

- Consultar Sesión: Nos dirigirá a la ventana de Historial Sesión.
- Administrar Juegos: Nos dirigirá a la ventana de Administrar Juegos.
- Modificar Datos: Nos dirigirá a la ventana de Modificar Datos.
- Cerrar Sesión: Nos dirigirá a la ventana de Login, y se cerrará la sesión.

3.8. **Historial Sesión**

Esta ventana será la encargada de darle a la logopeda la opción de crear nuevos niños o pacientes, cambiar la información de los existentes, crear sesiones y revisar el historial de los minijuegos jugados. A su vez, el tutor podrá visualizar las sesiones que la logopeda haya marcado como visibles para los tutores. Para lograr esto, la ventana tiene dos selectores (*JComboBox*) en la parte superior (Ver Figura 71). Uno será utilizado para seleccionar el paciente del que queremos ver los datos o para crear uno nuevo (en este caso, nos posicionaremos en el primer ítem, el cual es un ítem vacío). Para obtener el nombre de los niños, se ha utilizado el gestor *GestorNinos*. El otro *JComboBox*, en cambio, será utilizado para elegir la ventana que queremos ver. En este segundo caso, las diferentes opciones han sido introducidas directamente. Además, en caso de que no esté seleccionado ningún niño (o en caso de ser el tutor), el segundo *checkbox* quedará inhabilitado.

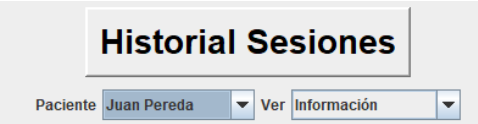


Figura 71: JComboBox *Historial Sesión*

Una vez explicado cómo funciona los dos *JComboBox*, el siguiente paso es explicar cómo se ha realizado cada apartado. Además, debido a que vamos a mostrar una gran cantidad de datos, la ventana se visualizará en pantalla completa inicialmente.

Información

Este apartado contendrá toda la información referente al niño, y solo podrá ser visualizada por la logopeda. Si no seleccionamos ningún niño, todos los campos de este apartado estarán vacíos. Además, aquellos campos con un asterisco serán obligatorios para rellenar. Por otro lado, esta ventana, al igual que los datos de los niños en la base de datos, están separados en partes, las cuales están todas metidas en un *ScrollView* para facilitar la navegación. Aquellos datos que su cuadro de texto ocupe más de una línea, tendrán una capacidad de 500 letras (serán los campos de información para la logopeda – por ejemplo, las enfermedades).

La primera parte será la información del paciente (Ver Figura 72). Esta parte está formada por los únicos campos que son obligatorios, entre los que se encuentran el nombre y apellidos, un número de teléfono y la fecha de nacimiento. El número de caso, en base de datos, se refiere a identificador. Este es opcional (puede que la logopeda tenga guardados números de caso y los quiera poner). Si no se proporciona, se pondrá el primer número no ocupado.

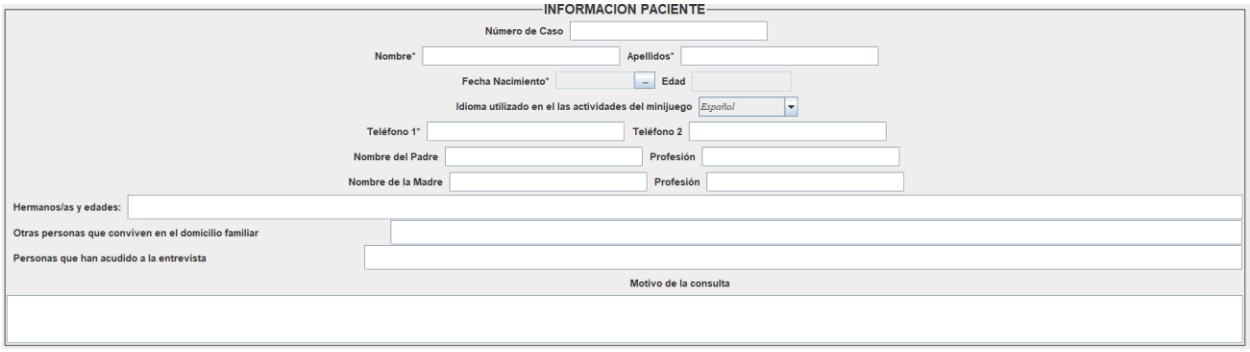


Figura 72: Información Paciente

Como se puede observar, el campo de la fecha de nacimiento está bloqueado. Para seleccionar una fecha, habrá que pulsar sobre el botón con los tres puntos. Este botón desplegará un calendario (Ver Figura 73), donde podremos seleccionar la fecha. Esto se ha logrado mediante las clases de java *JDatePickerImpl* y *JDatePanelImpl*. Además, se utilizan las *Properties* de java y la clase *UtilDateModel* para dar formato del calendario que se abre (el idioma de la fecha actual de la parte inferior). Por último, una vez seleccionada la fecha, no solo se introducirá en el campo correspondiente, sino que también se calculará la edad y se pondrá en su campo. Para ello, se ha utilizado la clase Calendar para obtener la fecha actual y compararla con la elegida.

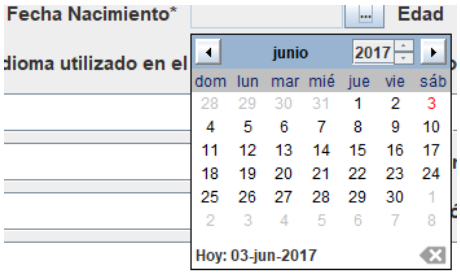


Figura 73: Calendario

Los siguientes apartados, son los referentes a la información proveniente del fichero de anamnesis. Los campos de estas partes serán todos de una longitud máxima de 500 letras. Además, no será obligatoria la introducción de los mismos. En la Figura 74 y

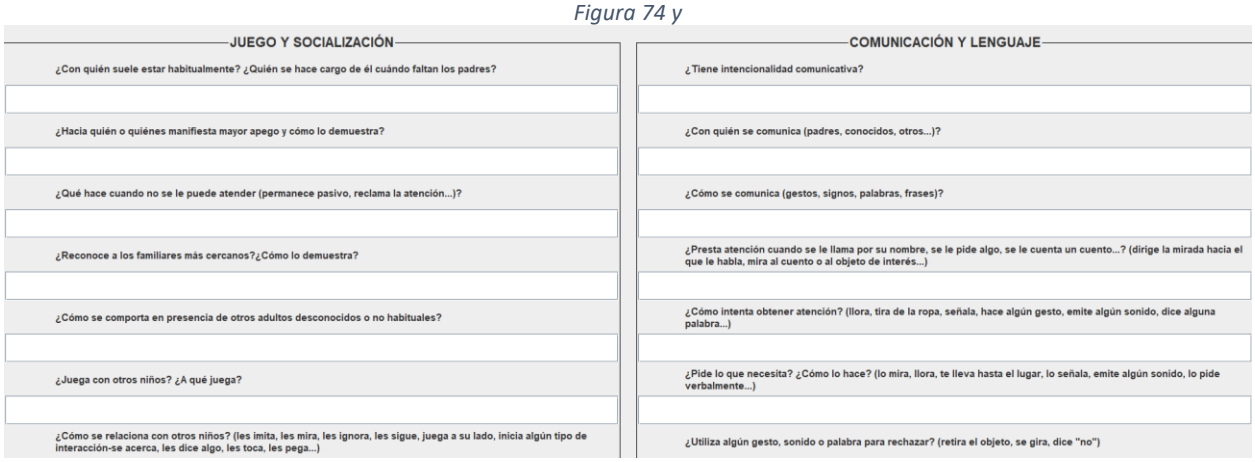


Figura 75 podemos observar el encabezado y algunos campos de estas dos partes. Un dato a mencionar es que los textos están formados mediante las etiquetas de html (están escritos entre ‘<html>’ y ‘</html>’). Esto se ha hecho para, en caso de que una pantalla sea más pequeña que el título de un campo, este salga escrito en más de una línea.

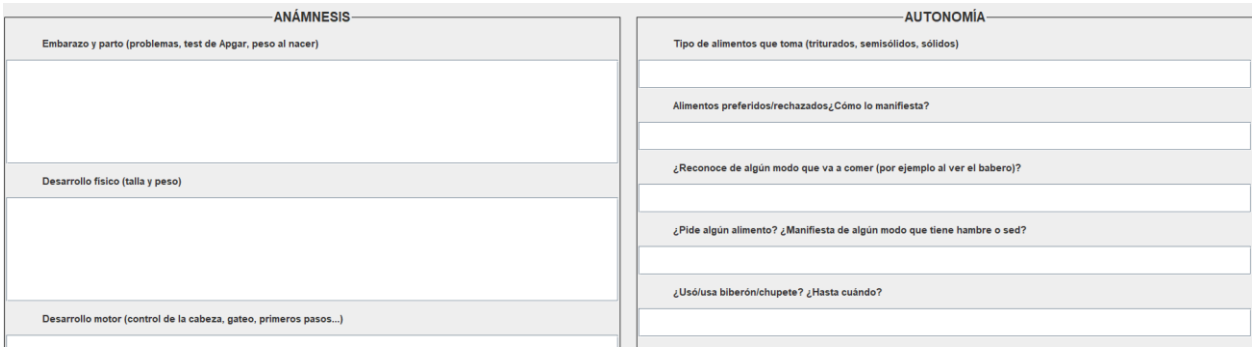


Figura 74: Anámnesis y Autonomía

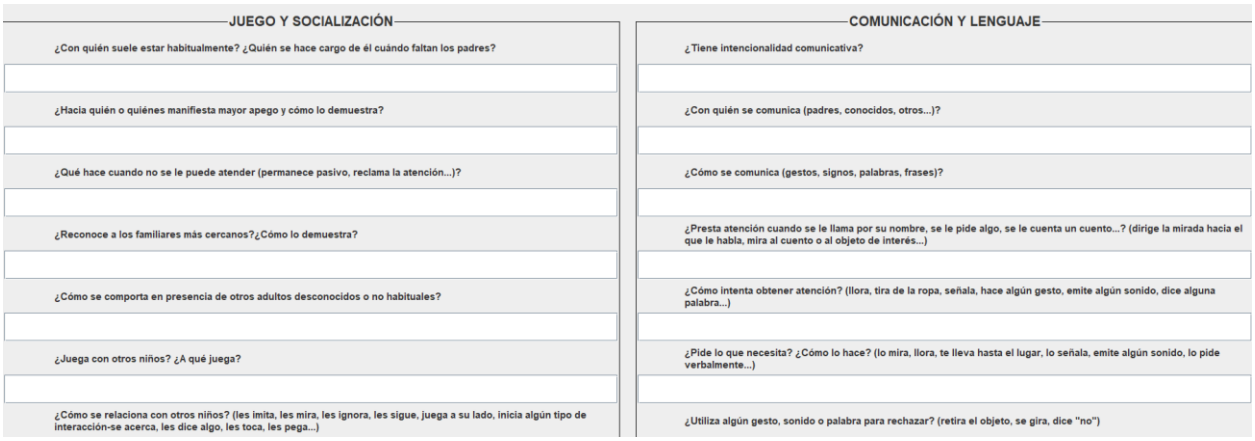


Figura 75: Juego y Socialización y Comunicación y Lenguaje

Una vez rellenados todos los campos obligatorios (y de los opcionales, los que se quieran) y pulsar en el botón de “Añadir Paciente”, en caso de no haber rellenado el campo del número del caso, se nos recordará (Ver Figura 76). En caso de dar a *Aceptar*, se almacenarán los datos del paciente.

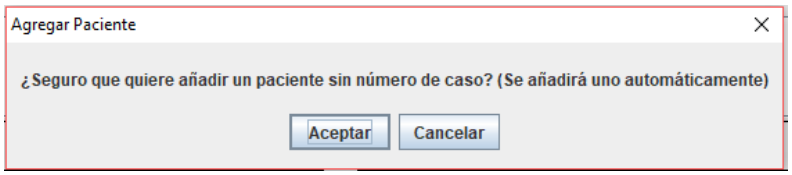


Figura 76: Sin Número de Caso

Para insertar al paciente, se obtendrán todos los datos en una lista, y se mandarán al *GestorNino*. A la hora de obtener los campos, el idioma se obtendrá como ‘espanol’ si es el primer ítem el que está seleccionado, y ‘euskera’ si es el segundo. Para iniciar con la inserción, se ejecutará el *GestorNino* en un nuevo hilo (todos los gestores extienden de la clase *Runnable*, por lo que tienen el método *Run*). Al crear la clase niño para insertar su información en base de datos, primero se llama a su constructora con los atributos principales, para más tarde añadir el resto de atributos uno a uno mediante los *setters* (60 exactamente). Una cosa a mencionar es que entre los campos a rellenar no existe ningún apartado para introducir ni la contraseña ni el nombre de usuario. Esto es porque al almacenar los datos del paciente por primera vez, como usuario y contraseña se pondrá su nombre + espacio + su apellido. En el caso de la contraseña, esta será cifrada mediante *SHA1* (Ver Figura 77).

```
Calendar cal = Calendar.getInstance();
cal.set(Integer.valueOf(d[0]), Integer.valueOf(d[1])-1, Integer.valueOf(d[2]));
Nino nino = new Nino(datosNino[0].length()>0 ? new Integer(datosNino[0]) : null, datosNino[1] + " " + datosNino[2], datosNino[1], datosNino[2],
        SHA1.getStringMensageDigest(datosNino[1] + " " + datosNino[2]), "espanol", datosNino[4], cal.getTime(), new Integer(datosNino[5]),
        datosNino[6] != null && datosNino[6].length() > 0 ? new Integer(datosNino[6]) : null);
nino.setNombrePadre(datos[7]);
nino.setProfesionPadre(datos[8]);
nino.setNombreMadre(datos[9]);
nino.setProfesionMadre(datos[10]);
nino.setHermanosEdades(datos[11]);
nino.setOtrasConvivencias(datos[12]);
nino.setEntrevistados(datos[13]);
nino.setMotivoConsulta(datos[14]);
nino.setEmbarazoParto(datos[15]);
```

Figura 77: Crear Paciente

Como los datos están distribuidos en varias tablas, la inserción de dichos datos se realizará tabla a tabla (Ver Figura 78).

```
NinoMapper mapper = sqlSession.getMapper(NinoMapper.class);
mapper.insertNino(nino);
mapper.insertNinoAnamnesis(nino);
mapper.insertNinoAutonomia(nino);
mapper.insertNinoSocializacion(nino);
mapper.insertNinoLenguaje(nino);
```

Figura 78: Inserción Paciente

Una vez terminada la inserción, se obtendrán todos los pacientes de nuevo (por si desde otro ordenador se ha creado algún otro). Una vez hecho esto, se notificará a la ventana.

Con los pacientes ya en nuestra base de datos, podemos ver y cambiar su información. Para ello, deberemos seleccionar el paciente que queramos en el primer *JComboBox* y seleccionar la opción 'Información' del segundo. La única diferencia que existirá entre las ventanas que se ven cuando se va a crear un nuevo paciente y cuando se va a modificar uno existente es los campos bloqueados. Estos campos serán el número de caso, el nombre y el apellido. Por otro lado, todos los datos del paciente serán introducidos en sus respectivos campos. Para ello, se creará una lista con todos los datos del usuario, y más tarde, la ventana posicionará dichos datos.

Sesión

Esta ventana contendrá las sesiones que introduzca la logopeda. A su vez, es la única sección de esta ventana que puede ver el tutor. En ella, aparecerá una lista con todas las sesiones guardadas (fecha de la sesión y comentario) que quiere la logopeda que puedan ser vistas por el tutor (en caso de estar con una cuenta de tutor) (Ver Figura 79).

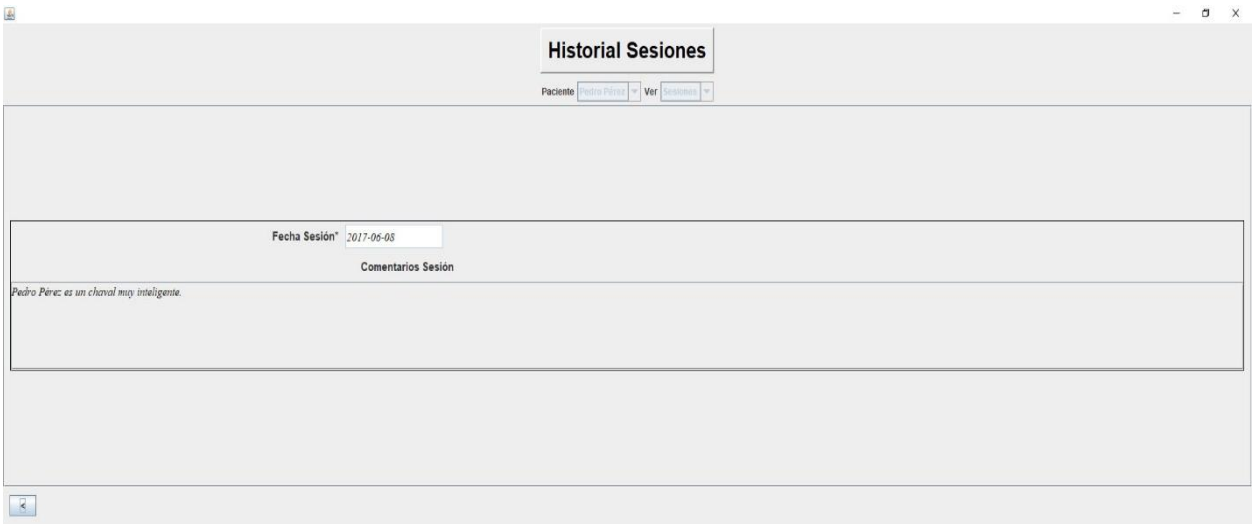


Figura 79: Sesión Tutor

En caso de estar en la cuenta de la logopeda, tendrá las mismas características, pero en este caso, se podrá modificar las sesiones creadas anteriormente y se podrá crear nuevas (Ver Figura 80). Además, aparecerá un *checkbox* en cada sesión para decidir si dicha sesión la pueden ver los tutores o no. Para añadir una sesión nueva, habrá que rellenar la primera sesión que se tiene (que en su botón aparecerá 'Añadir Sesión'). Para modificar cualquier sesión, basta con cambiar lo que queramos y pulsar el botón 'Modificar Sesión' que está debajo del comentario. Parar la fecha, se utiliza el mismo sistema que con la fecha de nacimiento.

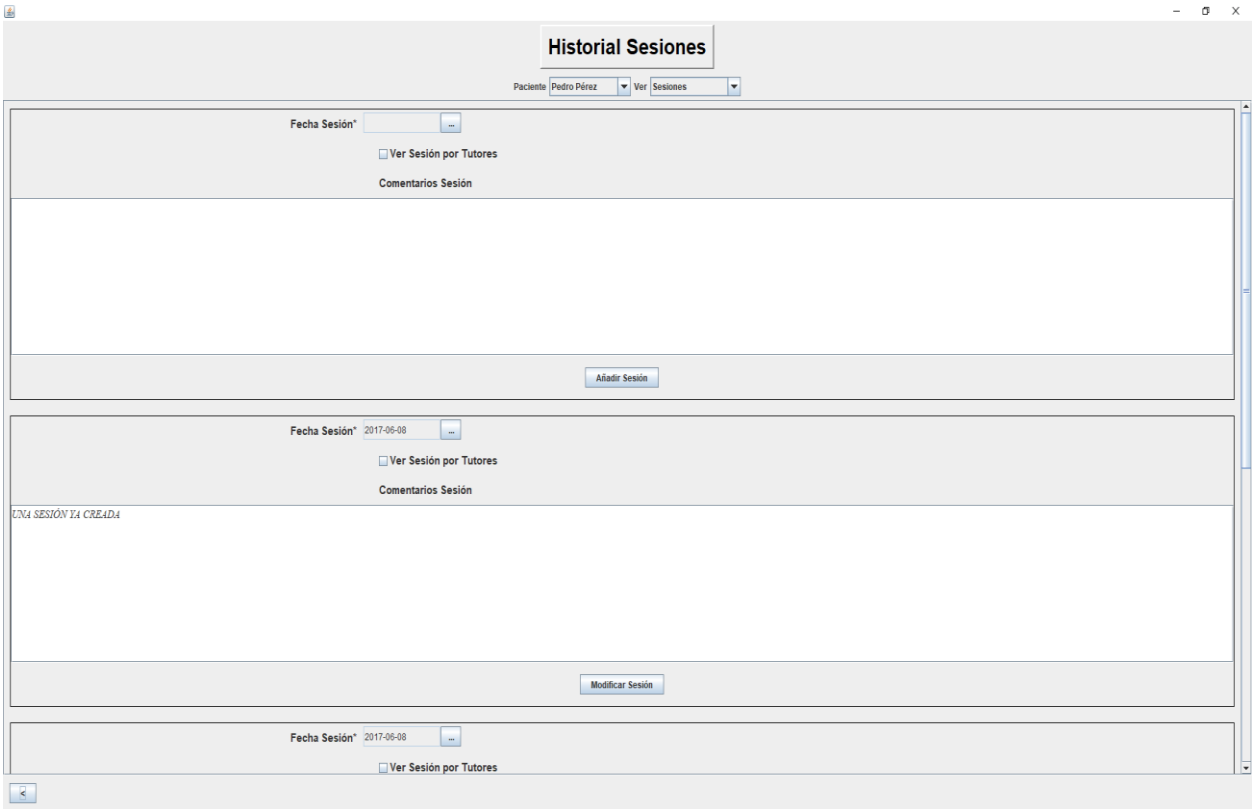


Figura 80: Sesiones Logopeda

Historial Minijuegos

El último apartado se trata del historial de minijuegos. En este apartado, se podrán visualizar los diferentes resultados que el paciente ha obtenido en los minijuegos que ha jugado. La información disponible será el nombre del minijuego, la fecha y hora en la que se ha jugado, si se ha completado el juego, las vidas restantes, los errores cometidos, y los segundos tardados. Todo esto aparecerá en una *ScrollView* (Ver Figura 81).

Figura 81: Historial Minijuegos

3.9. **Modificar Datos**

Está ventana es similar en apariencia a la ventana de registro. Además, al igual que la ventana de menú, parecerán diferentes elementos si se trata de la logopeda o de tutor (Ver Figura 82 y Figura 83)

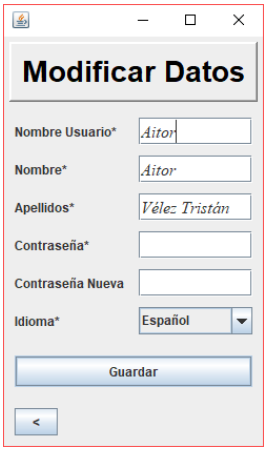


Figura 82: Modificar Datos Logopeda

Figura 83: Modificar Datos Tutor

Nada más entrar en la ventana, nuestros datos actuales serán introducidos en sus respectivos campos automáticamente. Además, para modificar cualquier dato será obligatorio introducir la contraseña actual. Además, si el campo contraseña nueva se deja vacío, la contraseña no se cambiará. Otro dato a mencionar es que la fecha de nacimiento la podremos seleccionar de la misma forma que en el Historial Sesión. Por último, una vez pulsado el botón guardar, y la contraseña sea correcta, se guardarán los datos y nos redirigirá al menú. En caso de cambiar el idioma, este se cambiará en la clase *IdiomaProperties* también, por lo que se nos cambiará el idioma global del programa.

En cuanto al gestor, utiliza el mismo que utiliza la pantalla de *Login* y Registro (*GestorSesion*)

3.10.**Administrar Juegos**

Este segundo menú, solo visible por la logopeda, es el encargado de dar a la logopeda las diferentes opciones de personalizar el videojuego. Está formado por 8 botones, los cuáles se pueden ver en la Figura 84.

Figura 84: Administrar Juegos

Respecto al gestor, en esta ventana ocurre lo mismo que en la ventana Menú. Además, si el usuario conectado no es logopeda, nos dirigirá al *Login*. Los botones nos dirigirán a las siguientes ventanas:

- Fondos: Nos dirigirá a la ventana de Fondos.
- Mapas: Nos dirigirá a la ventana de Mapas.
- Premios: Nos dirigirá a la ventana de Premios.
- Contenidos: Nos dirigirá a la ventana de Contenidos
- Grupos: Nos dirigirá a la ventana de Grupos.
- Minijuegos: Nos dirigirá a la ventana de Minijuegos.
- Permisos: Nos dirigirá a la ventana de Permisos.
- >: Botón de atrás mencionado anteriormente. Nos llevará a la ventana del Menú.

3.11.**Fondos**

Esta ventana será la que nos permitirá agregar nuevos fondos para los minijuegos. Para ello, dispone de un cuadro selector en su parte superior (*JComboBox*), un panel de imagen en el centro (*ImagePanel*), y un cuadro de texto y botón en su parte inferior (Ver Figura 85).

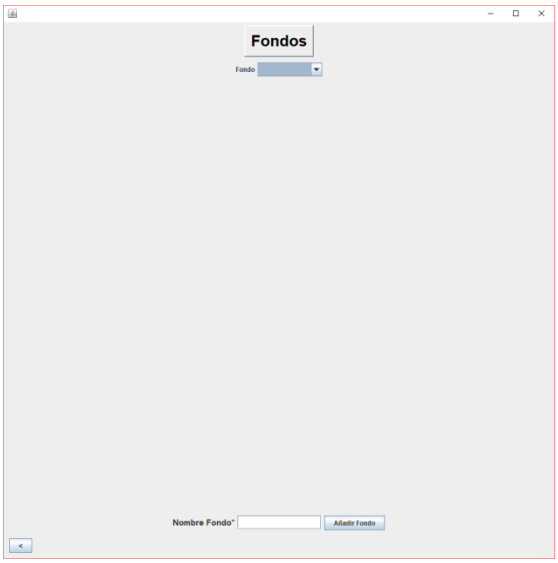


Figura 85: VentanaFondo

En el cuadro selector de la parte superior aparecerá el nombre de todos los fondos existentes. Dicho nombre, se le asigna al crearlo en la parte inferior (Por eso es necesario un nombre único para cada fondo). Por otro lado, el campo del nombre y su botón de crear solo estarán habilitados en caso de que esté seleccionada la primera opción del selector, es decir, la opción en blanco (ningún nombre). Una vez introducido un nombre y pulsar el botón de "*Añadir Fondo*", se comprobará que dicho fondo no existe (si existe aparecerá un error). En caso de no existir, se abrirá una nueva ventana donde podremos elegir la imagen del fondo. Además al seleccionar cualquier archivo que no sea carpeta, nos aparecerá a la derecha una imagen de avance del archivo seleccionado (Esto se hace gracias a la clase *ImagePreview*) (Ver Figura 86).

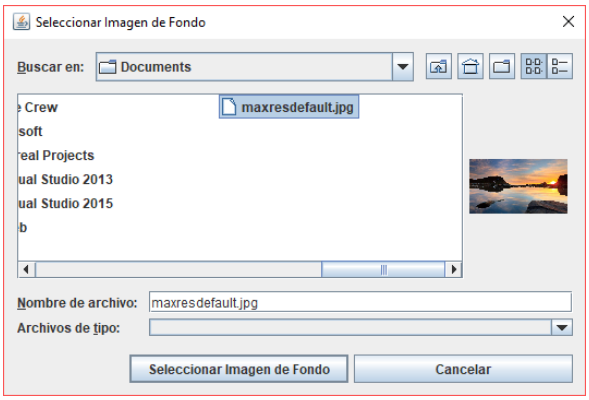


Figura 86: FileChooser Preview

Todo este proceso de selección de imagen se ha realizado mediante un selector de archivos (*JFileChooser*). Además, se ha restringido los archivos que podremos utilizar a aquellos que finalicen en "*.jpg*" (con la función *endsWith*) y que su dimensión sea mayor o igual a 640x360 (convirtiendo el archivo a imagen y obteniendo sus dimensiones). Esto último del tamaño se hace para que no se vea *pixelado* en el videojuego.

Una vez pulsado el botón de seleccionar imagen, esta se insertará gracias al gestor *GestorFondo*. Para ello, se le pasará la imagen y el nombre como parámetros y se iniciará en un hilo al igual que en las demás ventanas. A la hora de crear el fondo, la única novedad es la subida de la imagen al servidor. En la constructora del fondo (la que no necesita el id), se introduce el nombre del fondo, y en el caso del atributo *url*, utiliza el método *uploadFile* de la clase *GestorServidor* para subir la imagen.

Una vez insertado el fondo (mediante *FondoDAO*), se obtendrán de nuevo todos los fondos por si se han insertado algún otro (desde otro ordenador). Una vez terminado todo, si ha ido todo correcto, se le notificará a la ventana.

Por otro lado, al seleccionar cualquier ítem en el selector, se cargará la imagen asociada a dicho fondo en la pantalla (Ver Figura 87). Para esto, se utiliza la clase *ImagePanel*, descrita anteriormente.

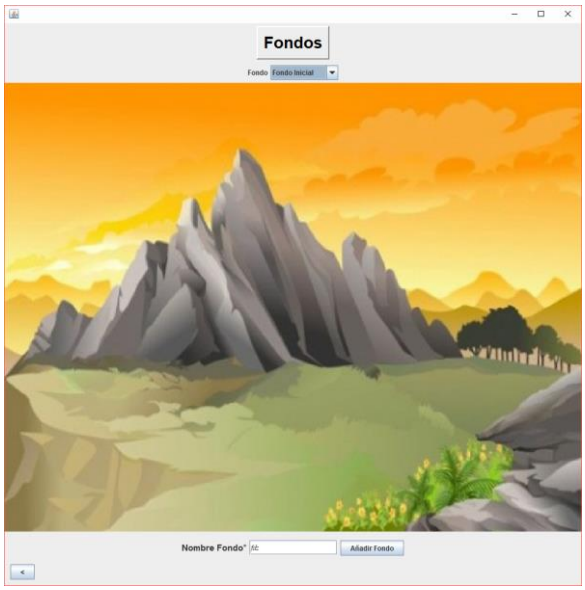


Figura 87: Ventana Fondo Imagen

Por último, al acceder a esta pantalla se cargarán todos los datos (mientras se muestra una barra de carga al igual que en el Historial_Sesion). En caso de que no exista ningún fondo en la base de datos, se introducirá uno automáticamente. Esto se hace para no obligar a la logopeda a buscar un fondo. Esta imagen estará posicionada en la carpeta *Source* del proyecto, y será accedida mediante la clase *ClassLoader* de java. Una vez cargados todos los datos, se le notificará a la ventana.

3.12.**Mapas**

En lo referente a la visión inicial de la ventana, la ventana del mapa es similar a la del fondo. Debido a esto, no repetiremos la explicación de los aspectos similares (Ver Figura 88).

Figura 88: Ventana Mapa

Las diferencias entre ambas pantallas residen una vez pulsemos el botón de “Añadir Mapa”. En este caso, en vez de añadir el mapa, nos lo mostrará en pantalla, junto con un botón nuevo en la parte inferior de la pantalla. Este botón sirve para crear nuevos botones (Ver Figura 89). Por otro lado, tanto en este momento, como cuando se seleccione para ver algún mapa, el tamaño de la ventana se bloqueará para no poder ser expandible. Esto se ha hecho para evitar que los botones se descuadren.

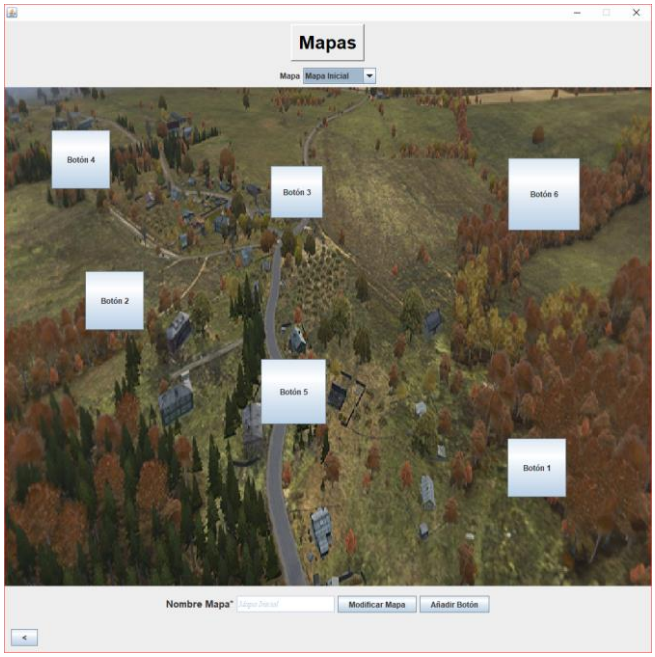


Figura 89: Ventana Mapa Imagen

Una vez pulsado el botón “Añadir Botón”, se creará un nuevo componente en la ventana (de tipo *JButton*) y se agregará a una lista de botones. Además, se le posicionará en la esquina superior izquierda del mapa, y con su tamaño mínimo (50x50).

Estos botones, tendrá además tres acciones posibles. La primera acción es la de moverlos. Para ello se les ha añadido un escuchador del movimiento del ratón (*MouseMotionListener*). Este *Listener* controlará que el botón no se salga de las coordenadas que ocupa la imagen del mapa. Esto se comprueba obteniendo la posición y tamaño del panel donde está el mapa, y cambiando la posición solo cuando no se encuentre fuera de dicho panel.

Por otro lado, tiene la opción de aumentarle el tamaño y eliminarlo. Para realizar ambas acciones, se les ha añadido un escuchador de *click* del ratón (*MouseListener*). De esta forma, en caso de pulsar el botón 3 (botón derecho) del ratón, nos aparecerá una ventana para confirmar la eliminación de dicho botón (Ver Figura 90).

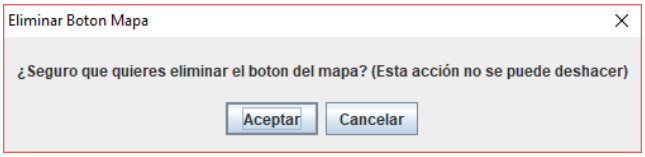


Figura 90: Seguro Eliminar Botón

Por último, nos encontramos con la tercera funcionalidad (cambiar el tamaño). Para esto, deberemos pulsar el botón 1 del ratón (botón izquierdo) y la tecla control al mismo tiempo (y sobre el botón). Esto aumentará el tamaño del botón en 10, hasta que su medida sea 150 (que se volverá a 50).

Una vez pulsado el botón “Guardar”, se comprobará que no está ningún botón encima de otro. Esto se comprobará observando la posición y las dimensiones de cada botón (Ver Figura 91).

```
if(btn1.getWidth() > btn2.getWidth()){
    bien = !(btn1.getBounds().contains(btn2.getX(), btn2.getY()) ||
             btn1.getBounds().contains(btn2.getX(), btn2.getY()+btn2.getHeight()) ||
             btn1.getBounds().contains(btn2.getX()+btn2.getWidth(), btn2.getY()) ||
             btn1.getBounds().contains(btn2.getX()+btn2.getWidth(), btn2.getY()+btn2.getHeight()));

}else{
    bien = !(btn2.getBounds().contains(btn1.getX(), btn1.getY()) ||
            btn2.getBounds().contains(btn1.getX(), btn1.getY()+btn1.getHeight()) ||
            btn2.getBounds().contains(btn1.getX()+btn1.getWidth(), btn1.getY()) ||
            btn2.getBounds().contains(btn1.getX()+btn1.getWidth(), btn1.getY()+btn1.getHeight()));
}
```

Figura 91: Validar Solapamiento Botones

Una vez validados los botones, se reducirá el tamaño del mapa a una unidad estándar (la cual se utilizará también en el videojuego). Esto sirve para que cada botón esté en el mismo lugar da igual cuales sean las resoluciones de pantalla. Esta medida estándar es 1280x720, y se calculará la relación entre el tamaño de la pantalla y estas dimensiones, para luego aplicárselo a cada medida o tamaño (Ver Figura 92).

```
float width = 1280f / panelMapa.getWidth();
float height = 720f / panelMapa.getHeight();

Float[][] medidas = new Float[botones.size()][4];
for(int i = 0; i < botones.size(); i++){
    medidas[i][0] = botones.get(i).getX()*width;
    medidas[i][1] = botones.get(i).getY()*height;
    medidas[i][2] = botones.get(i).getWidth()*width;
    medidas[i][3] = botones.get(i).getHeight()*height;
}
```

Figura 92: Relación Tamaños

Una vez hecho todo lo mencionado, se guardará tanto el mapa como los botones. Para ello, se creará el objeto mapa en el gestor *GestorMapa* y se le añadirán todos los botones mediante los *setters* (antes se tendrá que construir el *array* de botones). Por otro lado, en el *MapaDAO*, primero se insertará el mapa, y más tarde cada botón individualmente (recorriendo el *array* de botones del mapa). Una vez terminado la inserción, se cargarán todos los mapas de nuevo (al igual que pasaba con los fondos), se notificará a la ventana y se cerrará la ventana de carga.

Por último, a la hora de seleccionar algún mapa podremos modificar únicamente sus botones. Podremos eliminarlos, añadir nuevos, cambiar su tamaño o su posición, al igual que cuando se crearon. Además, al igual que con el fondo, en caso de que no exista ningún mapa, se creará uno predeterminado con 7 botones (con posiciones y tamaños predefinidos).

3.13.**Premios**

Esta ventana tiene las mismas características y funcionalidades que la ventana de *Fondo*. En este caso, el gestor será *GestorPremios*, y las consultas contra la base de datos las realizará mediante *PremiosDAO*. Por otro lado, las imágenes que se van a permitir obtener son aquellas con formato "*.png*" o "*.jpg*".

3.14.**Contenido**

La ventana de Contenido es la primera que contiene una tabla (a partir de aquí, las otras tres ventanas restantes también contienen alguna tabla). Además, como se puede ver en la Figura 93, esta ventana contendrá también una barra de búsqueda y tres botones.

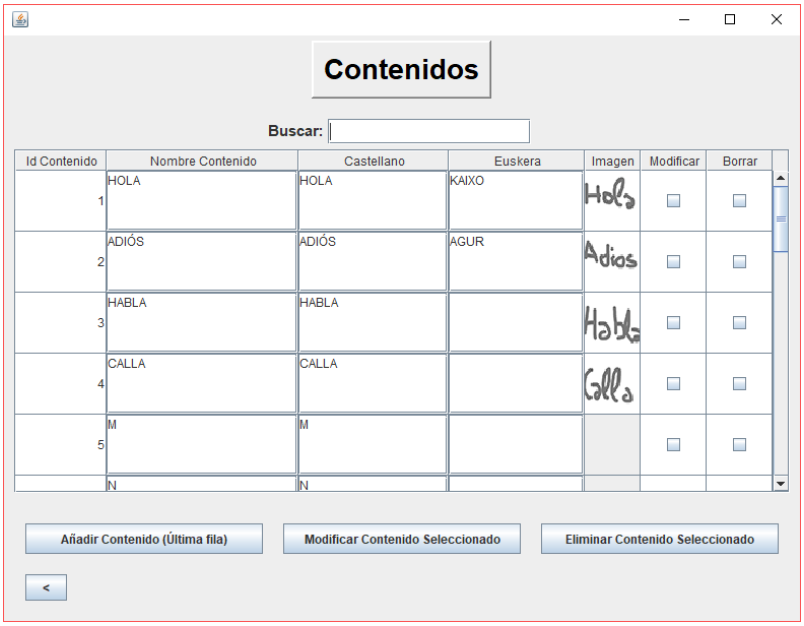


Figura 93: Ventana Contenido

Para lograr esto, se ha utilizado la clase *CustomJTable* explicada anteriormente. Para llamar a esta clase (mediante su constructora) se le mandan los datos directamente tras obtenerlos del gestor de la ventana (*GestorContenido*). En este caso, para configurar la tabla según nuestras necesidades, se ha fijado el tamaño de la primera y dos últimas columnas. Esto se ha hecho porque la primera columna será un valor fijo (id del contenido), y las otras dos columnas serán *CheckBox*. Por otro lado, se ha permitido que la tabla se pueda ordenar (siendo *sorteable*). En lo referente a las columnas que contienen texto a modificar ('Nombre Contenido', 'Castellano' y 'Euskera'), tendrán limitadas el número de caracteres que se pueden introducir. Por último, en el caso de la imagen, se ha limitado su tamaño y se ha puesto que sea una celda de tipo *FileChooserCellEditor* (subclase de la ventana). De esta forma, en caso de pulsar la casilla, se abrirá un selector de archivos (al igual que pasaba en la ventana de fondos, premios y mapas).

La subclase *FileChooserCellEditor* se ha creado como una subclase de la ventana debido a que era en el único lugar que era necesaria. Esta subclase implementará las clases *TableCellRenderer* y *TableCellEditor*, y extenderá de la clase *DefaultCellEditor*. Estas tres clases se utilizan para definir tanto la visualización como el comportamiento de cada celda al ser pulsadas. Dentro de la clase, el panel utilizado será *ImagePanel*, y al ser pulsada, se utilizará un *FileChooser* normal y corriente para obtener la imagen. En este caso, se podrán obtener tanto imágenes "*.jpg*", como "*.png*".

A la hora de obtener los datos en el gestor, se añadirá una fila más de las existentes. Esta última fila será utilizada para dar al usuario la opción de añadir nuevo contenido. Para realizar esta obtención de contenido, se recorrerá la lista de contenido del *GestorContenido* (la cual se obtuvo nada más cargar la pantalla, mientras se visualizaba una barra de carga), obteniendo cada campo, e introduciéndolo en su lugar. Además, se crearán dos columnas adicionales para las columnas 'Eliminar' y Modificar de la tabla (ambas de tipo *Boolean)* (Ver Figura 94).

```
public Object[][] obtenerContenido(){
    Object[][] cont = new Object[contenidos.size()+1][7];
    for(int i = 0; i < contenidos.size(); i++){
        cont[i][0] = contenidos.get(i).getIdContenido();
        cont[i][1] = contenidos.get(i).getNombre();
        cont[i][2] = contenidos.get(i).getCastellano();
        cont[i][3] = contenidos.get(i).getEuskera();
        cont[i][4] = new ImagePanel(contenidos.get(i).getImagen());
        cont[i][5] = new Boolean(false);
        cont[i][6] = new Boolean(false);
    }
    cont[cont.length-1][0] = null;
    cont[cont.length-1][1] = null;
    cont[cont.length-1][2] = null;
    cont[cont.length-1][3] = null;
    cont[cont.length-1][4] = new ImagePanel(null);
    cont[cont.length-1][5] = new Boolean(false);
    cont[cont.length-1][6] = new Boolean(false);
    return cont;
}
```

Figura 94: Obtener Datos Contenido

Otro punto a analizar es la barra de búsqueda. Esta barra será un campo de texto normal y corriente. La diferencia es que se le ha añadido un escuchador de documentos (*DocumentListener*) para que cuando modificamos el interior del campo se filtre en la tabla (Ver Figura 95).

```
filterText.getDocument().addDocumentListener(new DocumentListener() {

    @Override
    public void removeUpdate(DocumentEvent e) {
        table.newFilter(filterText.getText(), new int[] { 1, 2, 3 });
    }

    @Override
    public void insertUpdate(DocumentEvent e) {
        table.newFilter(filterText.getText(), new int[] { 1, 2, 3 });
    }

    @Override
    public void changedUpdate(DocumentEvent e) {
        table.newFilter(filterText.getText(), new int[] { 1, 2, 3 });
    }
});
```

Figura 95: Filtro Contenido

Por último, tenemos los tres botones inferiores (‘Añadir’, ‘Eliminar’ y ‘Modificar’).

- Añadir: Este botón añadirá un contenido nuevo. Para ello, habrá que rellenar la última fila de la tabla. Además, el nombre introducido no debe existir, y se debe rellenar como mínimo el campo ‘Castellano’ o ‘Euskera’. En caso de que alguna condición no se cumpla, se mostrará un error. Una vez verificadas todas estas restricciones, se mandará al *GestorContenido* y este lo insertará mediante *ContenidoDAO*. Una vez insertado, como en los demás casos, se obtendrá todos los datos de nuevo y se notificará a la ventana. Al igual que en los demás casos en los que se necesitan imágenes, en caso de seleccionar alguna imagen, esta será subida al servidor en la constructora del objeto *Contenido*.
- Modificar: Este botón modificará el contenido marcado mediante el *checkbox* correspondiente. En caso de no haber seleccionado ningún contenido, se mostrará un error. En caso contrario, se modificará mandando los datos mediante una matriz al *GestorContenido.* Además, para evitar la acumulación de imágenes basura en el servidor, en caso de que se haya modificado la imagen, se eliminará la imagen antigua tras realizar una modificación correcta.
- Eliminar: Este botón será encargado de eliminar el contenido seleccionado. Nada más pulsar el botón, se preguntará si realmente se quiere eliminar. En caso afirmativo, se comprobarán y recogerán el identificador de aquellos contenidos marcados. Si no existe ningún contenido marcado, se devolverá un error. Si se ha seleccionado algún contenido, esté será enviado al *GestorContenido* y será eliminado.

3.15.**Grupos**

La ventana de los grupos es una pantalla formada por un JC*omboBox* en la parte superior, una tabla y dos botones en la parte inferior (Ver Figura 96).

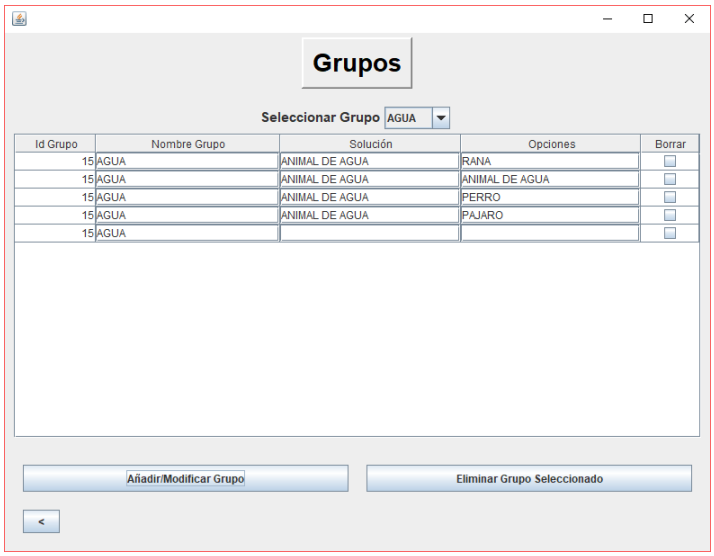


Figura 96: Ventana Grupo

El selector de la parte superior sirve para seleccionar el grupo que queremos modificar, o para añadir uno nuevo (para esto, hay que posicionarse en la primera posición del selector). La obtención de los nombres de los grupos se hace mediante el gestor de la pantalla (*GestorGrupos*), recorriendo la lista de los grupos (recolectados en la obtención de datos inicial), y cogiendo el nombre de cada uno. Además, para realizar la función de añadir un grupo nuevo, se ha añadido un ítem vacío al final de la lista.

Una vez seleccionado un ítem distinto al primero, se obtendrá su contenido mediante el *GestorGrupos*. Para ello, se obtendrá el contenido de la misma posición que el ítem seleccionado, pero quitando el espacio en blanco inicial.

La tabla que se visualiza está formada por cuatro columnas. La primera contendrá el identificador del grupo, y será fija. En caso de la última, la cual será también de tamaño fijo, en su interior se encontrarán *checkbox* para seleccionar los contenidos del grupo a eliminar. Las otras tres columnas contendrán el nombre del grupo (esta celda será la misma que el nombre del contenido), el contenido solución y el contenido opción. Las celdas de ambos contenidos serán *JComboBox*. De esta forma, podremos elegir entre el contenido que tengamos creado. Para crear estos JC*omboBox*, hay que modificar el editor y el renderizador de celdas. Para ello, se les asignará un nuevo *DefaultCellEditor*, con la lista de contenido como parámetros de la constructora (Ver Figura 97).

```
table.getColumnModel().getColumn(2).setCellEditor(new DefaultCellEditor(cbContenido));
```

Figura 97: Crear Tabla Grupo

Por otro lado, en caso de añadir un contenido nuevo al grupo, la última fila tendrá integrado el nombre y el identificador del grupo (y no serán modificables). A su vez, el contenido solución deberá ser el mismo para todo el grupo.

En lo que se refiere a los botones, tenemos dos posibilidades. En caso de pulsar el botón de Añadir/Modificar Grupo, añadirá o modificará el grupo en el que estamos. Esto dependerá de si existe algún dato en la tabla (modificar), o la tabla está vacía por defecto (añadir). Al pulsar al botón, comprobará los criterios antes mencionados en caso de tratarse de una modificación. Además, comprobará que el contenido opción no esté repetido. Por otro lado, al añadir un nuevo grupo, se comprobará que el nombre no esté repetido, y que se haya introducido tanto un contenido solución como un contenido opción.

Una vez que todo esté correcto, se preguntará si se quiere permitir a todos los usuarios (pacientes) poder utilizar este grupo en los minijuegos (Ver Figura 98). Da igual la opción que se pulse, se insertará el grupo mandándolo al gestor *GestorGrupo* y utilizando el *GrupoDAO* (al igual que se hace en las anteriores ventanas). La selección de agregar a todos los usuarios o no influirá una vez creado el grupo. En caso de haber pulsado el botón "Aceptar", se insertará a todos los usuarios con el identificador de este grupo en la tabla Permisos_Grupos (mediante una nueva llamada a base de datos).

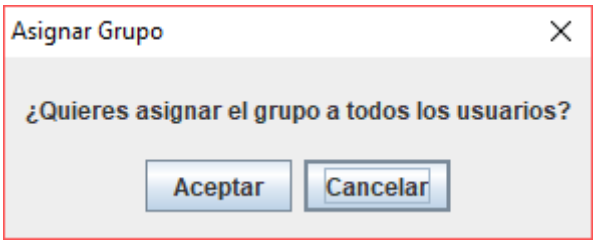


Figura 98: Asignar a todos

Por último, nos encontramos con el botón eliminar. Este botón permitirá eliminar un contenido de aquellos grupos que no estén asignados a ningún mininjuego. Esta restricción se ha decidido poner para evitar el máximo número de incoherencia de datos posibles. Por lo demás, la funcionalidad de este no se explicará debido a que es similar a la del botón eliminar de la ventana del contenido.

3.16.**Minijuegos**

En lo referente a la ventana de Minijuegos, esta contiene un JComboBox para seleccionar el minijuego que queremos modificar, una tabla donde aparecerán los grupos asignados al minijuego seleccionado, unos campos de texto para modificar el nombre y descripción del minijuego, y unos botones (Ver Figura 99). En caso de que se acceda por primera vez a esta ventana, se insertará por defecto la información de cada minijuego.

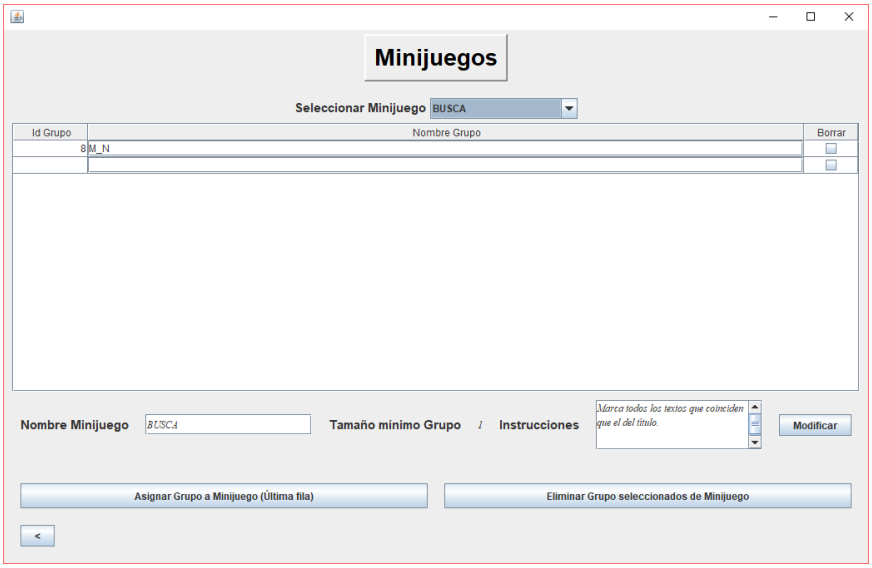


Figura 99: Ventana Minijuegos

En este caso, tanto la tabla como los botones referentes a ella funcionan de la misma forma que en la tabla de grupos. Por esto, estas no serán explicadas de nuevo. Lo que sí que hay que mencionar es la obtención de los grupos que se van a visualizar en el *JComboBox* de la columna central. Primero de todo, en este caso, de la lista de grupos disponibles, serán eliminados aquellos que ya están asignados. Por otro lado, debido a que no todos los grupos pueden formar parte de todos los minijuegos (un grupo con un texto grande como contenido no puede formar parte del minijuego *Buscar*, por ejemplo), a la hora de obtener los grupos disponibles se filtrará por el minijuego en el que estamos posicionados. Por esto, se realizarán los siguientes filtros por minijuegos:

- Buscar: El tamaño del texto del contenido (solución y opciones) es menor o igual a 2.
- Frase Correcta: La imagen de la solución existe.
- Laberinto Erróneo: No necesita grupos.
- Palabras Incompletas: La imagen de la solución existe y el contenido solución en castellano o euskera contiene un carácter '_'.
- Parejas: No se realiza ninguna comprobación
- Selecciona: Se comprueba que todo el contenido opciones contienen una imagen.
- Sopa de Letras: El texto del contenido solución en castellano y en euskera está compuesto por texto únicamente (sin símbolos ni números).

Una vez filtrado, podrán ocurrir diferentes cosas. La primera es que existan grupos para el minijuego, y se muestre la tabla. La segunda es el caso que no existan grupos para el minijuego porque no se han creado con las características necesarias. La última opción es la del minijuego 'Laberinto Erróneo', que no necesita de ningún grupo para funcionar.

Por último, se podrá modificar tanto el título del minijuego como la descripción que se va a ver como ayuda en cada minijuego. Para esto, se debe cambiar el valor en cada campo, y pulsar el botón modificar.

3.17.**Permisos**

En esta última pantalla podremos dar permisos a los usuarios (pacientes) tanto de los minijuegos como de los grupos. Debido a que se puede de ambas cosas, en la parte superior nos encontramos con un *JComboBox* para poder seleccionar uno u otro.

Grupos

Si seleccionamos el ítem grupos, se nos actualizará el segundo *JComboBox* superior y la tabla que tenemos (Ver Figura 100).

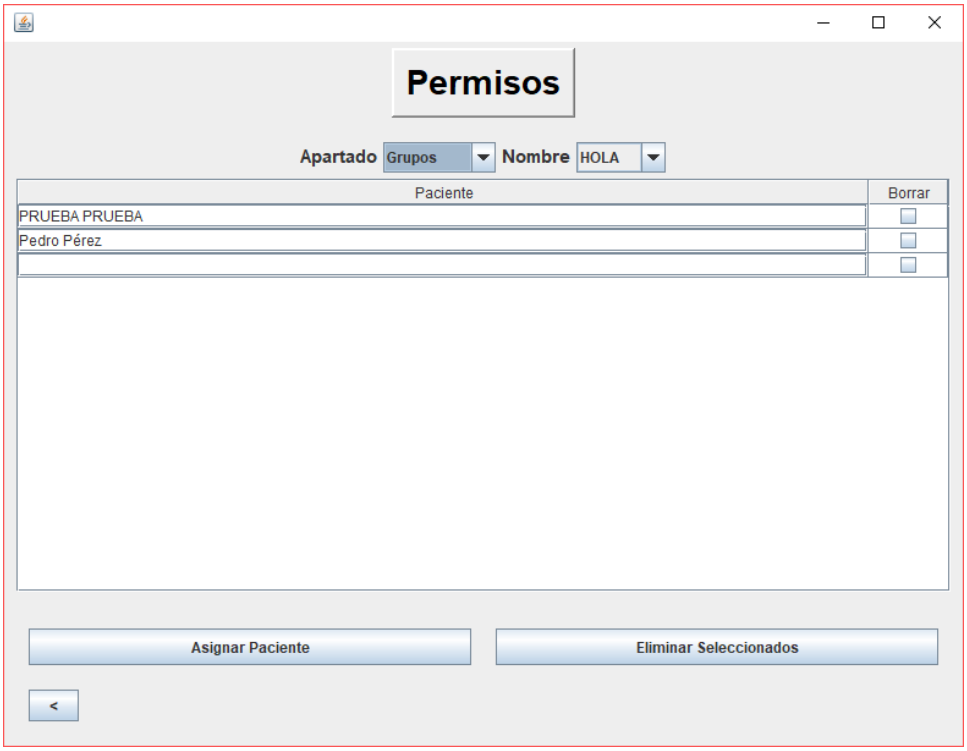


Figura 100: Permisos Grupos

Esta tabla contendrá dos columnas. En la primera podremos seleccionar los usuarios a los que queremos dar permisos en el minijuego del segundo *JComboBox*. Por otro lado, la segunda columna sirve para seleccionar aquellos usuarios a los que les queremos quitar el permiso. Debido a que la funcionalidad de añadir un nuevo usuario y de los botones es similar a otras pantallas, se omitirá en este caso.

Minijuegos

Si seleccionamos el ítem minijuegos, se nos actualizará el segundo *JComboBox* superior y la tabla que tenemos (Ver Figura 101).

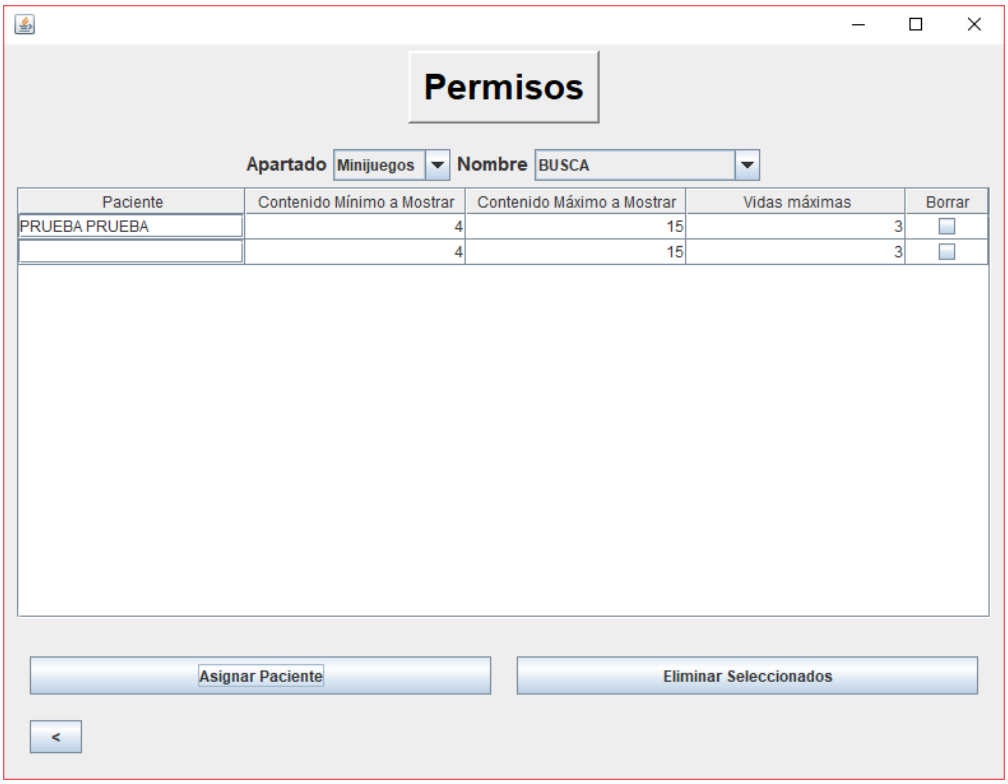


Figura 101: Permisos Minijuegos

En este caso, la tabla estará formada por cinco columnas. La primera será el usuario al que le queremos dar permisos (paciente). Las siguientes columnas son el contenido mínimo y máximo que se va a mostrar al jugar, y las vidas con las que se va a comenzar. Los parámetros máximo y mínimo se obtendrán de la base de datos, de la tabla MINIJUEGO. La diferencia será que, en el mínimo, los valores irán creciendo, desde el mínimo hasta el máximo de uno en uno. En caso del máximo, será al revés, decreciendo. Por último, las vidas se podrán elegir entre los valores del 3 al 21, aumentando de 3 en 3, es decir, los valores 3,6,9,12,15,18,21 (Ver Figura 102).

```
Integer[] min = new Integer[(int)datos[0][4]-(int)datos[0][2]];
for(int i = 0; i < min.length; i++) min[i] = (int)datos[0][2]+i;
Integer[] max = new Integer[(int)datos[0][4]-(int)datos[0][2]];
for(int i = 0; i < max.length; i++) max[i] = (int)datos[0][4]-i;
if(min.length == 0) min = new Integer[]{0};
if(max.length == 0) max = new Integer[]{0};
cbMaximo = new JComboBox<>(max);
cbMinimo = new JComboBox<>(min);
cbPuntuacion = new JComboBox<>(new Integer[]{3,6,9,12,15,18,21});
```

Figura 102: Máximo, mínimo y vidas

En el momento de la inserción o eliminación, estas funcionan de la misma forma que en las anteriores ventanas.

4. Videojuego Fase 2

La última parte del desarrollo engloba la parte faltante del videojuego. Observando lo comentado hasta el momento, podemos saber que el videojuego es la aplicación con la que los pacientes "jugarán" en casa. A su vez, esta está formada por diferentes minijuegos (los cuales han sido creados en la primera fase del videojuego, y queda de implementar su relación con la base de datos). Llegados a este punto, la base de datos está completamente diseñada y hecha, por lo que ya se puede proceder a la finalización del videojuego.

Por otro lado, hay que hacer un parón para aclarar que el videojuego, en un principio, se pensaba hacer parecido a un *tamagochi*, donde tendríamos una mascota virtual a la que cuidar. El problema vino en el apartado gráfico. Cuando llego la hora de realizar diferentes figuras en 3D, se vio el problema de que somos completamente programadores, por lo que nuestras capacidades artísticas son prácticamente nulas. Debido a esto, se pensó el realizar el otro tipo de videojuego (por niveles), y de dar la posibilidad a la logopeda de añadir los mapas y fondos. Debido a esto, la base de datos cambió, al igual que el programa de gestión. En ambos se añadió las ventanas y tablas relacionadas con el mapa o los botones, que ya han sido explicadas para evitar perderse en la lectura.

Para explicar las diferentes partes que contiene este apartado, se va a dividir esta fase en varios subapartados, los cuales se explicarán individualmente.

4.1. **Login**

La ventana de *Login* es la primera ventana que nos encontramos nada más entrar en la aplicación (tras pasar la animación de *Unity3D*, la cual no se puede modificar con la licencia gratuita). En esta ventana, se puede visualizar dos campos para la entrada de texto y un botón (Ver Figura 103).

Figura 103: Login Videojuego

El objetivo de esta ventana es la de comprobar las credenciales y encargarse de la sincronización. Esta segunda responsabilidad se va a explicar en el siguiente apartado debido a la dificultad que ha acarreado (se ha considerado la parte más compleja de todo el proyecto). Para esto, tras introducir los datos se comprobarán mediante una llamada a la función de "*LOGIN*" de la base de datos remota. En caso de que dichos datos sean correctos, se comenzará con la sincronización. En caso contrario, se devolverá un error de datos incorrectos. Además, en caso de que no se pueda conectar con la base de datos (por no estar conectado a internet o a la *VPN*), aparecerá un error de que no se puede conectar.

Por otro lado, si existe algún usuario conectado (existen sus datos en la base de datos remota), se comprobará si se puede sincronizar. En caso de conseguir sincronizar, o no poder hacerlo debido a un error en la conexión, se cambiará a la ventana del mapa.

Por último, todas estas acciones que se realizan contra la base de datos remota se realizan mediante hilos (*Thread*) (Ver Figura 104). Esto se hace así para evitar el consumo total de la memoria del dispositivo.

```
sync.setUsuario (user);
sync.setContrasena (password);
Thread t = new Thread (new ThreadStart (sync.login));
t.Start ();
```

Figura 104: Thread Login

Además, mientras se están comprobando los datos o se está sincronizando, en la pantalla aparecerá una nueva ventana. En esta ventana, aparecerá la palabra "Despertando" (o su traducción en euskera) y unos puntos suspensivos (Ver Figura 105). Estos puntos irán apareciendo según el tiempo pase (De tres puntos, pasarán a cero, y luego volverán a aumentar), y sirven para comprobar que la aplicación no se ha quedado bloqueada, y para hacer esta parte un poco más dinámica. Para conseguir este efecto, se ha utilizado el método *InvokeRepeating*. De esta forma, cada segundo desde el primer segundo, se llamará a la función *cambioCargando*, y esta se encargará de cambiar el texto de los puntos suspensivos.

Figura 105: Despertando

4.2. **Sincronización**

Como ya se ha dicho anteriormente, la sincronización de la aplicación ha resultado la parte más complicada de todo el proyecto. Esto es debido a que había que tener en cuenta varias acciones posibles en la utilización de la aplicación, lo que hacía el tema de la concurrencia un problema. Debido a esto, si a lo largo de todo el videojuego existe una incompatibilidad de datos, o datos incorrectos (como que no existan minijuegos o grupos suficientes para el minijuego que se vaya a jugar), se mandará al usuario a la ventana de Login con un error de comunicar un error de datos a la logopeda.

Una vez mencionado la información de los datos incorrectos, comenzaremos a explicar cómo se ha desarrollado esta parte. De forma general, para las sentencias SQL tanto remotas como locales se utiliza *StringBuilder* o *String*, dependiendo de la longitud de la consulta. En caso de ser consultas extensas, se ha utilizado *StringBuilder* porque es un constructor de *strings* que reduce el tamaño que estos ocupan. Además, elimina posibles caracteres erróneos en la concatenación. Esta clase es útil si se realizan concatenaciones en un *string*.

En cuanto se manda a la clase *Sync* sincronizar (clase encargada de la conexión y de cualquier acción contra la base de datos remota), está comenzará por obtener la fecha de la última sincronización del usuario. Además, se obtendrá la fecha de la última sincronización de los mapas y de los fondos. Estas tres acciones se realizan mediante el gestor de base de datos local (clase *GestorSQLite*) y la tabla de base de datos Tiempos_Imagenes. Debido a errores encontrados a la hora de utilizar el mismo formato de fecha obtenido de la base de datos local en la base de datos remota, se decidió el uso de fecha en formato *string.*

Una vez obtenidas las fechas, se sincronizará los datos existentes en local a remoto (si no existiera un usuario conectado, no se sincronizaría nada en esta parte). Estos datos se tratan del historial mapa, el historial juego y los premios (las tres tablas que se pueden alterar al jugar). Para realizar esto, se obtendrán los datos existentes mediante la clase *GestorSQLite*, y se introducirán en la base de datos remota (Ver Figura 106).

```
for (int i = 0; i < historial_juegos.Count; i++) {
    using (MySqlCommand cmdMysql = new MySqlCommand (insert.ToString (), conexion)) {
        cmdMysql.Parameters.AddWithValue ("@id_usuario", usu);
        cmdMysql.Parameters.AddWithValue ("@id_minijuego", historial_juegos [i] ["id_minijuego"]);
        cmdMysql.Parameters.AddWithValue ("@fecha", historial_juegos [i] ["fecha"]);
        cmdMysql.Parameters.AddWithValue ("@completo", historial_juegos [i] ["completo"]);
        cmdMysql.Parameters.AddWithValue ("@vidas", historial_juegos [i] ["vidas"]);
        cmdMysql.Parameters.AddWithValue ("@errores", historial_juegos [i] ["errores"]);
        cmdMysql.Parameters.AddWithValue ("@tiempo", historial_juegos [i] ["tiempo"]);
```

Figura 106: Actualizar Remoto

En caso del historial mapas, se utiliza una llamada a un procedimiento porque pueden existir inconsistencias entre los datos en remoto y los datos en local (porque el historial mapa se altera cada vez que se entra en el mapa, por lo que en dos dispositivos diferentes se altera con datos diferentes) (Ver Figura 107). En las tres actualizaciones, se obtiene el usuario existente en la base de datos antes de realizar la inserción.

```
using (MySqlCommand cmdMysql = new MySqlCommand ("SINCRONIZAR_HISTORIAL_MAPA", conexion)) {
    cmdMysql.CommandType = CommandType.StoredProcedure;
    cmdMysql.Parameters.AddWithValue ("@p_id_usuario", usu);
    cmdMysql.Parameters.AddWithValue ("@p_id_mapa", historial_mapas [i] ["id_mapa"]);
    cmdMysql.Parameters.AddWithValue ("@p_pos_mapa", historial_mapas [i] ["pos_mapa"]);
    cmdMysql.Parameters.AddWithValue ("@p_id_minijuego", historial_mapas [i] ["id_minijuego"]);
    cmdMysql.Parameters.AddWithValue ("@p_id_boton", historial_mapas [i] ["id_boton"]);
    cmdMysql.Parameters.AddWithValue ("@p_estado", historial_mapas [i] ["estado"]);
    cmdMysql.Parameters.Add ("@po_agregado", MySqlDbType.Int32).Direction = ParameterDirection.Output;
```

Figura 107: Actualización Procedimiento

Una vez realizada la actualización, se conectará a la base de datos local y se creará una transacción (para en caso de existir un problema, no guardar ningún dato al realizar *rollback*). Una vez hecho esto, comenzará la obtención de los datos. Las sentencias SQL utilizadas en esta obtención de datos dependerán de si se ha obtenido fechas de sincronización anteriormente. Esto se ha hecho introduciendo todas las sentencias en dos *arrays* (uno con las sentencias referentes a las tablas de imágenes fijas – Premios, Mapas y Fondos -, y el otra para el resto). La construcción de ambos *arrays* dependerá de si las variables de tiempo correspondiente son nulas o no (obtenidas al principio). En caso de ser nulas, se filtrarán los datos por la acción. De esta forma, los datos borrados (acción 'B') no se obtendrán. En caso de no ser nulas, se utilizará el tiempo, y se filtrará en la cláusula *WHERE* por el tiempo. Una vez construidos ambos *arrays*, se procederá a juntarlos todos en un mismo *StringBuilder*. Una vez hecho esto, se realizará la consulta.

En lo que se refiere a la obtención de los resultados, se irá recorriendo cada sentencia SQL con el comando *NextResult()* de la clase *MySqlDataReader*. Además, para cada consulta (cada vez que entre en el bucle), se comprobará si existen más datos mediante el comando *Read()* de la misma clase. Por último, se introducirán automáticamente cada columna en un *Dictionary*, utilizando como clave el nombre de la columna (obtenida mediante el comando *GetName*). Todos los campos se obtendrán como *String*. Por último, se insertará dicho *Dictionary* en otro *Dictionary*. En este caso, el valor será una lista de *Dictionary*, y la clave será el nombre de la tabla (guardada en una lista creada al comienzo de manera global) (Ver Figura 108)

```
do {
    while (reader.Read ()) {
        aux.Add (new Dictionary<String, object> ());
        for (int i = 0; i < reader.FieldCount; i++) {
            if (reader.IsDBNull (i)) {
                aux [aux.Count - 1].Add (reader.GetName (i), null);
            } else {
                aux [aux.Count - 1].Add (reader.GetName (i), reader.GetString (i));
            }
        }
    }
    list.Add (nombre_tablas [k], aux);
    aux = new List<Dictionary<String, object>> ();
    k++;
} while (reader.NextResult ());
```

Figura 108: Obtención Datos Sincronización

Una vez obtenidos todos los datos, se recorrerá una lista de acciones creadas globalmente (Ver Figura 109). Gracias a esta lista, se podrá realizar automáticamente la inserción de todos los datos (sin tener que ir tabla a tabla llamando a su método).

```
sinc = new Action[] {
    () => sqlite.sincronizarUsuario (),
    () => sqlite.sincronizarContenido (),
    () => sqlite.sincronizarGrupo (),
    () => sqlite.sincronizarMinijuego (),
    () => sqlite.sincronizarGrupoMinijuego (),
    () => sqlite.sincronizarPremios (),
    () => sqlite.sincronizarFondos (),
    () => sqlite.sincronizarMapas (),
    () => sqlite.sincronizarBotones (),
    () => sqlite.sincronizarHistorial_Mapas ()
};
```

Figura 109: Lista Acciones

A la vez que se recorre esta lista, se comprobará si para cada tabla se ha obtenido algún resultado. En caso de existir datos, se llamará a su acción correspondiente mediante Hilos (en este caso, se utilizan hilos para poder realizar todas las inserciones de forma asíncrona). Estas acciones se corresponden con funciones de la clase *GestorSQLite*. En caso contrario, si no es la primera sincronización y no es ni la tabla historial_mapas ni historial_juegos, se producirá un error. En caso de las tablas con imágenes fijas, se creará dicho error si nunca se han sincronizado, es decir, si no existen datos en ellas (Ver Figura 110). Al crear el error, se cancelarán todos los hilos, los cuales están todos guardados en una lista. La sincronización finalizará o cuando terminen todos los hilos, o cuando exista un error.

```
for (int i = 0; i < sinc.Length && error_code == 0; i++) {
    if (list.ContainsKey (nombre_tablas [i]) && list [nombre_tablas [i]].Count > 0) {
        Thread t = new Thread (new ThreadStart (sinc [i]));
        t.Start ();
        threads.Add (t);
    } else {
        if (!nombre_tablas [i].Equals ("historial_mapas") && !nombre_tablas [i].Equals ("historial_juego") && fechaSincro == null) {
            switch (nombre_tablas [i]) {
            case "mapas":
                if (fechaMapas == null)
                    error_code = -3;
                break;
            case "fondos":
                if (fechaFondos == null)
                    error_code = -3;
                break;
            default:
                error_code = -3;
                break;
            }
        }
    }
}
```

Figura 110: Sincronización

En lo referente a la inserción de datos, se realiza en todas las tablas de forma similar. Lo primero que se hace, es obtener el *Dictionary* correspondiente a la tabla (se obtiene desde el método llamado al crear cada hilo). Una vez obtenidos los datos, se recorrerán para proceder a su inserción. Dependiendo de si el atributo *ACCIÓN* obtenido es 'A', 'M' o 'B', realizará una acción u otra. En caso de 'A' o 'M', realizará un *REPLACE* sobre la tabla correspondiente. En caso de 'B', un *DELETE*. Por último, se comprobará en aquellos campos que sean imágenes si son nulos. En caso de no serlo, se obtendrá la imagen correspondiente a la dirección URL guardada mediante el método *obtenerImagen* de la clase *Constantes* (esta clase utilizará la clase WebClient y el método *DownloadData* para obtener un *array* de *bytes)*. Para hacerse una idea mejor, se va a utilizar la sincronización de la tabla Contenido como ejemplo (Ver Figura 111). En caso de existir algún error por el camino, se cancelarán todos los hilos y se mostrará el error mencionado anteriormente.

```
List<Dictionary<string, object>> contenido = listASincronizar ["contenido"];
try {
    conectar ();
    String sqlDelete = "DELETE FROM CONTENIDO WHERE C_ID_CONTENIDO = :id_contenido";
    String sqlInsert = "INSERT OR REPLACE INTO CONTENIDO (C_ID_CONTENIDO, C_CASTELLANO, C_EUSKERA, C_IMAGEN) VALUES (:id_contenido, :castellano, :euskera, :imagen)";
    for (int i = 0; i < contenido.Count; i++) {
        if (conexion.State == ConnectionState.Executing) {
            i--;
        } else {
            if (Constantes.quitarNull (contenido [i] ["ACCION"]).Equals ("B")) {
                using (SqliteCommand cmd = new SqliteCommand (sqlDelete, conexion)) {
                    cmd.Parameters.Add (new SqliteParameter ("id_contenido", contenido [i] ["ID_CONTENIDO"]));
                    cmd.Dispose ();
                }
            } else {
                if (!Constantes.quitarNull (contenido [i] ["ID_CONTENIDO"]).Equals ("")) {
                    using (SqliteCommand cmd = new SqliteCommand (sqlInsert, conexion)) {
                        cmd.Parameters.Add (new SqliteParameter ("id_contenido", contenido [i] ["ID_CONTENIDO"]));
                        cmd.Parameters.Add (new SqliteParameter ("castellano", contenido [i] ["CASTELLANO"]));
                        cmd.Parameters.Add (new SqliteParameter ("euskera", contenido [i] ["EUSKERA"]));
                        cmd.Parameters.Add (new SqliteParameter ("imagen", Constantes.getConstantes ().obtenerImagen (contenido [i] ["IMAGEN"])));
                        cmd.ExecuteNonQuery ();
                        cmd.Dispose ();
                    }
                }
            }
        }
    }
} catch (Exception e) {
    Debug.Log (e);
    Sync.getSync ().setErrorCode (-1);
}
```

Figura 111: Inserción Datos Sincronización

Una vez terminada la sincronización, si todo ha ido correcto, se actualizarán todas las fechas de sincronización y se realizará *commit* de la transacción. La fecha utilizada se ha creado al inicio de la sincronización. La razón de crearla al inicio, y no en el momento de la inserción, es porque si se han actualizado los datos remotos durante la sincronización, para la próxima sincronización no se van a recoger. En caso de existir algún error por el camino, se realizará *rollback*.

4.3. **Mapa**

Nada más entrar en la ventana del mapa, en caso de que no se provenga de la pantalla de *Login*, se intentará sincronizar. Mientras se está realizando esta acción, al igual que pasaba en el *Login*, se visualizará una ventana con puntos suspensivos, pero con el mensaje "Recuperando Energía" (o su traducción en euskera) como texto (Ver Figura 112). Esta sincronización se realizará de la misma manera que desde la ventana de *Login*.

Figura 112: Recuperando Energía

Una vez terminada la sincronización (o nada más entrar a la ventana si es desde la pantalla de *Login*) se obtendrán los datos del mapa. Estos datos residen en la tabla Historial_Mapa. Debido a que al mismo tiempo se van a poder visualizar tres mapas (junto con sus botones), es necesario que en Historial_Mapa estén estos tres mapas, junto con sus botones con el estado de 'Actual' o 'Futuro' (el estado 'Pasado' sirve para los botones que ya se han completado, los cuales no se van a mostrar). En caso de no existir tres mapas, se crearán tantos como sean necesarios para cumplir este número Para obtener los tres mapas, se utilizará la cláusula "*LIMIT 0,3*", "*ORDER BY POS_MAPA ASC*" y "GROUP BY POS_MAPA" en la *SELECT* para obtener los tres últimos diferentes mapas (Ver Figura 113).

```
String selectCount = "SELECT HM_POS_MAPA FROM V_MAPAS GROUP BY HM_POS_MAPA ORDER BY HM_POS_MAPA ASC LIMIT 0,3;";
```

Figura 113: Select Mapas

En caso de que nos devuelva un resultado inferior a 3 (la vista *V_MAPAS* solo devuelve los resultados con un estado diferente a 'Pasado'), se insertarán nuevos campos. Para ello, se obtendrá el máximo POS_MAPA (para insertar el siguiente aumentado en 1) y se elegirá aleatoriamente un mapa.

```
String selectMapas = "SELECT MA_ID_MAPA FROM MAPA ORDER BY RANDOM() LIMIT 0,1;";
```

Figura 114: Obtención Mapa Aleatorio

Una vez hecho esto, se obtendrán los botones del mapa. A continuación, se insertará los datos en la tabla Historial_Mapas, dando al minijuego un valor aleatorio obtenido de la vista V_Minijuegos (Vista que muestra la lista de minijuegos que contienen un número suficiente de grupos para jugarlos).

Una vez creados los tres mapas, se comprobará de que existe algún mapa con el estado 'Actual'. En caso de no existir, se cambiará el estado de todos los botones del primer mapa obtenido con estado 'Futuro' por el estado 'Actual'.

Una vez hecho esto, se obtendrán la información de cada POS_MAPA guardado y se gestionará por la clase Mapa (la clase que gestiona como se va a ver el mapa y qué hacer cuando se pulse en cualquier botón). Además, al comienzo de esta obtención del historial del mapa, se modificarán todos los minijuegos asociados a un botón con un estado 'Futuro' (esto se hace para que los minijuegos de cada botón no estén fijados al crear el mapa por primera vez). También se cambiarán aquellos botones que tengan asignado un minijuego que ya no tiene el usuario asignado o que no se puede jugar (porque no tenga suficientes grupos asignados). Para hacer esto, primero se obtendrán dichos botones (con su mapa y su pos_mapa), y luego se insertará seleccionándolo de todos los minijuegos aleatoriamente (al igual que la primera vez que se inserta en Historial_Mapas explicado en el párrafo anterior).

Una vez que la clase *Mapa* obtiene los datos, el siguiente paso es gestionar dichos datos. Para esto, se obtendrán los diferentes botones y mapas que existen. De esta forma, se obtendrán las imágenes de los mapas y las medidas de los botones . En caso de los mapas, al igual que va a pasar con cualquier imagen proveniente de base de datos, hay que crear la imagen (*Texture2D*) en el mismo momento que obtenemos el *array* de *bytes* de la base de datos. Para esto, primero se creará una *Texture2D*, y luego se introducirá la imagen mediante el comando *LoadImage*, que tiene como parámetro un *array* de *bytes (*Ver Figura 115).

```
mapas [i] = new Texture2D (width, height);
mapas [i].LoadImage ((byte[])reader.GetValue (0));
mapas [i].Apply ();
```

Figura 115: Obtener Imagen

Por otro lado, está la obtención de las medidas de los botones. En este caso, se utiliza *Dictionary* para guardar los botones. Este *Dictionary* contendrá una lista con el id del botón, el Rect formado por su posición y su tamaño (el cual se transformará según el tamaño de la pantalla para igualar las posiciones que tenían en el mapa de la logopeda), el estado ('Pasado', 'Actual' o 'Futuro') y el id del minijuego.

Por último, también se obtienen los premios del usuario. Esto no necesita mucha explicación debido a que se realiza una simple *SELECT* de todos los premios, junto con su atributo de conseguido. Además, se generan todos los *Rects* (del mapa, de los premios y de opciones). En caso del *Rect* del mapa, ocurre algo similar al utilizado en el minijuego *Selecciona*. Para refrescar la memoria, este minijuego utilizaba una *scrollBar*, por lo que las posiciones de los elementos en su interior eran posiciones superiores a la pantalla. En este caso, todos los mapas ocuparán la pantalla completa. Además, el primer mapa estará posicionado en el punto (0,0). El siguiente mapa se encontrará en la posición (0, altura de pantalla), y así sucesivamente. Por último, a parte de los mapas con botones creados por la logopeda, se visualizará un mapa más en la pantalla. Este mapa será el último de todos, no contendrá ningún botón, y simulará una niebla (como si los siguientes mapas no están disponibles de momento). Para no notar el corte entre los mapas, se ha puesto también una muralla entre ellos. De esta forma, se fomenta el objetivo de los minijuegos de cada mapa (obtener las llaves para abrir la puerta). Esta muralla ha sido creada combinando diferentes partes de murallas del videojuego *Age Of Empire*. Con todo esto, esta ventana se verá como se muestra en la Figura 116 (En este caso, la captura se ha realizado entre dos mapas, para poder visualizar los diferentes botones disponibles y la muralla).

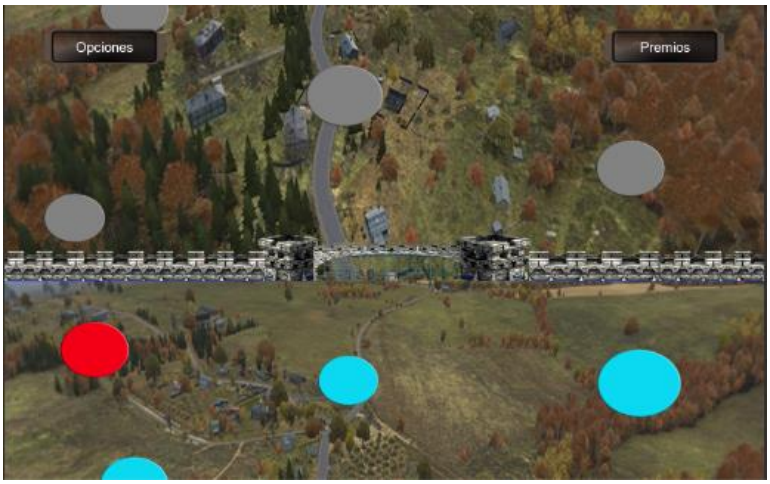


Figura 116: Mapa

Como se puede ver en la Figura 116, dependiendo del estado del botón, este cogerá un color u otro. Esto, al igual que la posibilidad de moverse verticalmente por el mapa, se ha conseguido gracias a la función *OnGUI*. Como se ha mencionado en el párrafo anterior, se ha utilizado, al igual que en el minijuego *Seleccionar*, una *scrollBar*. Para esto, Se ha utilizado una *ScrollView* (a la cual, se le pasan como parámetros el Rect de posición, la posición de la barra de *scroll*, el *Rect* de lo que se va a ver, que no se vean siempre la barra de scroll vertical y horizontal, y el estilo invisible que toma las dos barras de scroll en caso de que se vieran) (Ver Figura 117).

```
scrollPosition = GUI.BeginScrollView (new Rect (0, 0, Screen.width, Screen.height), scrollPosition, new Rect (0, rectMapas [rectMapas.Length - 1].y,
                                      Screen.width, Screen.height * rectMapas.Length), false, false, GUIStyle.none, GUIStyle.none);
```

Figura 117: Scroll Mapa

Por otro lado, se creará un grupo donde se pintará el mapa, y se mostrará en rojo el botón, y como un botón (*GUIButton*) en caso de que esté con el estado 'Actual', en azul y como una caja (*GUIBox*) en caso de que esté 'Pasado', y en gris y como una caja (*GUIBox*) en caso de que sea 'Futuro'. Este grupo se crea de una forma similar al minijuego *Seleccionar*.

Por último, en lo referente al mapa, están las acciones de los botones. Al pulsar en un botón rojo, se Instanciará el *Prefab* (*GameObject* – Objeto de *Unity3D*) correspondiente al minijuego que tiene asociado dicho botón. Para esto, se utilizará el id del minijuego, buscando su *Prefab* gracias al *array* de minijuegos creado en la clase Constantes creado. (Ver Figura 118).

```
minijuegos = new Dictionary<int, string> () {
    { 1, "Busca" },
    { 2, "FraseCorrecta" },
    { 3, "LaberintoErroneo" },
    { 4, "PalabrasIncompletas" },
    { 5, "Parejas" },
    { 6, "Selecciona" },
    { 7, "SopaDeLetras" }
};
```

Figura 118: Referencia Prefabs

En este punto, solo quedan dos cosas de explicar. La primera, es lo relacionado a los premios. Al pulsar en el botón de "Premios", se nos mostrará una pantalla donde podremos ver los premios desbloqueados, y los que aún tenemos que desbloquear (estos aparecerán con su silueta en negro) (Ver Figura 119). Este efecto de oscurecer se ha realizado convirtiendo todos los pixeles en los que existe algún color a negro. De esta forma, sólo se dibujará la silueta.

Figura 119: Premios

La segunda cosa a explicar es el menú de opciones. En él se puede modificar el idioma y desconectarse. Al modificar idioma sólo influirá en el mismo juego (no tendrá efectos en la base de datos remota). Por otro lado, el botón de desconectar intentará actualizar los datos de la base de datos remota (al igual que en la sincronización). Esto se hace para evitar la pérdida de datos. Si se consigue dicha actualización, se borrarán los datos de todas las tablas a excepción de la tabla de Mapas, Premios y Fondos, y nos dirigirá a la pantalla de *Login*. En caso contrario, se volverá al mapa.

4.4. **Minijuegos**

La última parte a explicar se trata la parte relacionada con la base de datos de los minijuegos. Debido a que ya se ha explicado en la fase 1 del videojuego cómo están formados los diferentes minijuegos, en esta parte únicamente se comentarán las sentencias utilizadas para la obtención de los datos, y las acciones a realizar cuando se llama a la función *terminarJuego* de la clase *Minijuego*.

En lo que se refiere a la función *terminarJuego*, es una función encargada de gestionar la finalización del mismo e insertar los datos correspondientes en la tabla Historial_Juegos. Para ello, primero comprobará si se ha completado el juego, es decir, si se ha finalizado el juego con alguna vida. Para ello, se comprobará si la variable puntación es mayor a 0. Por otro lado, se calcularán el número de vidas.

Una vez obtenidos todos estos datos, se insertará en la base de datos mediante la clase *GestorSQLite*. Este proceso de inserción incluye tanto la inserción en la tabla en la tabla Historial_Juegos, como la actualización del botón correspondiente en el mapa en caso de que el juego se haya completado. Además, en caso de que se actualice el botón y que sea el último botón del mapa, se actualizará el estado de todos los botones del siguiente mapa a 'Actual'. Debido a que, en caso de error, puede ocasionar un problema posteriormente (por existir unos datos y no otros), lo relacionado con el mapa será realizado en una misma transacción.

Una vez finalizada la inserción/actualización, el siguiente paso es el de crear la posibilidad de haber ganado un premio en caso de completar el nivel. Para ello, se obtendrá un número aleatorio del 0 al 5 (ambos incluidos), y se comprobará si dicho número es 0 (un 20% de posibilidades). En dicho caso, se obtendrá un premio aleatoriamente (mediante la cláusula *ORDER BY RANDOM()*). Este premio puede que haya sido ya obtenido por el usuario, o que sea un premio nuevo (todo depende del azar). Dependiendo de si se ha conseguido premio, y si se ha terminado con vidas, la ventana que se muestre será diferente. Por último, se explicará cómo se ha realizado la obtención de datos en cada minijuego. En todos ellos, los datos se obtendrán ordenados por la función *RANDOM*, es decir, desordenados.

4.4.1. Buscar

En este minijuego, se obtendrá el texto de la solución y de una de las opciones de cada grupo. Debido a que este minijuego sólo está formado por un grupo, se limitará a sólo obtener un resultado con la etiqueta "*LIMIT 0,1*".

4.4.2. Frase Correcta

En este minijuego, se obtendrá de la base de datos el id del grupo, el id de cada contenido, la imagen solución, y el texto solución y opción. Una vez obtenidos, se almacenarán en un Dictionary de listas de objetos dependiendo de si el id del contenido solución coincide con el de la opción. Además, solo se obtendrán 4 opciones (entra las que estará la solución) (Ver Figura 120).

```
int id_grupo = reader.GetInt32 (0);
int id_contenido_solucion = reader.GetInt32 (1);
int id_contenido_opcion = reader.GetInt32 (4);
if (!datosAux.ContainsKey (id_grupo)) {
    datosAux.Add (id_grupo, new Dictionary<String, List<object>> ());
    datosAux [id_grupo].Add ("Opciones", new List<object> ());
}
if (datosAux [id_grupo] ["Opciones"].Count < 3
    || (datosAux [id_grupo] ["Opciones"].Count == 3 && datosAux [id_grupo].ContainsKey ("Solucion"))
    || (datosAux [id_grupo] ["Opciones"].Count == 3 && id_contenido_opcion == id_contenido_solucion)) {
    datosAux [id_grupo] ["Opciones"].Add (reader.GetValue (5));
}
if (id_contenido_opcion == id_contenido_solucion) {
    datosAux [id_grupo].Add ("Enunciado", new List<object> ());
    Texture2D t = new Texture2D (1, 1);
    t.LoadImage ((byte[])reader.GetValue (2));
    t.Apply ();
    datosAux [id_grupo] ["Enunciado"].Add (t);
    datosAux [id_grupo].Add ("Solucion", new List<object> ());
    datosAux [id_grupo] ["Solucion"].Add (datosAux [id_grupo] ["Opciones"].Count - 1);
}
```

Figura 120: Obtener Frase Correcta

Una vez obtenidos los datos, se cogerán los primeros "n" grupos. Este valor n será el valor aleatorio entre el valor mínimo puesto por la logopeda, y el valor mínimo entre el valor máximo puesto por la logopeda y el tamaño de la lista (Para evitar que la lista sea mayor a los datos disponibles) (Ver Figura 121).

```
maximo = Mathf.Min (maximo, datosAux.Count);
int total = UnityEngine.Random.Range (minimo, maximo);
datos.AddRange (datosAux.Values);
datos.RemoveRange (total, (datos.Count - total));
```

Figura 121: Limitar Frase Correcta

4.4.3. Laberinto Erróneo

No se necesita ninguna obtención de datos, por lo que, en esta fase no se ha modificado.

4.4.4. Palabras Incompletas

En este minijuego, se obtendrá el texto e imagen del enunciado, y un texto aleatorio de entre todas las posibles opciones. Además, se comprobará que la solución tenga como mínimo un carácter "_" para poder realizar el hueco a rellenar. El texto se guardará en mayúsculas y eliminando tildes o acentos (Ver Figura 122). Una vez obtenidos los datos, se filtrará el tamaño como en los otros minijuegos.

```
fila [0] = reader.GetString (0).Trim ().ToUpper ().Replace (" ", "").Replace ("Á", "A")
        .Replace ("É", "E").Replace ("Í", "I").Replace ("Ó", "O").Replace ("Ú", "U");
```

Figura 122: Eliminar acentos

4.4.5. Parejas

En este minijuego, se obtendrán el texto e imagen de las soluciones y se guardará en una matriz. Una vez obtenidos los datos, se cogerán los primeros "n" grupos al igual que en los minijuegos anteriores. Además, se comprobará si entre el contenido obtenido, todos contienen imagen. En caso de que alguno no contenga, se eliminarán todas las imágenes de la matriz.

4.4.6. Selecciona

Como se ha mencionado en la definición y explicación de este minijuego, tiene una cierta similitud con el minijuego Frase Correcta. Debido a esto, la obtención de datos será similar, pero cambiando las imágenes por texto y viceversa.

4.4.7. Sopa de Letras

En este minijuego, se obtendrán el texto e imagen de las soluciones y se guardará en una matriz. En caso del texto, al igual que en el minijuego *Palabras Incompletas*, se guardará en mayúsculas y eliminando todas las tildes o acentos. La gestión de los datos y de las imágenes se realizará de la misma forma que en el minijuego *Parejas*.

Verificación y Evaluación

En este apartado se llevará a cabo la exposición del plan de pruebas realizado. Para ello, se realizarán varias tablas (cada una para cada requerimiento). Estas tablas estarán formadas por un identificador de la prueba (utilizado para identificar la prueba en caso de no pasarla), cómo se va a probar, resultados esperados, resultados obtenidos y si ha sido correcta o la prueba (I -> Incorrecta, C -> Correcta). Una vez terminadas las pruebas de cada apartado, se comentarán las soluciones que se han dado a las diferentes pruebas que no han sido correctas (se identificarán con el id).

A pesar de que el programa utilizado para realizar el videojuego es *Unity3D*, para realizar las pruebas se ha utilizado el programa *Android Studio*. Esto ha sido porque esta herramienta contiene un apartado de *log*, donde podremos ver los errores que la aplicación pueda dar.

1. Login

Se verificará tanto la comprobación de los datos de usuarios en el videojuego como en el programa.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Introducir un usuario o contraseña incorrecta en el videojuego	Mensaje de error	Mensaje de error	C
2	Introducir un usuario o contraseña correcto en el videojuego	Inicia Sincronización	Inicia Sincronización	C
3	Introducir un usuario o contraseña incorrecta en el programa	Mensaje de error	Mensaje de error	C
4	Introducir un usuario o contraseña correcto en el programa	Se muestra la ventana del menú	Se muestra la ventana del menú	C

Tabla 3: Pruebas Login

2. Ventana Sesión

Se verificará la inserción de los niños, con sus parámetros, la obtención de los niños, la visualización del historial de minijuegos, y el historial de sesiones. Este último se comprobará con una cuenta de logopeda y de tutor.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Rellenar los datos del niño (como mínimo los obligatorios)	Se agrega el niño	Da un error de formato de fecha.	I
2	Seleccionar un niño en el *JComboBox*	Se muestra la información del niño	Se muestra la información del niño	C
3	Añadir una sesión que pueda ver los tutores, y otra que no. Luego, comprobar con la cuenta del tutor que sólo tiene visible la correspondiente	Solo se observa la sesión que se ha puesto visible para el tutor	Solo se observa la sesión que se ha puesto visible para el tutor	C
4	Completar algún nivel en el videojuego (algún minijuego) y observar la pestaña historial minijuegos de la ventana historial sesión	Se muestra la información sobre el minijuego jugado	Se muestra la información sobre el minijuego jugado	C

Tabla 4: Pruebas Ventana Sesión

- Prueba 1: El error residía en que no se recogían correctamente los diferentes campos que se pueden rellenar. Esto causaba un error al intentar convertir un texto en fecha. Para solucionarlo, se recogieron correctamente los campos.

3. Ventana Modificar Datos

Se verificará que se guarda la información cambiada y que si se entra con una cuenta de tutor aparecerán los campos para la fecha de nacimiento y teléfonos.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Cambiar algún dato y pulsar a guardar	Los datos se guardan	Los datos se guardan	C
2	Entrar con una cuenta de tutor, y con una cuenta de logopeda	Con la cuenta del tutor aparecen los tres campos de más	Con la cuenta del tutor aparecen los tres campos de más	C

Tabla 5: Pruebas Ventana Modificar Datos

4. Ventana Fondos

Se verificará la inserción de los fondos, la comprobación del nombre y la visualización de los fondos.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Se escribe un nombre de fondo que no exista y guardar una imagen seleccionada	El fondo se guarda	El fondo se guarda, pero no se guarda correctamente la imagen.	I
2	Se escribe un nombre de fondo que existe y guardar una imagen seleccionada	Aparece un error diciendo que el nombre ya existe	Aparece un error diciendo que el nombre ya existe	C
3	Seleccionar un fondo en el *JComboBox*	Se visualiza el fondo y su nombre	Se visualiza el fondo y su nombre	C

Tabla 6: Pruebas Ventana Fondos

- Prueba 1: El error residía en que, a la hora de guardar el archivo mediante el PHP, este guardaba la imagen sin extensión y sin antes decodificarlo de Base64. Esto causaba que la imagen se guardara, pero de forma incorrecta. Para solucionarlo, se descodificaron las imágenes antes de guardarlas (en el PHP) y se pone la extensión que se le pasa como parámetro.

5. Ventana Mapas

Se verificará la inserción de los fondos, la comprobación del nombre y la visualización de los fondos.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Se escribe un nombre de mapa que no exista, y guardar una imagen seleccionada e introducir como mínimo un botón	El mapa y los botones se guardan	Da un error.	I
2	Se escribe un nombre de mapa que exista, y guardar una imagen seleccionada e introducir como mínimo un botón	Aparece un error diciendo que el nombre ya existe	Aparece un error diciendo que el nombre ya existe	C
3	Seleccionar un mapa en el *JComboBox*	Se visualiza el mapa, su nombre y los botones que se le habían creado en sus posiciones	Se visualiza el mapa, su nombre y los botones que se le habían creado en sus posiciones	C
4	Cambiar el tamaño de un botón y pulsar en “Modificar Mapa” y volver a cargar el mapa	La posición del botón es la modificada	La posición del botón es la modificada	C
5	Crear un botón encima de otro y pulsar en “Crear Mapa”/”Modificar Mapa”	Aparece un error diciendo que los botones no se pueden superponer.	El botón se modifica/crea.	I

Tabla 7: Pruebas Ventana Mapas

- Prueba 1: El error residía en que no se guardaba el mapa, por lo que, a la hora de guardar los botones, estos no tenían un identificador de mapa y daba error. Para solucionarlo, se ha insertado el mapa.

- Prueba 5: El error residía en que no se comprobaba correctamente si dos botones estaban en el mismo lugar o se superponían (solo se comprobaba la posición). Se solucionó calculando la posición que ocupa cada botón mediante su posición y tamaño.

6. Ventana Premios

Se verificará la inserción de los premios, la comprobación del nombre y la visualización de los premios.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Se escribe un nombre de premios que no exista y guardar una imagen seleccionada	El fondo se guarda	El premios se guarda	C
2	Se escribe un nombre de premios que existe y guardar una imagen seleccionada	Aparece un error diciendo que el nombre ya existe	Aparece un error diciendo que el nombre ya existe	C
3	Seleccionar un fondo en el *JComboBox*	Se visualiza el fondo y su nombre	Se visualiza el fondo y su nombre	C

Tabla 8: Pruebas Ventana Premios

7. Ventana Contenido

Se verificará la inserción, modificación y eliminación del contenido.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Se inserta un contenido sin rellenar el campo castellano y euskera	Aparece un error diciendo que se deben rellenar uno de los dos campos	Aparece un error diciendo que se deben rellenar uno de los dos campos	C
2	Se insertará un contenido con el nombre repetido	Aparece un error diciendo que el nombre no puede estar repetido	Aparece un error diciendo que el nombre no puede estar repetido	C
3	Se insertará un contenido con imagen	Se inserta correctamente y se muestra	Se inserta correctamente y se muestra	C
4	Se insertará un contenido sin imagen	Se inserta correctamente y se muestra	Se insertará correctamente y se muestra	C
5	Se modificará un contenido (cualquier campo) y se selecciona a modificar	Se modifica correctamente y se actualiza la tabla	Se modifica correctamente y se actualiza	C
6	Se modificará un contenido (cualquier campo) y no se selecciona ningún contenido a modificar	No se modifica	No se modifica	C
7	Se modificará un contenido (cualquier campo) y se selecciona otro contenido a modificar	No se modifica	No se modifica	C
8	Se selecciona un contenido a eliminar	Se elimina el contenido y se actualiza la tabla	Se elimina el contenido y se actualiza la tabla	C

Tabla 9: Pruebas Ventana Contenido

8. Ventana Grupos

Se verificará la inserción, modificación y eliminación del grupo.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Se inserta un contenido opción repetido en un grupo	Aparece un error diciendo que el contenido opción no se puede repetir	Aparece un error diciendo que el contenido opción no se puede repetir	C
2	Se insertará un grupo con el nombre repetido	Aparece un error diciendo que el nombre no puede estar repetido	Aparece un error diciendo que el nombre no puede estar repetido	C
4	Se modificará un grupo que está asignado en algún grupo	Aparece un error diciendo que el grupo no puede estar asignado en ningún minijuego para ser modificado	Aparece un error diciendo que el grupo no puede estar asignado en ningún minijuego para ser modificado	C
5	Se modificará un grupo	Se modifica correctamente y se actualiza la tabla	No se modifica.	I
6	Se selecciona un grupo a eliminar	Se elimina el grupo y se actualiza la tabla	Se elimina el grupo y se actualiza la tabla	C

Tabla 10: Pruebas Ventana Grupos

- Prueba 5: El error residía en que se recogía incorrectamente el grupo a modificar. Por esto, se mandaban modificar unas filas no seleccionadas (se modificaban poniendo los mismos datos). Se solucionó comprobando cada fila de la tabla si tiene pulsado la celda “Modificar”, y en dicho caso, recogiendo y modificando su contenido.

9. Ventana Minijuegos

Se verificará la inserción, modificación y eliminación del minijuego.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Se seleccionará un minijuego sin grupos compatibles	Aparece un error diciendo que no existen grupos compatibles con el minijuego	Aparece un error diciendo que no existen grupos compatibles con el minijuego	C
2	Se seleccionará un minijuego sin necesidad de grupos	Aparece un mensaje de que ese minijuego no necesita ningún grupo	Aparece un error diciendo que no existen grupos compatibles con el minijuego	I
4	Se insertará un grupo en el minijuego	Se inserta el grupo y se actualiza la tabla	Se inserta el grupo y se actualiza la tabla	C
5	Se eliminará un grupo del minijuego	Se elimina el grupo y se actualiza la tabla	Se elimina el grupo y se actualiza la tabla	C
6	Se modificará la información del minijuego	Se modifica la información y se actualiza	Se modifica la información y se actualiza	C

Tabla 11: Pruebas Ventana Grupos

- Prueba 2: El error residía en que tanto si no existía grupos, como si el minijuego no necesitaba grupos, se inicializaba la lista de los grupos. De esta forma, a la hora de comprobar si necesitaba grupos el minijuego (si es *null* la lista), nunca iba a darse el caso. Para solucionarlo, a la hora de obtener los grupos, en caso de ser un minijuego que no necesita ningún grupo, no se inicializará la lista y se devolverá *null*.

10. Ventana Permisos

Se verificará la asignación y eliminación de permisos.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Se asignará permisos de un minijuego	Se asigna los permisos y se actualiza la tabla	Se asigna los permisos y se actualiza la tabla	C
2	Se asignará permisos de un grupo	Se asigna los permisos y se actualiza la tabla	Se asigna los permisos y se actualiza la tabla	C
4	Se eliminarán permisos de un minijuego	Se eliminan los permisos y se actualiza la tabla	Se eliminan los permisos y se actualiza la tabla	C
5	Se eliminarán permisos de un grupo	Se eliminan los permisos y se actualiza la tabla	Se eliminan los permisos y se actualiza la tabla	C

Tabla 12: Pruebas Ventana Permisos

11. Trigger

Se verificará que los *triggers* de base de datos se activan correctamente y realizan sus funciones.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Insertar datos en aquellas tablas con *triggers*	Se actualiza la tabla correspondiente (de TS normalmente).	No ocurre nada.	I
2	Modificar datos en aquellas tablas con *triggers*	Se actualiza la tabla correspondiente (de TS normalmente)	Se actualiza la tabla correspondiente (de TS normalmente)	C
3	Eliminar datos en aquellas tablas con *triggers*	Se actualiza la tabla correspondiente (de TS normalmente)	Se actualiza la tabla correspondiente (de TS normalmente)	C

Tabla 13: Pruebas Triggers

- Prueba 1: El error residía en que las acciones que se realizaban a la hora de insertar se hacían antes de la inserción (en los *trigger* estaba puesto "BEFORE INSERT"). La solución ha sido la de poner que se realícela acción después de la inserción ("AFTER INSERT").

12. Mapa

Se verificará los botones que funcionan en el mapa son solo los rojos, que se ven los premios conseguidos (y en los que no solo la silueta) y que se puede cambiar de idioma.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Pulsar sobre un botón azul	No ocurre nada	No ocurre nada	C
2	Pulsar sobre un botón rojo	Se cambia al minijuego correspondiente	Se cambia al minijuego correspondiente	C
3	Pulsar sobre un botón gris	No ocurre nada	No ocurre nada	C
4	Ver los Premios	Los premios conseguidos se ven enteros, y los que no sólo la silueta	Todos los premios se ven completos	C

Tabla 14: Pruebas Mapa

13. Sincronización

Se verificará que funciona correctamente toda la sincronización. Para ellos, en cada prueba se creará/modificará/eliminará un dato y se sincronizará la aplicación.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Sincronizarse por primera vez en el videojuego (los datos están correctos)	Los datos se sincroniza y se nos abre la ventana del *Mapa*	Los datos no se sincroniza porque da un error.	I
2	Sincronizarse (no por primera vez), habiendo añadido / modificado / eliminado algún campo	Los datos añadidos / modificados / eliminados se sincronizan correctamente	Los datos añadidos / modificados / eliminados no se sincronizan	I
3	Sincronizarse habiendo modificado contenido	El contenido modificado se sincroniza	El contenido modificado se sincroniza	I
4	Sincronizarse habiendo añadido un grupo	El grupo añadido se sincroniza	El grupo añadido se sincroniza	C
5	Sincronizarse habiendo modificado un grupo	El grupo modificado se sincroniza	El grupo modificado se sincroniza	C
6	Sincronizarse habiendo eliminado un grupo	El grupo eliminado se elimina	El grupo eliminado se elimina	C
7	Sincronizarse habiendo asignado un minijuego	El minijuego asignado se sincroniza y dependiendo del azar, es asignado a un botón	El minijuego asignado se sincroniza y dependiendo del azar, es asignado a un botón	C
8	Sincronizarse habiendo eliminado un minijuego y el minijuego no está entre los minijuegos del mapa actual	El minijuego eliminado se elimina	El minijuego eliminado se elimina	C
9	Sincronizarse habiendo eliminado un minijuego y el minijuego está entre los minijuegos del mapa actual	El minijuego eliminado se elimina y se reasigna el minijuego correspondiente al botón	El minijuego eliminado se elimina y se reasigna el minijuego correspondiente al botón	C
10	Sincronizarse habiendo añadido un mapa	El mapa añadido se sincroniza y dependiendo del azar, se verá entre los mapas	El mapa añadido se sincroniza y dependiendo del azar, se verá entre los mapas	C
11	Sincronizarse habiendo añadido un botón en un mapa	El botón añadido se sincroniza. Además, se le asigna un minijuego	El botón añadido se sincroniza. Además, se le asigna un minijuego	C
12	Sincronizarse habiendo modificado un botón de un mapa	El botón modificado se sincroniza	El botón modificado se sincroniza	C
13	Sincronizarse habiendo eliminado un botón de un mapa	El botón eliminado se elimina del mapa en el videojuego	El botón modificado se sincroniza	I
14	Sincronizarse habiendo añadido un botón en un fondo	El fondo añadido se sincroniza y dependiendo del azar, se verá entre los fondo	El fondo añadido se sincroniza y dependiendo del azar, se verá entre los fondo	C
15	Sincronizarse habiendo añadido un premio	El premio añadido se sincroniza y está como no conseguido (silueta en negro)	El premio añadido se sincroniza y está como no conseguido (silueta en negro)	C
16	Tener el juego en dos dispositivos, pasarse un nivel en un minijuego y sincronizar en el otro	El nivel que nos hemos pasado se verá reflejado en el segundo dispositivo	El nivel que nos hemos pasado se verá reflejado en el segundo dispositivo	C

Tabla 15: Pruebas Sincronización

- Prueba 1: El error que da es debido a que la memoria de base de datos local se llena (*OutOfMemory*). Para solucionar esto, antes de cada consulta, y siempre que no esté dentro de una transacción, se abrirá la conexión. De la misma forma, tras obtener o modificar los datos, se cerrará la conexión (si no está dentro de una transacción).
- Prueba 2: Este problema es debido a los tipos de tiempo que utilizan *SQLite* y *MySQL*. AL ser de diferente tipo, se ha tenido que obtener la fecha de sincronización como un texto y transformarla a fecha en la *SELECT* mediante el atributo "*CONVERT(@tiempo,DATETIME)* ".
- Prueba 13: Este problema es debido a que, a la hora de sincronizar los botones, no se había comprobado si los botones estaban borrados o no. Esto causaba que siempre se modificarían. Para solucionar esto se ha realizado una comprobación que en caso de ser 'B' la acción, se eliminará el botón de su tabla y del historial mapa.

14. Desconectar

Se comprobará si funcionan correctamente las desconexiones del programa y del videojuego.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Cerrando Sesión en el programa	Nos envía a la ventana de *Login*	Nos envía a la ventana de *Login*	C
2	Estando conectado a la base de datos (con internet), pulsando a 'Desconectar' en el videojuego	Se eliminan los datos de la base de datos remota y se vuelve a la ventana de *Login*	Se eliminan los datos de la base de datos remota y se vuelve a la ventana de *Login*	C
3	Estando desconectado a la base de datos (sin internet), pulsando a 'Desconectar' en el videojuego	Desaparece la ventana de "Desconectando…"	No desaparece la ventana de "Desconectando…".	I

Tabla 16: Pruebas Desconectar

- Prueba 3: No se gestionaba que hacer cuando aparecía un error. Se ha solucionado poniendo que cuando exista un error, se cierre la ventana de "Desconectando…".

15. Opciones

Se verificará que al pulsar en el botón Reiniciar del menú de opciones del videojuego se reinicia el juego, y que al pulsar en Abandonar se sale.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Pulsando botón Reiniciar en el menú de juego	Se reinicia el nivel y el tiempo	No ocurre nada.	I
2	Pulsando el botón Abandonar	Se cambia a la ventana del Mapa y el nivel no se completa	Se cambia a la ventana del Mapa y el nivel no se completa	C

Tabla 17: Pruebas Opciones

- Prueba 1: El error residía en un problema de conceptos respecto a la herencia. Al pulsar en el botón, se hacía una llamada al método *Start* de la clase padre (*Minijuego*) en vez a la del hijo. Para solucionarlo, se ha cambiado esto, obteniendo el componente Minijuego del objeto en el que estamos y llamando a su método Start (el objeto será de tipo Minijuego porque todos los minijuegos heredan de este) (Ver Figura 123).

```
this.gameObject.GetComponent<Minijuego> ().Start ();
opcionesOn = false;
```

Figura 123: Reiniciar Nivel

16. Minijuego

Se verificará que se guarda la partida una vez terminada, y que se visualiza un fondo, el título del minijuego y su descripción. Además, se comprobará si, en ocasiones, se asignan premios.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Jugando más de una ocasión	Se ve el título y la descripción	Se ve el título y la descripción	C
2	Terminar una partida (ganando)	Se guarda el juego y se gana un premio. Además, se completa el nivel.	Se guarda el juego y se gana un premio. Además, se completa el nivel.	C
3	Terminar una partida (ganando)	Se guarda el juego y no se gana un premio. Además, se completa el nivel.	Se guarda el juego y no se gana un premio. Además, se completa el nivel.	C
	Terminar una partida (perdiendo)	Se guarda el juego	Se guarda el juego	C

Tabla 18: Pruebas Minijuego

17. Sopa de Letras

Se verificará que las diferentes funcionalidades funcionan.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Jugando más de una ocasión	Se muestran imágenes	Se muestran imágenes	C
2	Jugando más de una ocasión	Se muestran textos	Se muestran textos	C
3	Seleccionando una palabra correcta en la sopa de letras	La imagen o el texto se oscurece y se selecciona en la sopa de letras	La imagen o el texto se oscurece y se selecciona en la sopa de letras	C
4	Seleccionando una palabra incorrecta en la sopa de letras	Se resta una vida y se vuelve a su estado anterior las casillas	Se resta una vida y se desactivan las casillas (se ponen como al principio en vez de su estado anterior)	I

Tabla 19: Pruebas Sopa de Letras

- Prueba 4: El error residía en que una vez se confirmaba que, la palabra era incorrecta, se ponían a todas las casillas el estado inicial que tenían en vez del estado anterior. Para solucionarlo, se ha guardado los estados, y se ha puesto el estado guardado.

18. Buscar

Se verificará que las diferentes funcionalidades funcionan.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Pulsando sobre un elemento correcto	Se pone en verde el elemento	Se pone en verde el elemento	C
2	Pulsando sobre un elemento incorrecto	Se pone en rojo el elemento y se resta una vida	Se pone en rojo el elemento y se resta una vida	C

Tabla 20: Pruebas Buscar

19. Palabras Incompletas

Se verificará que las diferentes funcionalidades funcionan.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Seleccionar un hueco	Aparece el hueco seleccionado y las letras en los laterales	Aparece el hueco seleccionado y las letras en los laterales	C
2	Completar una palabra de forma incorrecta	Se muestra el fallo en rojo y se resta una vida	Se muestra el fallo en rojo y se resta una vida	C

Tabla 21: Pruebas Palabras Incompletas

20. Laberinto Erróneo

Se verificará que se crea correctamente el camino correcto y que las casillas que se muestran son sólo aquellas que delimitan con una casilla pulsado.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Jugando más de una ocasión	Se puede completar el minijuego siguiendo el camino	Se llega a un punto, donde no se puede continuar porque no existe más camino	I
2	Jugando más de una ocasión	Se muestra sólo aquellas casillas que delimitan con una casilla pulsada.	En ocasiones, se muestra una casilla al inicio (aparte de la casa)	I
3	Pulsar una casilla correcta	Se muestran sus casillas de alrededor.	Se muestran sus casillas de alrededor.	C
4	Pulsar una casilla incorrecta	Se resta una vida	Se resta una vida	C
5	Pulsar una casilla invisible	No ocurre nada	No ocurre nada	C

Tabla 22: Pruebas Laberinto Erróneo

- Prueba 1: El problema residía en la creación del camino. No se rehacía en caso de que no hubiera opciones. Se ha puesto este caso.
- Prueba 2: El problema residía en que, en caso de no crear correctamente un camino a la primera, el camino se reiniciaba, pero la matriz donde se guardaba si cada casilla estaba visible o no, no. La solución ha sido reinicializar esta matriz cuando pase eso.

21. Frase Correcta

Se verificará que las diferentes funcionalidades funcionan.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Seleccionar alguna opción incorrecta	Se muestra la opción incorrecta en rojo y se resta una vida	Se muestra la opción incorrecta en rojo y se resta una vida	C
2	Seleccionar todas las opciones correctas	Se gana la partida	Se gana la partida	C

Tabla 23: Pruebas Frase Correcta

22. Seleccionar

Se verificará que las diferentes funcionalidades funcionan.

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Seleccionar alguna opción incorrecta	Se muestra la opción incorrecta en rojo y se resta una vida	Se muestra la opción incorrecta en rojo y se resta una vida	C
2	Seleccionar todas las opciones correctas	Se gana la partida	Se gana la partida	C
3	Utilizar el scroll.	Se cambia a la siguiente o anterior fase.	Se queda la vista entre dos fases.	I

Tabla 24: Pruebas Seleccionar

Prueba 3: El error residía en que, a la hora de cambiar entre fases, se sumaba un valor incorrecto a la posición del scroll. Para solucionarlo, se ha calculado y puesto el valor correcto.

23. Parte Subjetiva

Debido a que el videojuego tiene que estar dirigido a un público de baja edad, el videojuego tiene que ser fácil de usar, divertido y de una alta jugabilidad (que se puede jugar a más de una cosa dentro de él). Debido a esto, se ha pedido a 5 personas sin conocimientos de informática que lo probarán, y se les ha hecho una encuesta sobre estos tres puntos, teniéndolos que valorar del 1 al 5. Una vez obtenido los resultados, se ha hecho la media (la cuál es la que se va a visualizar en la

Id	¿Cómo se Prueba?	Resultado esperado	Resultado obtenido	
1	Fácil de Usar	Media mayor a 2.5	Media de 4.	C
2	Divertido	Media mayor a 2.5	Media de 5.	C
3	Jugabilidad	Media mayor a 2.5	Media de 4.5	C

Figura 124: Pruebas Subjetivas

Conclusiones y Trabajo Futuro

Tras finalizar el proyecto, llega la hora de echar la vista atrás. Es hora de realizar un análisis crítico tanto del proyecto en general, como de carácter personal y de líneas futuras.

1. Conclusiones Generales

Debido a que lo primero que si hizo en el proyecto fue el análisis del alcance que tendría, se comenzará por valorar si las expectativas previstas en un comienzo se han conseguido. Para ello, obtendremos de nuevo la tabla que creamos en su momento, pero esta vez, sólo utilizaremos los indicadores generales. Además, compararemos dichos indicadores con los finales (Ver Tabla 25).

Tarea	Horas dedicadas (Estimación)	Horas dedicadas (Reales)
Planificación	33 horas	47h
Análisis	40 horas	55h
Diseño Técnico	65 horas	85h
Diseño Gráfico	20 horas	5h
Implementación	378 horas	550h
Pruebas	36 horas	50h
Documentación	216 horas	220h
TIEMPO TOTAL	788 horas	1012h

Tabla 25: Horas reales

Para valorar mejor los tiempos, se ha creado una gráfica (Ver Figura 125). Como se puede observar, en la mayoría de indicadores los tiempos son similares. El indicador en el que se nota una diferencia abismal (224 horas) es en el de Implementación. Esto ha sido porque en ciertas partes del proyecto se vieron dificultades para realizar las pantallas. Un ejemplo de esto pueden ser las tablas de las ventanas, la sopa de letras del videojuego o la sincronización del mismo. Por esto, las horas no se ajustaron a lo estimado. Otro dato a comentar es que el único indicador que su estimación ha sido mayor a la real es en el apartado del diseño gráfico. Esto ha sido porque al no haber realizado ni figuras en 3D ni los fondos o mapas, el tiempo en este apartado se vio reducido. Con todo esto, la diferencia de horas totales entre las estimadas y las totales es de 224 horas.

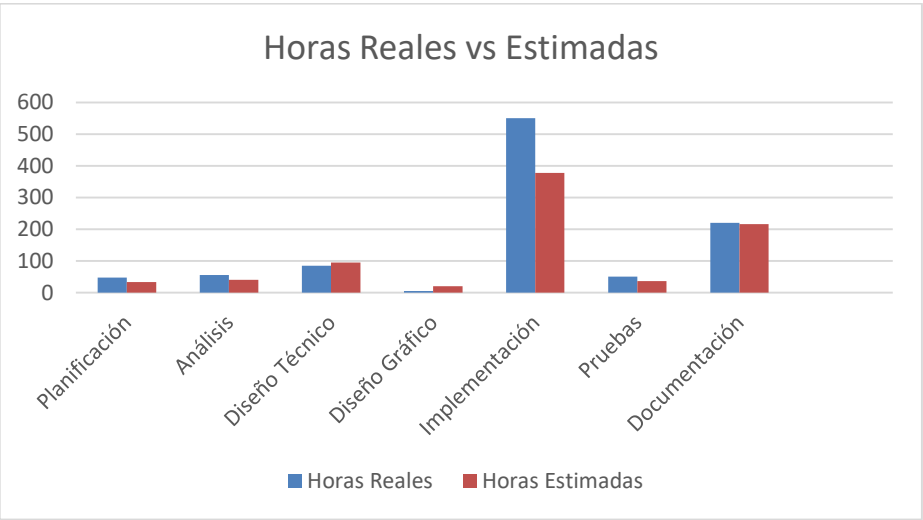


Figura 125: Horas Reales vs Estimadas

Por otro lado, nos encontraríamos con los objetivos principales. Se podría decir que todos los objetivos han sido completados satisfactoriamente. Por otro lado, el objetivo de realizar una aplicación móvil intuitiva y divertida se ha valorado mediante una encuesta a cinco personas diferentes. Estas personas, sin conocimientos en informática, han probado la aplicación y han respondido a tres preguntas para comprobar este objetivo, dando un resultado positivo.

Por último, nos encontramos con la valoración de la utilidad de la gestión de riesgos utilizada. En nuestro caso, nos han surgido varios riesgos de los planteados a lo largo del proyecto, pero gracias a su plan de contingencia, se han solucionado. Uno de estos riegos ha sido las actualizaciones. A mitad del proyecto apareció una nueva actualización del programa *Unity3D*. Esta actualización se descartó (como se planteó en un principio) hasta haber realizado completamente el proyecto. Otro problema ha sido la planificación incorrecta. Debido a ser la primera planificación, como ya se ha visto al comienzo de estas conclusiones, el tiempo empleado ha sido muy diferente. Por esto, según se acercaba la fecha de entrega, se han ido aumentando el número de horas que se empleaban para poder cumplir las fechas.

2. Conclusiones Personales

En cuanto al tema personal, se podría comentar que se ha aprendido a gestionar un proyecto, desde su primera fase de análisis de datos, hasta la parte final de pruebas. A pesar de que en el transcurso del mismo se han encontrado diversos problemas (por ejemplo, el tema de dibujar en 3D) que han creado la necesidad de cambiar el proyecto, se podría valorar la gestión del mismo positivamente.

Por otro lado, se han aprendido la utilización de diversas librerías para realizar las mismas actividades que se han dado a lo largo de la carrera, pero de otra forma. Una de estas librerías es *MyBatis*. Además, se ha comprobado de que a pesar de que se haga una estimación de un proyecto, esta puede acabar siendo totalmente errónea debido a diversos factores.

Por último, se ha aprendido a gestionar una base de datos desde cero. Para esto, se han tenido que utilizar todo el conocimiento aprendido durante la carrera para gestionar las bases de datos lo mejor posible. Aun así, debido a las limitaciones de la base de datos (que no somos administradores de la misma), nos hemos quedado con las ganas de una mayor administración de la misma (como los permisos de usuario y la administración de los bloqueos). Además, a falta de tiempo, se ha dejado alguna cosa sin terminar de pulir, las cuales serán comentadas en las líneas futuras.

3. Conclusiones sobre Líneas Futuras

A pesar de haber terminado la elaboración del proyecto, no se descarta el seguir trabajando en el mismo. Como se ha mencionado anteriormente, esta aplicación y programa pueden ser mejorados, pero por falta de tiempo, no se ha podido hacer todavía. Por esto, a continuación, se explicarán las diferentes mejoras pensadas, y entre paréntesis se pondrá una división correspondiente a la importancia de la mejora entre el coste de tiempo que conllevaría (ambos valores de 1 a 10).

En cuanto al programa, se podría mejorar la interfaz gráfica (2/5). La que tiene actualmente es la correspondiente a los componentes de *Java*. Además, se podría gestionar la obtención de los datos, ya que, en caso de disponer de muchos, el tiempo que se demora puede ser muy superior al esperado. Esto se podría hacer guardando internamente la información de cada tabla de base de datos, o mejorando la arquitectura de las diferentes sentencias que se han realizado (10/6). Por otro lado, se podría añadir la posibilidad de eliminar los mapas, fondos y premios creados (Esto no se ha realizado debido a que complicaba la sincronización) (6/3). Esta última opción se realizaría siempre y cuando se hubiera mejorado la sincronización del videojuego, lo cual, se comentará a continuación.

En lo referente al videojuego, como se ha mencionado en el párrafo anterior, una de las mejoras necesarias es el tema de la sincronización (10/6). Hoy por hoy, en caso de existir muchas imágenes, la obtención de estas se demora mucho. Por todo esto, habría que mejorar este aspecto de alguna forma (como no eliminando el contenido tras desconectarse, y al sincronizar un contenido modificado, verificar si la imagen ha variado antes de descargarla). Por otro lado, se podría mejorar la interfaz del videojuego, sobre todo, en la parte del mapa (2/5). Un añadido en esta parte podría ser la visualización de las llaves conseguidas en cada nivel completado, o la visualización de qué minijuego se trata al pulsar en los botones del mapa (antes de entrar en el propio videojuego) (3/3). También existe la posibilidad de aumentar la cantidad de minijuegos, añadiendo aquellos que se descartaron debido a la complejidad que tenían (5/5). Como última, se podría plantear la posibilidad de meter música en el videojuego, y sonidos en los juegos (8/6). Estos sonidos serán similares a las imágenes, los tendrán que introducir la logopeda mediante el programa (grabaciones de pronunciación, por ejemplo).

Por último, se podría mejorar la base de datos. Para esto, se podría trabajar como administrador de bases de datos, y crear nuevos usuarios y gestionar los permisos de los mismos (5/2). De esta forma, se evitaría la obtención de datos de carácter confidencial (en caso de que se sepa la contraseña). Por otro lado, se gestionarían el tema de los bloqueos para agilizar la obtención de datos de la base de datos. Además, se protegería la contraseña de la conexión de la base de datos para evitar la obtención de la misma en caso de conseguir el código fuente (por ejemplo, guardándola en un PHP del servidor).

Además de todo esto, se podría gestionar el tema de *marketing* de la aplicación y del videojuego. Con esto se quiere decir que no sólo podría utilizar ambos programas el gabinete de logopeda *Hitz-Kiribil*, sino que podría estar en manos de aquellas empresas o gabinetes que pagaran por ella. Para esto, se tendría que gestionar de una forma más adecuada la base de datos para solo poder visualizar lo correspondiente a tu empresa (4/3). En relación con esto, se ha pensado la monetización del videojuego con la introducción de anuncios a través de *Google (Google Ads*), pero sabiendo que está dedicada a su utilización por niños, esto se ha descartado por los diferentes problemas que podría acarrear (por ejemplo, que un niño pulse en un anuncio y compre algo).

Todas estas mejoras llevarían una gran cantidad de tiempo. Todas ellas son opcionales, pero puede que la referente a la sincronización es la única que podría acercarse más a ser obligatoria, porque es necesaria la reducción del tiempo de la misma (es la que mayor valor en importancia/coste se le ha dado).

Bibliografía

1. Libros

Gregory, J. (2014). Game Engine Architecture. En *Game Engine Architecture* (pág. 1014). CRC Press.

2. Referencias Web

Bleszinski, C. (23 de Febrero de 2010). *IGN*. Obtenido de http://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine?page=1

Brodkin, J. (3 de Junio de 2013). *Dice*. (Dice) Obtenido de http://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/

Consejería de Educación del Principado de Asturias. (s.f.). *Logopedia Escolar en Asturias.* Obtenido de Logopedia Escolar en Asturias: http://web.educastur.princast.es/proyectos/lea/uploads/file/TRASTORNO%20DEL%20LENGUAJE/EVALUACION/ANAMNESIS.pdf

Cuanto Gana. (s.f.). *Cuanto Gana*. Obtenido de Cuanto Gana: https://cuantogana.xyz/un-programador/

Epic Games. (2004). *Unreal Engine*. (Epic Games) Obtenido de https://www.unrealengine.com/unreal-engine-4

Europa Press. (16 de Mayo de 2013). *Europa Press*. Obtenido de Europa Press: http://www.europapress.es/portaltic/software/noticia-google-lanza-android-studio-nuevo-entorno-programacion-android-20130516142704.html

Fraga, A. I. (6 de Febrero de 2016). *TICbeat*. Obtenido de http://www.ticbeat.com/cyborgcultura/videojuegos-la-industria-multimillonaria-detras-del-mayor-entretenimiento-del-planeta/

Gestionet. (2015). *Jugar es Serio: Aplicaciones para alumnos con necesidades educativas especiales.* Obtenido de Jugar es Serio: Aplicaciones para alumnos con necesidades educativas especiales: https://jugaresserio.wordpress.com/2015/03/09/aplicaciones-para-alumnos-con-necesidades-educativas-especiales/

Gestionet. (2016). *Jugar es Serio: Cuadernos de vacaciones 2.0.* Obtenido de Jugar es Serio: Cuadernos de vacaciones 2.0: https://jugaresserio.wordpress.com/2016/07/29/cuadernos-de-vacaciones-2-0/

Google. (2016). *Developer Android*. Obtenido de Developer Android: https://developer.android.com/studio/intro/index.html?hl=es-419

Joan Morales Moras, G. S. (Noviembre de 2016). *Artediez*. Obtenido de Artediez: http://artediez.es/paperback/wp-content/uploads/sites/13/2016/12/arti%CC%81culo-games-definitivo.pdf

Maven. (2017). *MyBatis*. Obtenido de http://www.mybatis.org/mybatis-3/es/

Sánchez Gómez, M. (2008). *Buenas Prácticas en la Creación de Serious Games (Objetos de Aprendizaje Reutilizables).* Obtenido de Buenas Prácticas en la Creación de Serious Games (Objetos de Aprendizaje Reutilizables): http://ceur-ws.org/Vol-318/Sanchez.pdf

Statista, Inc. (2016). *Statista*. Obtenido de Statista: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

TodoTech. (s.f.). *TodoTech*. Obtenido de TodoTech: https://www.todotech.com/android/apps/juegos-mas-populares-android_t161.html

Unity Technologies. (2001). *Unity 3D*. (Unity Technologies) Obtenido de https://unity3d.com/es/unity/editor

Wonnova. (2016). *Wonnova*. Obtenido de Wonnova: http://www.wonnova.com/blog/diferenciar-gamificacion-serious-games-201402

ANEXO I

Casos de Uso Extendidos

1. Registrarse

Nombre: Registrarse

Descripción: Permite registrase en el sistema como Logopeda., de tal forma que pueda acceder a sus funcionalidades. Una vez registrado, se dirigirá a la identificación.

Actores: Usuario Invitado.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. El usuario rellena todos los campos (Ver Figura 126).
[Si se rellenan todos los campos]
    [Si el usuario no existe]
        2.a. Se crea el usuario logopeda y se cambia a la ventana de *login*.
    [si no]
        2.b. Aparece un error de usuario (Ver Figura 127).
    [Fin si]
[si no]
    2.c. Aparece un error de campos obligatorios (Ver Figura 128).
[Fin si]

Poscondiciones: Se creará un usuario Logopeda.

Figura 126: Casos de Uso Registro

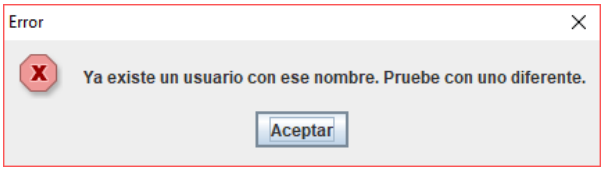


Figura 127: Casos de Uso Registro Error nombre

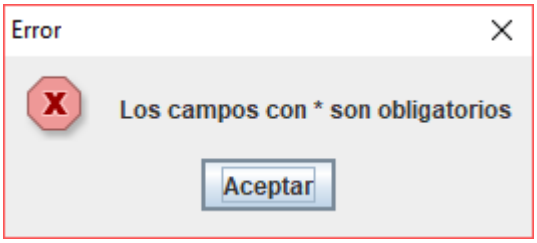


Figura 128: Casos de Uso Registro Campos Obligatorios

2. Identificarse

Nombre: Identificarse.

Descripción: Permite identificarse en el programa (como Logopeda. o Tutor) y en el videojuego para poder acceder a las funcionalidades de su actor.

Actores: Usuario Invitado.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. El usuario introduce su usuario y contraseña y pulsa el botón (Ver Figura 129).
[Si los datos son correctos]
    2.a. Entra en la ventana de menú (Ver Figura 131 o Figura 131).
[Si no]
    2.b. Aparece un error de autentificación (Ver Figura 132).
[Fin si]

Poscondiciones: Se accederá al sistema.

Figura 129: Casos de Uso Login Gestor

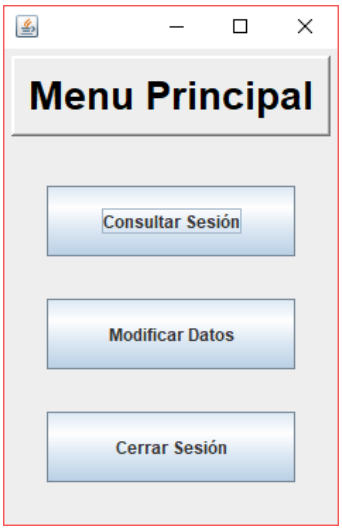


Figura 130: Casos de Uso Menú Paciente

Figura 131: Casos de Uso Menú Logopeda

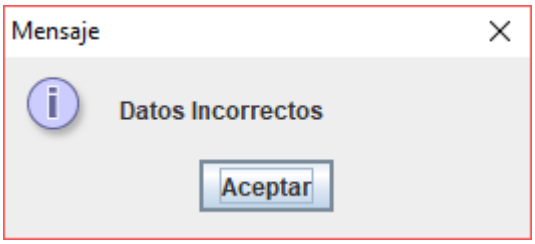


Figura 132: Casos de Uso Login Error datos

3. Recuperar Contraseña

Nombre: Recuperar Contraseña.

Descripción: Permite recuperar la contraseña del usuario.

Actores: Usuario Invitado.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. El usuario introduce el usuario y pulsa el botón recuperar contraseña.
[Si se el usuario introducido existe]
    2.a. Aparece una mensaje diciendo que la contraseña se ha recuperado, y es su nombre (Ver Figura 133).
[si no]
    2.b. Aparece un error de no existe el usuario (Ver Figura 134).
[Fin si]

Poscondiciones: Se modificarán los datos del usuario conectado.

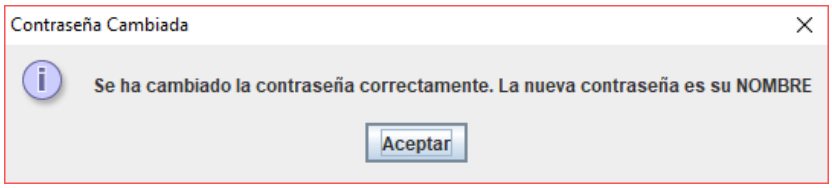


Figura 133: Casos de Uso Recuperar Contraseña

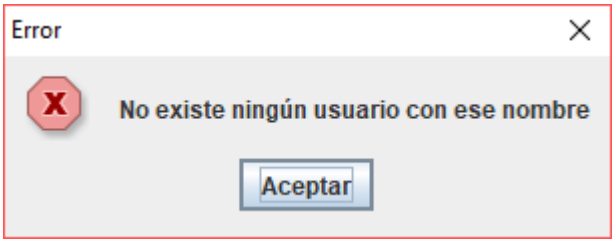


Figura 134: Casos de Uso Recuperar Contraseña error

4. Modificar Datos

Nombre: Modificar Datos.

Descripción: Permite modificar los datos personales.

Actores: Usuario Identificado.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. El usuario modifica los datos (Ver Figura 135 o Figura 136).
[Si se ha introducido la contraseña actual]
    [Si la contraseña es correcta]
        2.a.a. Se modifican los datos y aparece un mensaje (Ver Figura 137)
    [si no]
        2.a.b. Aparece un error de contraseña (Ver Figura 138).
    [Fin si]
[Si no]
    2.b. Aparece un error de campos obligatorios (Ver Figura 139).
[Fin si]

Poscondiciones: Se modificarán los datos del usuario conectado**.**

Figura 135: Casos de Uso Modificar Datos Logopeda

Figura 136: Casos de Uso Modificar Datos Paciente

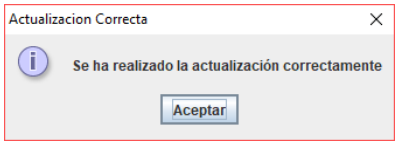


Figura 137: Casos de Uso Modificar Datos correcto

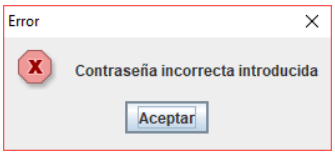


Figura 138: Casos de Uso Modificar Datos Error contraseña

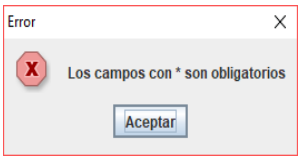


Figura 139: Casos de Uso Modificar Datos Error obligatorio

5. Cerrar Sesión

Nombre: Cerrar Sesión.

Descripción: Permite cerrar la sesión con la que se está conectado.

Actores: Usuario Identificado.

Precondición: Ninguna.

Requisitos no funcionales: Ninguno.

Flujo de eventos:

1. El usuario pulsa sobre Cerrar Sesión.
2. Se vuelve a la ventana de Login.

Poscondiciones: Se cerrará la sesión.

6. Historial Sesión

Nombre: Historial Sesión.

Descripción: Permite acceder a la ventana de la información de los pacientes.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se muestra la ventana del Historial de sesión.
[Si se selecciona algún paciente]
    2.a. EXTENDS Ver Paciente.
[si no]
    2.b. EXTENDS Crear Paciente.
[Fin si]

Poscondiciones: Ninguna.

7. Ver Paciente

Nombre: Ver Paciente.

Descripción: Permite ver la información de un paciente.

Actores: Logopeda.

Precondición: Haber creado algún paciente.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se muestra las ventanas a mostrar.
[Si se selecciona información]
    2.a. Se muestra la información (Ver Figura 140).
[Si se selecciona sesión]
    2.b. Se muestran las sesiones del paciente (Ver Figura 141).
[Si se selecciona Historial Juegos]
    2.c. Se muestran el historial de minijuegos jugados (Ver Figura 142).
[Fin si]

Poscondiciones: Se visualizará el paciente seleccionado.

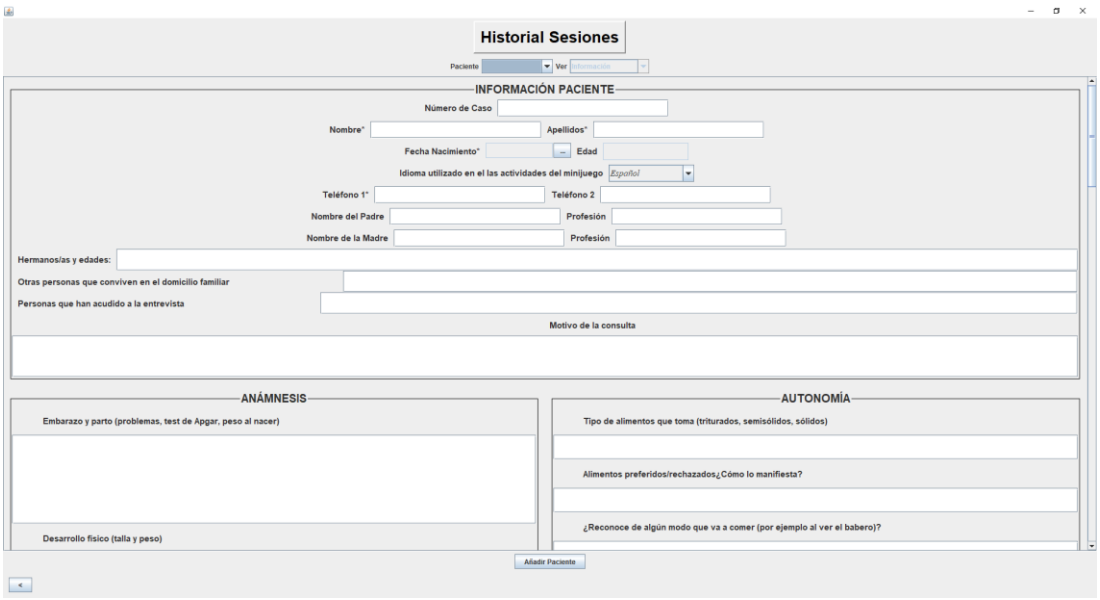


Figura 140: Casos de Uso Historial Sesión Información

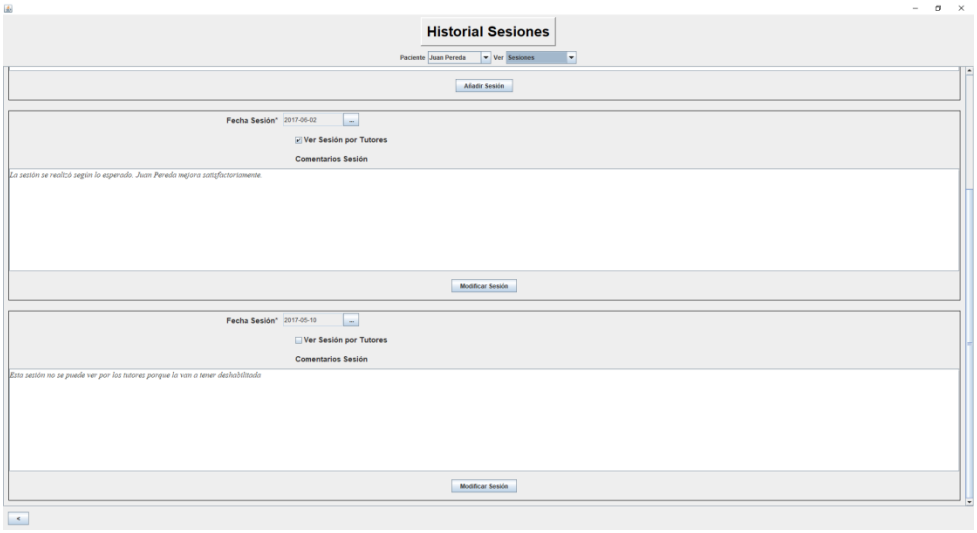


Figura 141: Casos de Uso Historial Sesión Sesiones

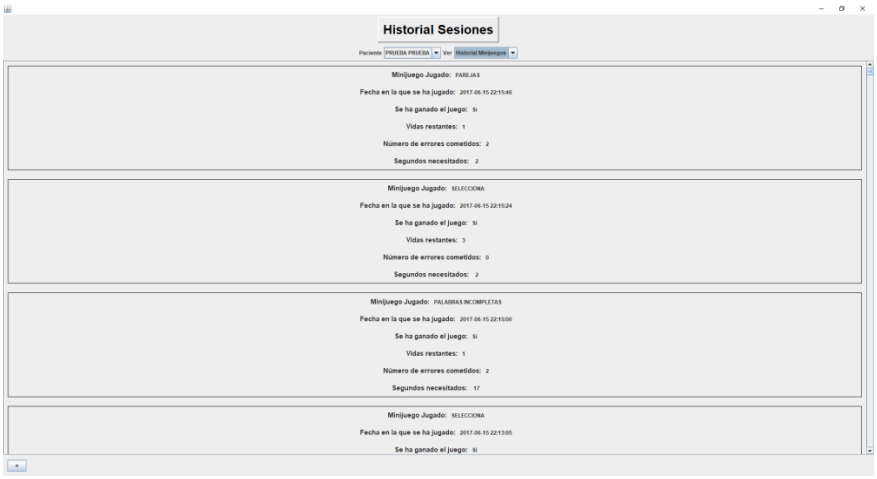


Figura 142: Casos de Uso Historial Sesión Historial Minijuegos

8. Crear Paciente

Nombre: Crear Paciente.

Descripción: Permite crear un nuevo paciente. Para ello tendrá que introducir sus datos.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se muestra una ventana con campos a rellenar (Ver Figura 143).
2. Se rellenan los campos y se pulsa en el botón Crear Paciente.

[Si se rellenan los campos obligatorios]

3.a. El paciente se crea y se muestra su usuario y contraseña (Ver Figura 144).

[si no]

3.b. Aparece un error de campos obligatorios (Ver Figura 145).

[Fin si]

Poscondiciones: Se creará el Paciente.

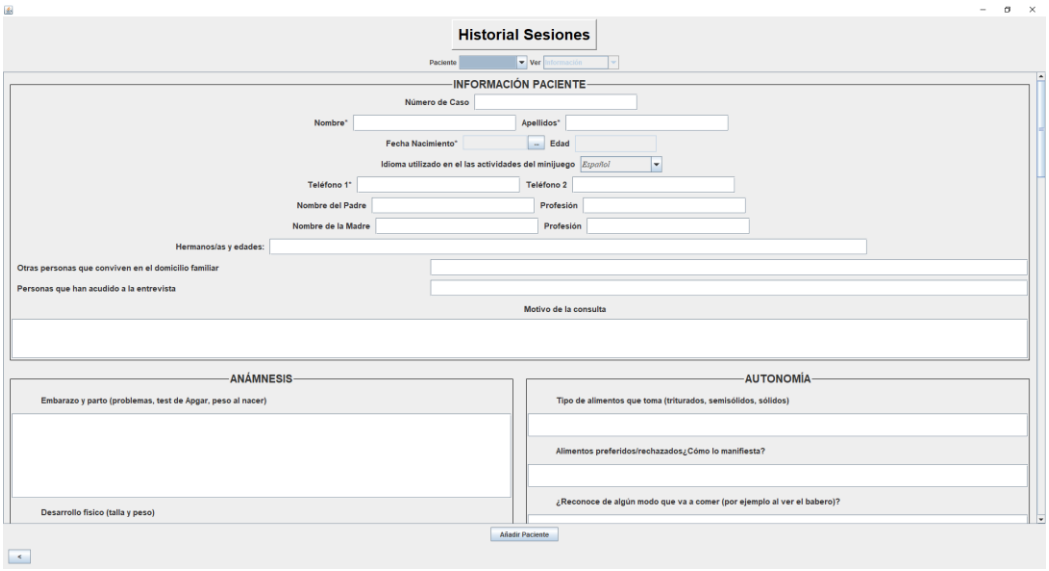


Figura 143: Casos de Uso Historial Sesión Crear Paciente

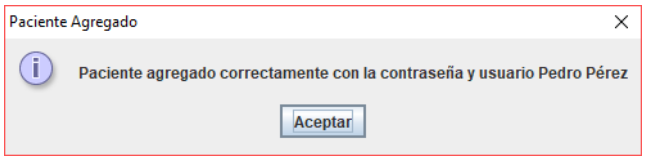


Figura 144: Casos de Uso Historial Sesión creado

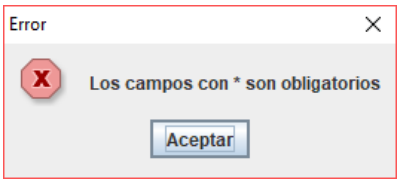


Figura 145: Casos de Uso Historial Sesión Crear Paciente error

9. Crear Sesión

Nombre: Crear Sesión.

Descripción: Permite crear una nueva sesión para un paciente. Para ello se deberá introducir la fecha de la sesión, si la puede ver el tutor, y el comentario.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1.Se visualiza una pantalla con un campo para comentario, un campo de fecha y una casilla (Ver Figura 146).
2.Se rellenan los campos y se pulsa el botón Crear Sesión.
3.Se crea la sesión y se visualiza (Ver Figura 147).

Poscondiciones: Se creará la sesión del paciente.

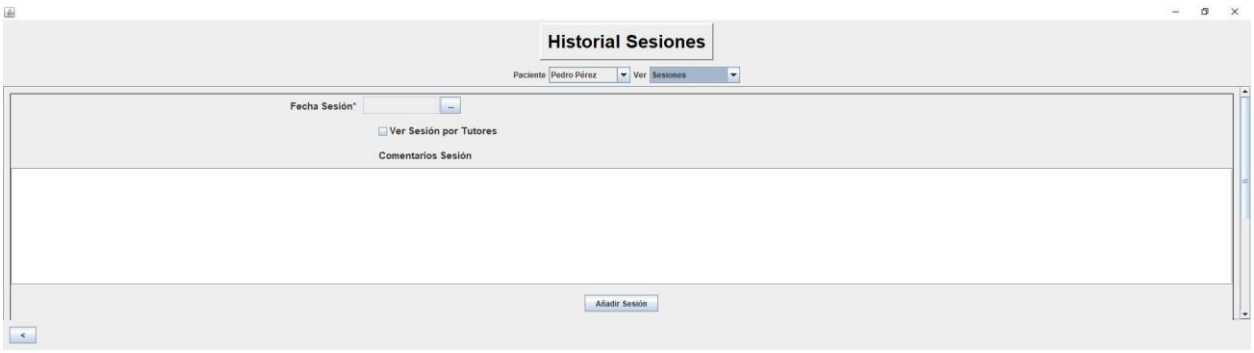


Figura 146: Casos de Uso Historial Sesión Crear Sesión

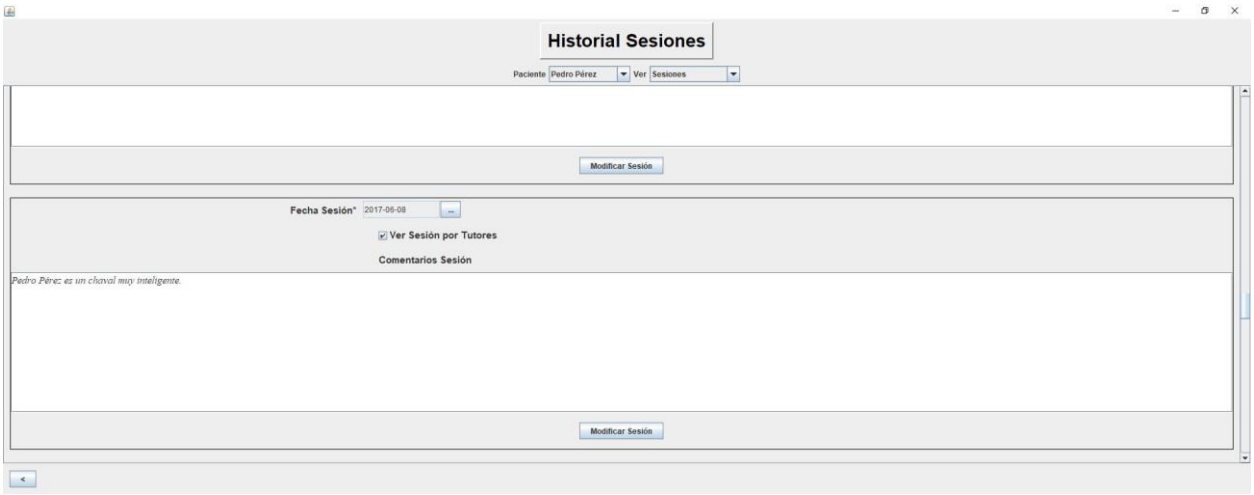


Figura 147: Casos de Uso Historial Sesión Ver Sesión

10. Administrar Juego

Nombre: Administrar Juego.

Descripción: Permite acceder a la ventana de administración general de los juegos.

Actores: Logopeda.

Precondición:

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1.Se visualiza una ventana con diferentes botones.
[SI se pulsa en Fondos]
    2.a. EXTENDS Gestionar Fondos
[SI se pulsa en Mapas]
    2.a. EXTENDS Gestionar Mapas
[SI se pulsa en Premios]
    2.a. EXTENDS Gestionar Premios
[SI se pulsa en Contenidos]
    2.a. EXTENDS Gestionar Contenidos
[SI se pulsa en Grupos]
    2.a. EXTENDS Gestionar Grupos
[SI se pulsa en Minijuegos]
    2.a. EXTENDS Gestionar Minijuegos
[SI se pulsa en Permisos]
    2.a. EXTENDS Gestionar Permisos
[Fin si]

Poscondiciones: Ninguna.

11. Gestionar Fondos

Nombre: Gestionar Fondos.

Descripción: Permite acceder a la ventana donde se gestionan los fondos.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se visualiza una ventana con un selector y un botón (Ver Figura 148).
[Si se selecciona un fondo]
    2.a. Se verá el fondo (Ver Figura 149).
[si no]
    2.b. EXTENDS Añadir Fondo.
[Fin si]

Poscondiciones: Ninguna.

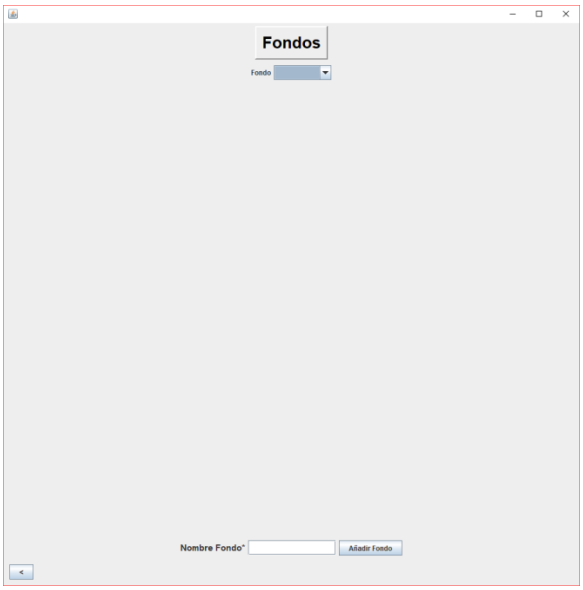


Figura 148: Casos de Uso Fondo

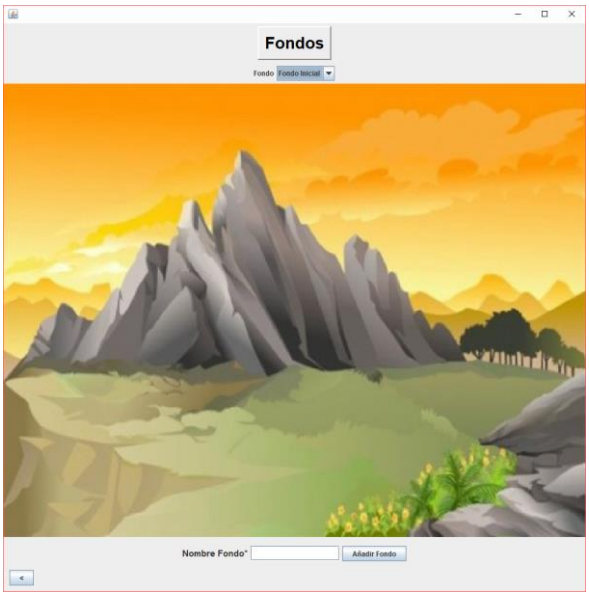


Figura 149: Casos de Uso Con Fondo

12. Añadir Fondo

Nombre: Añadir Fondo.

Descripción: Permite añadir un fondo nuevo. Para ello, se deberá seleccionar una imagen e introducir un nombre de fondo.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se añade un nombre para el fondo y se pulsa el botón de Añadir Fondo.
[Si el nombre no existe]
    2.a. Aparece una ventana para seleccionar la imagen (Ver Figura 150).
    3.a. Se selecciona una imagen y se pulsa en el botón Seleccionar Imagen.
    4.a. Se crea el fondo (Ver Figura 151).
[si no]
    2.b. Aparece un error de que el nombre del fondo ya existe (Ver Figura 152).
[Fin si]

Poscondiciones: Se creará el fondo.

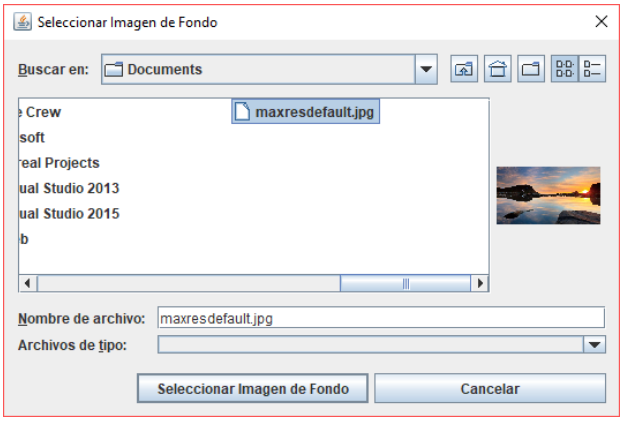


Figura 150: Casos de Uso Fondo Seleccionar

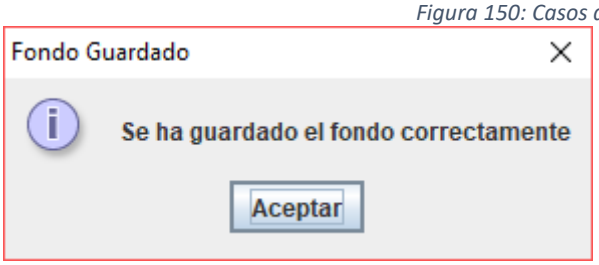


Figura 151: Casos de Uso Fondo Creado

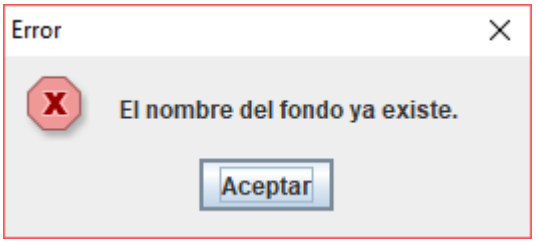


Figura 152: Casos de Uso Fondo Error

13. Gestionar Mapas

Nombre: Gestionar Mapas.

Descripción: Permite acceder a la ventana donde se gestionan los mapas.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1.Se visualiza una ventana con un selector y un botón (Ver Figura 153).
[Si se selecciona un fondo]
    2.a. Se verá el mapa (Ver Figura 154).
    3.a. EXTENDS Modificar Mapa.
[si no]
    2.b. EXTENDS Añadir Mapa.
[Fin si]

Poscondiciones: Ninguna.

Figura 153: Casos de Uso Mapas

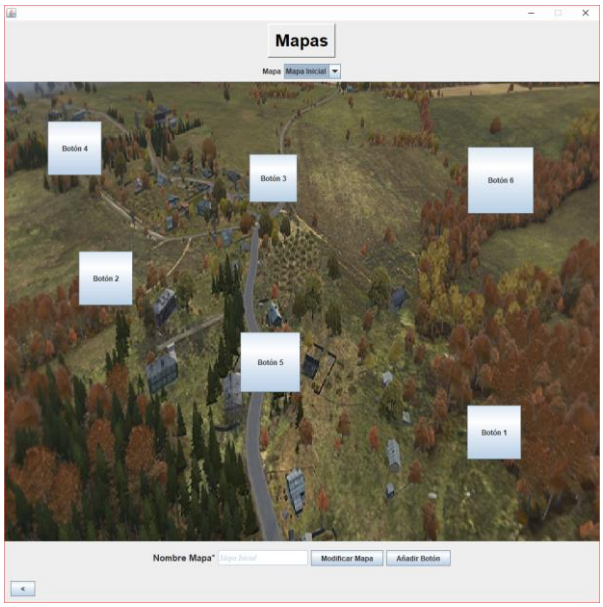


Figura 154: Casos de Uso Con Mapas

14. Añadir Mapa

Nombre: Añadir Mapas.

Descripción: Permite añadir un mapa nuevo y sus botones. Para ello, se deberá seleccionar una imagen e introducir un nombre de mapa.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se añade un nombre para el premio y se pulsa el botón de Añadir Mapa.

[Si el nombre no existe]
2.a. Aparece una ventana para seleccionar la imagen (Ver Figura 155).
3.a. Se selecciona una imagen y se pulsa en el botón Seleccionar Imagen.
4.a. Aparece un nuevo botón para crear un botón.
[Si se pulsa en crear botón]
5.a.a. Se crea un botón.
[Fin si]
[Si se pulsa en un botón con la tecla Control Pulsada]
6.a.a. Se hace el botón más grande o más pequeño.
[Fin si]
[Si se hace click derecho en un botón]
7.a.a. Aparece un mensaje para eliminar el botón (Ver Figura 156).
[Si se pulsa Aceptar]
8.a.a.a. Se elimina el botón.
[Fin si]
[Fin si]
[Si se pulsa en el botón Añadir Mapa]
[Si los botones no se solapan]
8.a.a. Se crea el mapa y sus botones (Ver Figura 158).
[si no]

8.a.b. Aparece un error de solapamiento (Ver Figura 157).
[Fin si]
[Fin si]
[si no]
2.b. Aparece un error de que el nombre del mapa ya existe (Ver Figura 159).
[Fin si]

Poscondiciones: Se creará un mapa y sus botones.

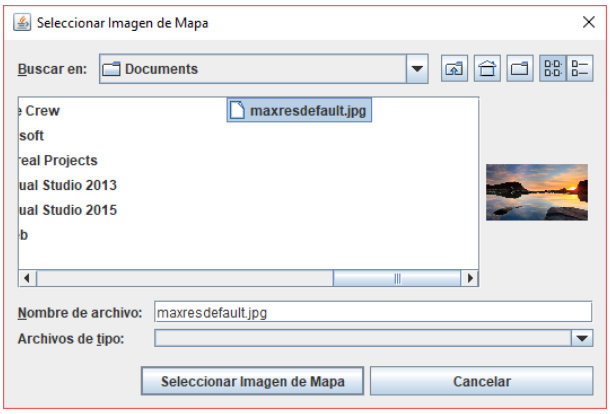


Figura 155: Casos de Uso Mapa Seleccionar

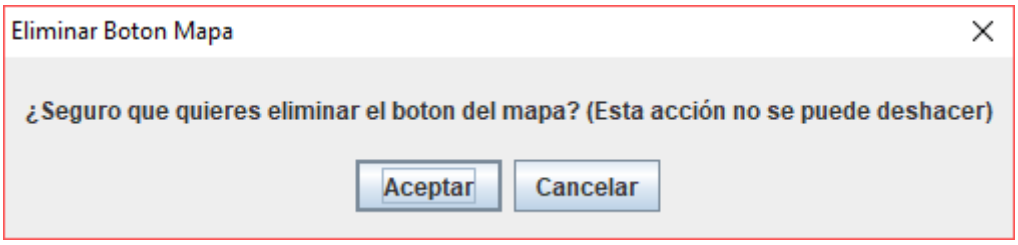


Figura 156: Casos de Uso Mapa Eliminar

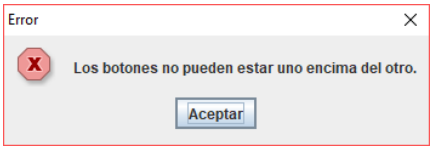


Figura 157: Casos de Uso Mapa Error Solapar

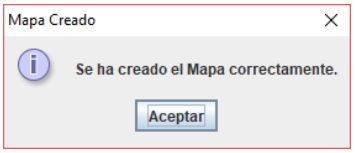


Figura 158: Casos de Uso Mapa Creado

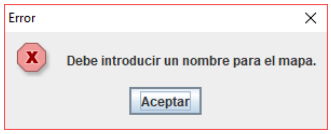


Figura 159: Casos de Uso Mapa Error Nombre

15. Modificar Mapa

Nombre: Modificar Mapa.

Descripción: Permite modificar los botones de un mapa. Se podrá modificar la posición y tamaño de los botones, añadir nuevos y eliminar los existentes.

Actores: Logopeda.

Precondición: Haber seleccionado algún mapa.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se selecciona un mapa y aparece un botón para crear botones, la imagen del mapa y sus botones.
[Si se pulsa en crear botón]
    2.a. Se crea un botón.
[Fin si]
[Si se pulsa en un botón con la tecla Control Pulsada]
        3.a. Se hace el botón más grande o más pequeño.
[Fin si]
[Si se hace click derecho en un botón]
    4.a. Aparece un mensaje para eliminar el botón.
    [Si se pulsa Aceptar]
        5.a.a. Se elimina el botón.
    [Fin si]
[Fin si]
[Si se pulsa en el botón Modificar Mapa]
    [Si los botones no se solapan]
        6.a. Se modifica el mapa y sus botones (Ver Figura 160).
    [si no]
        6.a.b. Aparece un error de solapamiento.
    [Fin si]
[Fin si]

Poscondiciones: Se modificará los botones del mapa seleccionado.

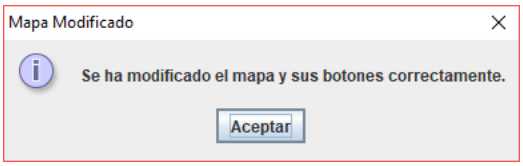


Figura 160: Casos de Uso Mapa Modificado

16. Gestionar Premios

Nombre: Gestionar Premios.

Descripción: Permite acceder a la ventana donde se gestionan los premios.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se visualiza una ventana con un selector y un botón (Ver Figura 161).
[Si se selecciona un fondo]
    2.a. Se verá el premio (Ver Figura 162).
[si no]
    2.b. EXTENDS Añadir Premio.

Poscondiciones: Ninguna.

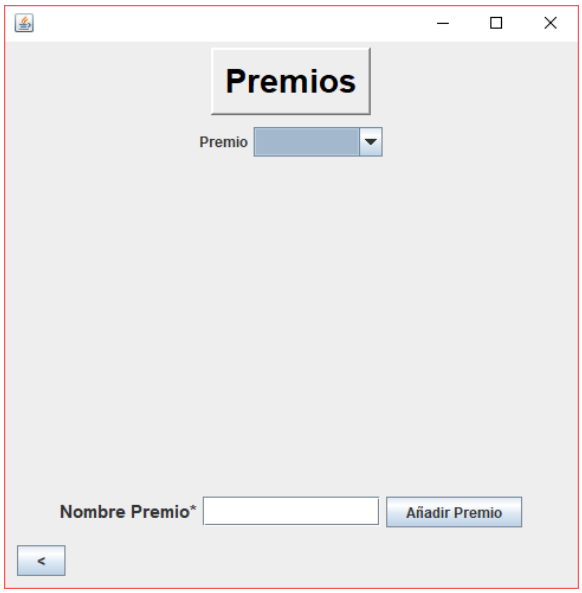


Figura 161: Casos de Uso Premio

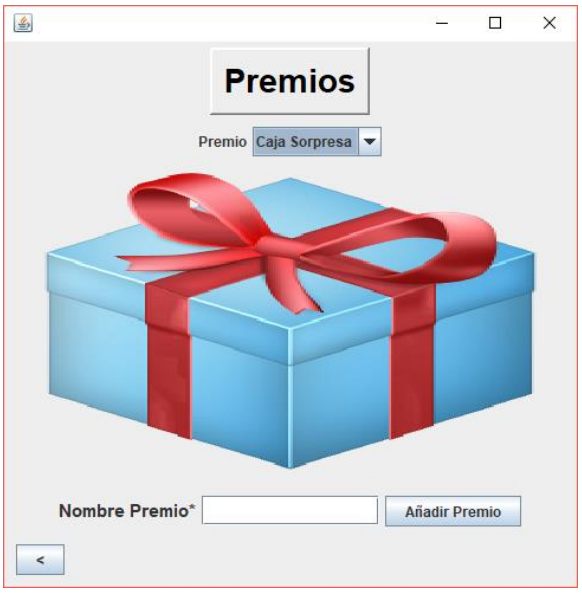


Figura 162: Casos de Uso Con Premio

17. Añadir Premio

Nombre: Añadir Premio.

Descripción: Permite añadir un premio nuevo. Para ello, se deberá seleccionar una imagen e introducir un nombre de premio.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se añade un nombre para el premio y se pulsa el botón de Añadir Premio.

[Si el nombre no existe]

2.a. Aparece una ventana para seleccionar la imagen (Ver Figura 163).

3.a. Se selecciona una imagen y se pulsa en el botón Seleccionar Imagen.

4.a. Se crea el premio (Ver Figura 164).

[si no]

2.b. Aparece un error de que el nombre del premio ya existe (Ver Figura 165).

[Fin si]

Poscondiciones: Se creará el premio.

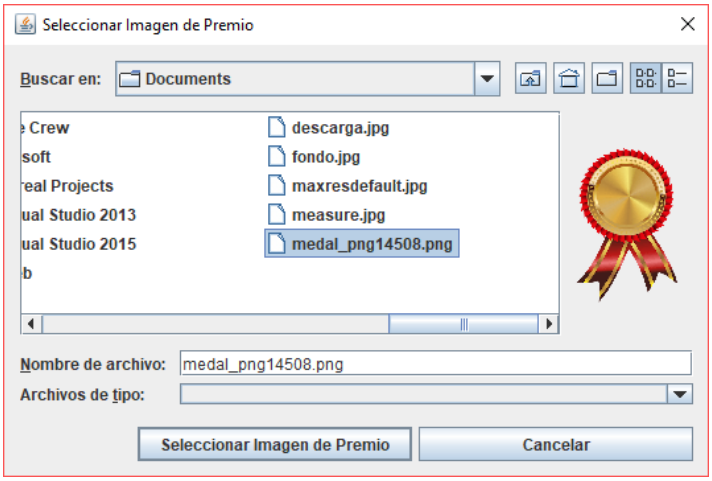


Figura 163: Casos de Uso Premio Seleccionar

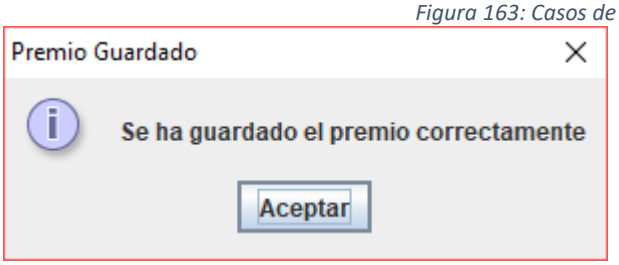


Figura 164: Casos de Uso Premio Creado

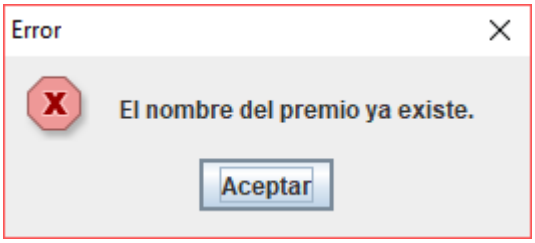


Figura 165: Casos de Uso Premio Error

18. Gestionar Contenido

Nombre: Gestionar Contenido.

Descripción: Permite acceder a la ventana donde se gestionan el contenido.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se visualiza una tabla con todo el contenido y tres botones (Ver Figura 166).
[Si se quiere añadir un contenido]
    2.a. EXTENDS Añadir Contenido
[Si se quiere modificar un contenido]
    2.b. EXTENDS Modificar Contenido
[Si se quiere eliminar un contenido]
    2.c. EXTENDS Eliminar Contenido
[Fin si]

Poscondiciones: Ninguna.

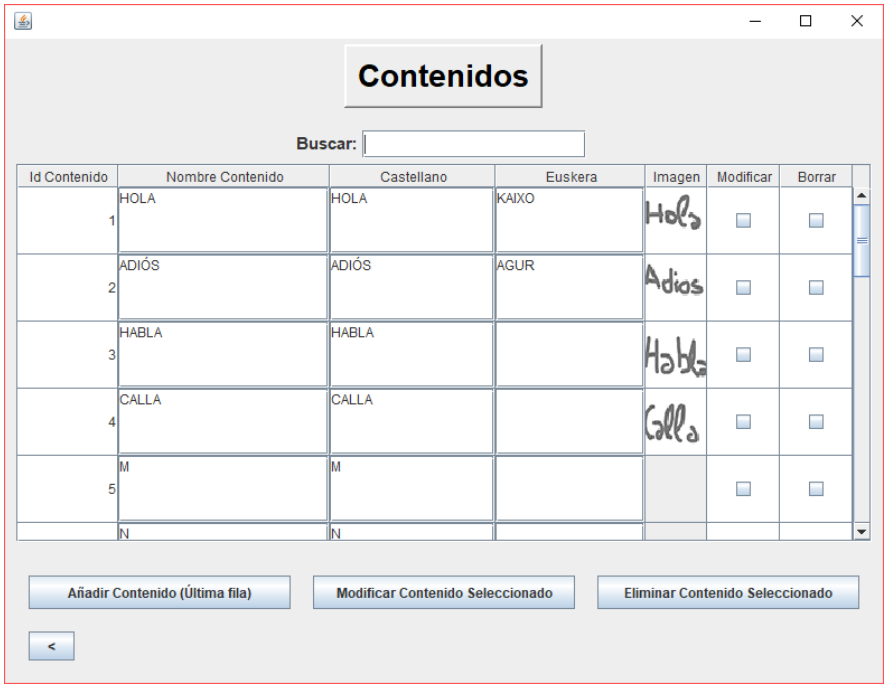


Figura 166: Casos de Uso Contenido

19. Añadir Contenido

Nombre: Añadir Contenido.

Descripción: Permite añadir contenido nuevo. Para ello se tendrá que introducir el nombre del contenido, el texto en castellano o euskera. También se puede introducir una imagen.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se rellenan los campos de la última fila de la tabla y se pulsa en el botón *Crear Contenido*.
[Si se ha introducido un nombre]
    [Si el nombre no existe]
        [Si se ha introducido un valor en el campo castellano o euskera]
            2.a.a.a. Se crea el Contenido (Ver Figura 168).
        [si no]
            2.a.a.b. Aparece un error de campos obligatorios (Ver Figura 167).
        [Fin si]
    [si no]
        2.a.b. Aparece un error de que el nombre del contenido existe (Ver Figura 170).
    [Fin si]
[si no]
        2.b. Aparece un error de nombre obligatorio (Ver Figura 169).
[Fin si]

Poscondiciones: Se creará un contenido.

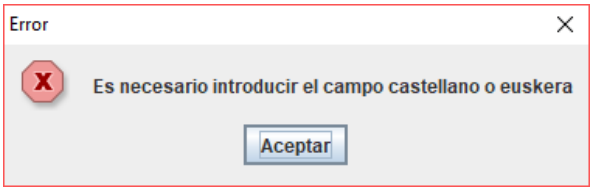


Figura 167: Casos de Uso Contenido Error Obligatorio

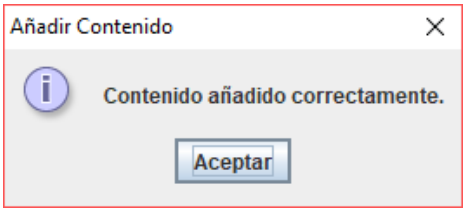


Figura 168: Casos de Uso Contenido Creado

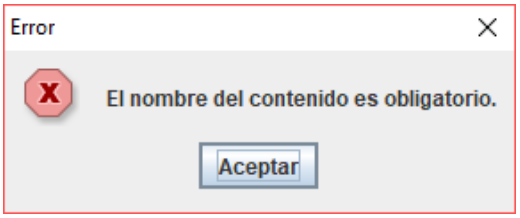


Figura 169: Casos de Uso Contenido Error Nombre

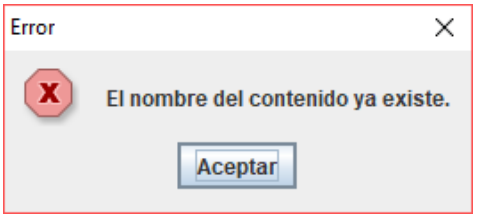


Figura 170: Casos de Uso Contenido Error Nombre Existe

20. Modificar Contenido

Nombre: Modificar Contenido.

Descripción: Permite modificar un contenido creado.

Actores: Logopeda.

Precondición: Haber creado algún contenido.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se modifican los campos de cualquier fila de la tabla, se pulsa sobre la celda modificar y se pulsa en el botón *Modificar Contenido*.
[Si se ha seleccionado alguna celda modificar]
[Si se ha introducido un valor en el campo castellano o euskera en cada celda con modificar]
2.a.a. Se modifica el Contenido (Ver Figura 171).
[si no]
2.a.b. Aparece un error de campos obligatorios.
[Fin si]
[si no]
2.b. Aparece un error de seleccionar alguna celda (Ver Figura 172).
[Fin si]

Poscondiciones: Se modificará el contenido seleccionado.

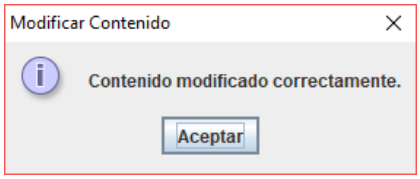


Figura 171: Casos de Uso Contenido Modificado

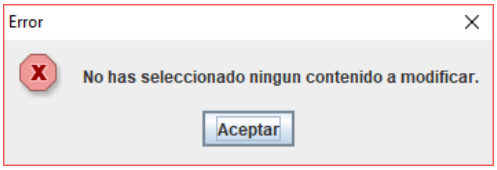


Figura 172: Casos de Uso Contenido Error Seleccionar Modificar

21. Eliminar Contenido

Nombre: Eliminar Contenido.

Descripción: Permite eliminar un contenido creado.

Actores: Logopeda.

Precondición: Haber creado algún contenido.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se pulsa sobre la celda eliminar y se pulsa en el botón *Eliminar Contenido*.
[Si se ha seleccionado alguna celda eliminar]
2.a. Se elimina el Contenido (Ver Figura 173).
[si no]
2.b. Aparece un error de seleccionar alguna celda (Ver Figura 174).
[Fin si]

Poscondiciones: Se eliminará el contenido seleccionado.

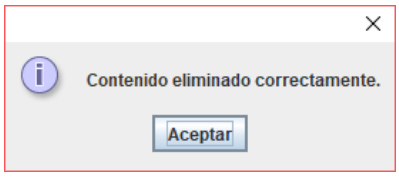


Figura 173: Casos de Uso Contenido Eliminado

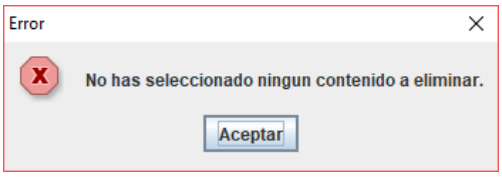


Figura 174: Casos de Uso Contenido Error Seleccionar Eliminar

22. Gestionar Grupos

Nombre: Gestionar Grupos.

Descripción: Permite acceder a la ventana donde se gestionan los grupos.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se visualiza una tabla, un selector y dos botones (Ver Figura 175).
[Si se selecciona algún grupo]
    2.a. Se visualiza el grupo.
    [Si se pulsa sobre el botón *Crear/Modificar Grupo*]
        3.a.a. EXTENDS Añadir/Modificar Grupo.
    [Si se pulsa sobre el botón *Eliminar Grupo*]
        3.a.b. EXTENDS Eliminar Grupo.
    [Fin si]
[si no]
    2.b. EXTENDS Añadir/Modificar Grupo.
[Fin si]

Poscondiciones: Ninguna.

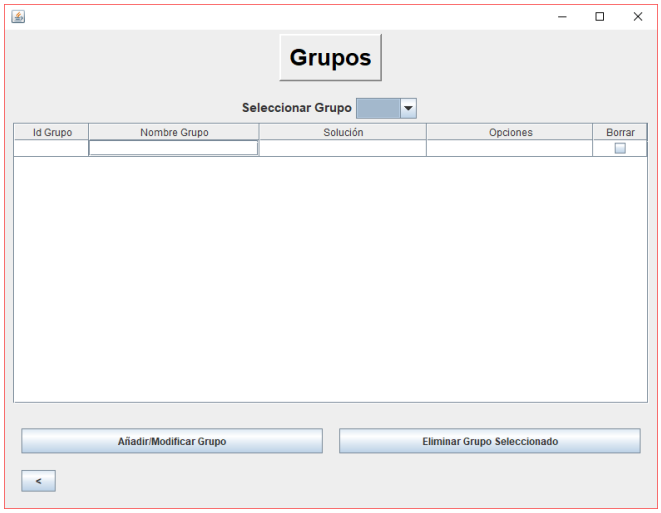


Figura 175: Casos de Uso Grupo

23. Añadir/Modificar Grupo

Nombre: Añadir Grupo.

Descripción: Permite añadir un grupo nuevo o añadir contenido a uno existente. Para ello, se deberá introducir un nombre del grupo, y seleccionar el contenido opción y contenido solución entre todo el contenido existente.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se introduce el nombre del grupo, el contenido solución y el contenido opción, y se pulsa en el botón *Añadir/Modificar Grupo*.
[Si está seleccionado un grupo]
[Si todos los campos están introducidos]
[Si el campo contenido opción no está repetido]
2.a.a.a. Se añade el campo al grupo (Ver Figura 181

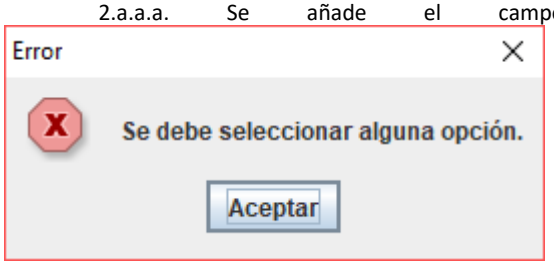


Figura 182).
[si no]
2.a.a.b. Aparece un error de que el contenido opción no puede estar repetido (Ver Figura 176).
[Fin si]
[si no]
2.a.b. Aparece un error de que se debe seleccionar un contenido (Ver Figura 182).
[Fin si]
[si no]
[Si todos los campos están introducidos]
[Si el campo nombre no existe]
2.b.a.a. Se añade el contenido al grupo (Ver Figura 178).
3.b.a.a. Aparece una ventana para asignar el grupo a todos los pacientes (Ver Figura 177).
[Si se pulsa Aceptar]
4.b.a.a.a. Se asigna el grupo a todos los pacientes.
[Fin si]
[si no]
2.b.a.b. Aparece un error de que el nombre del grupo ya existe (Ver Figura 179).
[Fin si]
[si no]
2.b.b. Aparece un error de que se deben rellenar los campos que faltan (Ver o Figura 180 o Figura 182).
[Fin si]
[Fin si]

Poscondiciones: Se creará un grupo o añadirá contenido a uno existente.

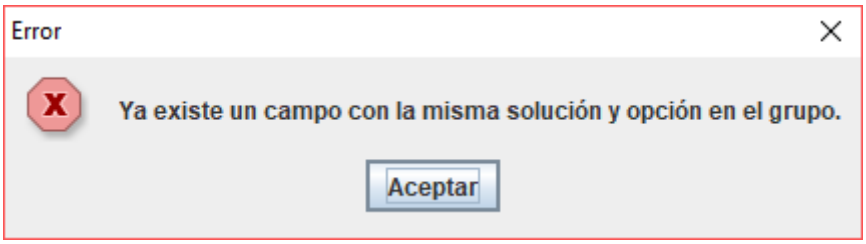


Figura 176: Casos de Uso Grupo Misma Opción

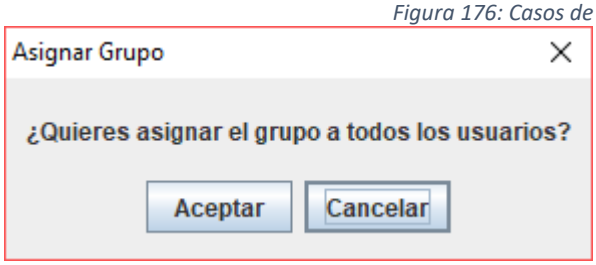


Figura 177: Casos de Uso Grupo Asignar a Todos

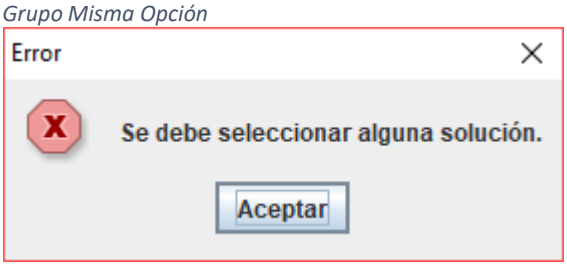


Figura 178: Casos de Uso Grupo Error Solución

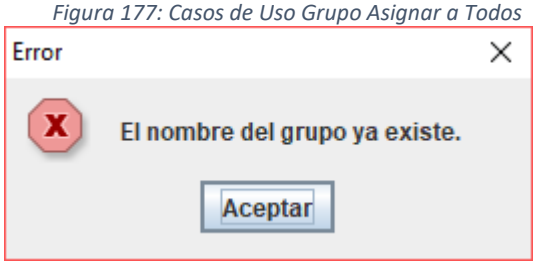


Figura 179: Casos de Uso Grupo Error Nombre Existe

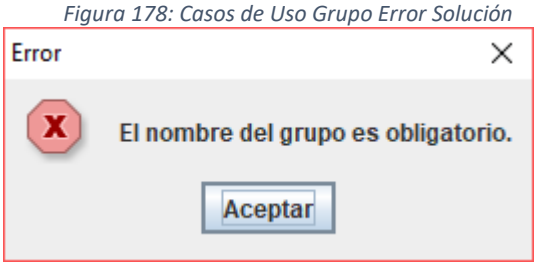


Figura 180: Casos de Uso Grupo Error Nombre

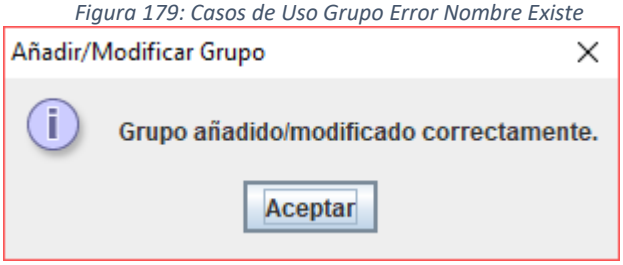


Figura 181: Casos de Uso Grupo Añadido/Modificado

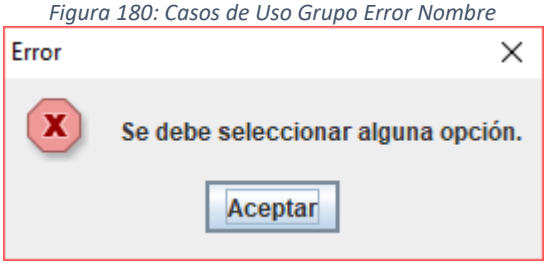


Figura 182: Casos de Uso Error Opción

24. Eliminar Grupo

Nombre: Eliminar Grupo.

Descripción: Permite eliminar un grupo creado.

Actores: Logopeda.

Precondición: Haber creado y seleccionado algún grupo.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se selecciona el contenido del grupo a eliminar y se pulsa el botón *Eliminar Grupo*.
[Si se ha pulsado alguna celda de *Eliminar*]
2.a. Se elimina el grupo (Ver Figura 184).
[Si no]
2.b. Aparece un error de que se tiene que seleccionar alguna fila (Ver Figura 183).
[Fin si]

Poscondiciones: Se eliminará el grupo seleccionado.

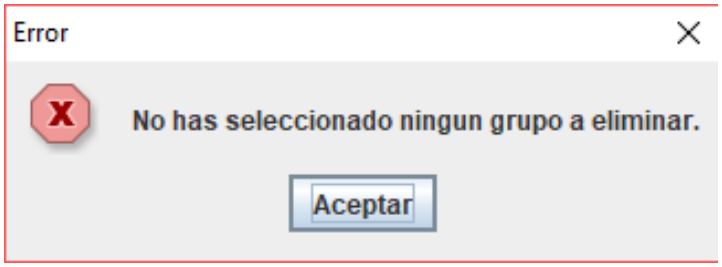


Figura 183: Casos de Uso Grupo Error Eliminar

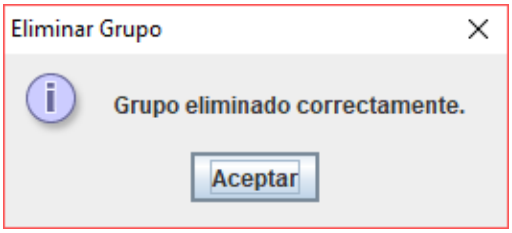


Figura 184: Casos de Uso Eliminado

25. Gestionar Minijuegos

Nombre: Gestionar Minijuegos.

Descripción: Permite acceder a la ventana donde se gestionan los minijuegos.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se visualiza una tabla, un selector y tres botones (Ver Figura 185).
2. Se selecciona algún minijuego y se visualiza sus grupos y su información.
[Si se pulsa sobre el botón *Asignar Grupo a Minijuego*]
3.a. EXTENDS Asignar Grupo a Minijuego.
[Si se pulsa sobre el botón *Eliminar Grupo Seleccionado de Minijuego*]
4.a. EXTENDS Eliminar Grupo de Minijuego.
[Fin si]
[Si se pulsa sobre el botón *Modificar*]
5.a. EXTENDS Modificar Minijuego.
[Fin si]

Poscondiciones: Ninguna.

Figura 185: Casos de Uso Minijuegos

26. Modificar Minijuego

Nombre: Modificar Minijuego.

Descripción: Permite modificar la información y nombre de un minijuego.

Actores: Logopeda.

Precondición: Haber seleccionado algún minijuego.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se cambia la información del minijuego (nombre o descripción) y se pulsa el botón *Modificar*.
[Si ambos campos están rellenados]
    2.a. Se modifica la información del minijuego (Ver Figura 186).
[si no]
    2.b. Aparece un error de datos obligatorios (Ver Figura 187 o Figura 188).
[Fin si]

Poscondiciones: Se modificará el minijuego seleccionado.

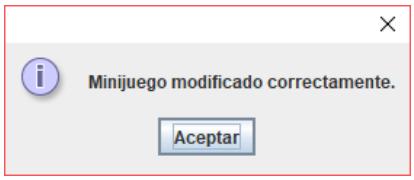


Figura 186: Casos de Uso Minijuego Modificado

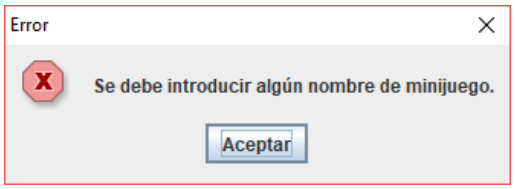


Figura 187: Casos de Uso Minijuego Error Nombre

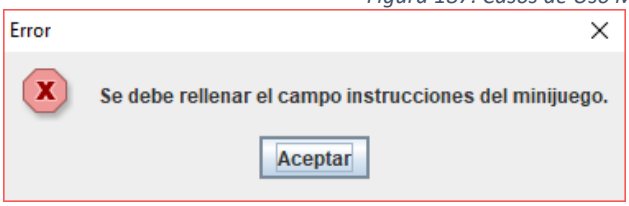


Figura 188: Casos de Uso Minijuego Error Instrucciones

27. Asignar Grupo a Minijuego

Nombre: Asignar Grupo a Minijuego.

Descripción: Permite asignar un grupo a un minijuego. Para ello, se deberá seleccionar el grupo que se quiere asignar entre los posibles.

Actores: Logopeda.

Precondición: Haber seleccionado algún minijuego.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se rellena la última fila de la tabla con el grupo que se quiere introducir y se pulsa en el botón *Asignar Grupo a Minijuego*.
[Si se pulsa sobre el botón *Asignar Grupo a Minijuego*]
    2.a. Se asigna el grupo al minijuego (Ver Figura 189).
[si no]
    2.b. Aparece un error de campos obligatorios (Ver Figura 190).
[Fin si]

Poscondiciones: Se asignará un grupo al minijuego seleccionado.

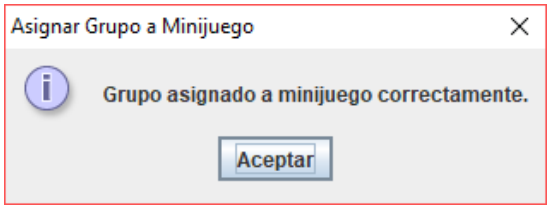


Figura 189: Casos de Uso Minijuego Asignado

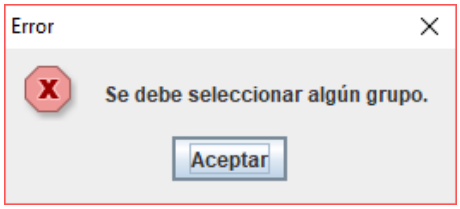


Figura 190: Casos de Uso Minijuego Error Asignado

28. Eliminar Grupo de Minijuego

Nombre: Eliminar Grupo de Minijuego.

Descripción: Permite eliminar un grupo de un minijuego.

Actores: Logopeda.

Precondición: Haber seleccionado algún minijuego

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se selecciona la celda *Eliminar* de los grupos que se quieran eliminar y se pulsa en el botón Eliminar *Grupos Seleccionados de Minijuego*.
[Si se ha seleccionado algún grupo]
    2.a. Se eliminan los grupos seleccionados (Ver Figura 191).
[si no]
    2.b. Aparece un error de seleccionar algún grupo (Ver Figura 192).
[Fin si]

Poscondiciones: Se eliminará un grupo del minijuego seleccionado.

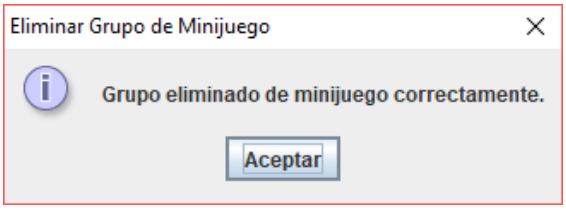


Figura 191: Casos de Uso Minijuego Eliminado

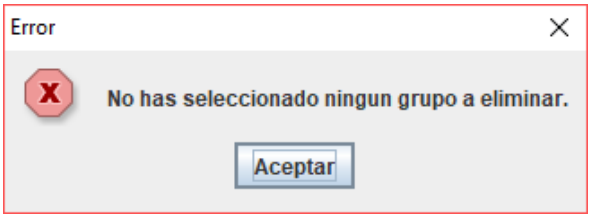


Figura 192: Casos de Uso Minijuego Error Eliminado

29. Gestionar Permisos

Nombre: Gestionar Premios.

Descripción: Permite acceder a la ventana donde se gestionan los permisos.

Actores: Logopeda.

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se visualiza una tabla, dos selectores y dos botones (Ver Figura 193).
[Si se selecciona *Minijuegos* en el primero selector]
    2.a. Se selecciona un minijuego en el segundo selector.
    [Si se pulsa *Asignar Paciente*]
        3.a.a. EXTENDS Asignar Minijuego.
    [Fin si]
    [Si se pulsa *Eliminar Paciente*]
        4.a.a. EXTENDS Eliminar Permiso Minijuego.
    [Fin si]
[si no]
    2.a. Se selecciona un grupo en el segundo selector.
    [Si se pulsa *Asignar Paciente*]
        3.a.a. EXTENDS Asignar Grupo.
    [Fin si]
    [Si se pulsa *Eliminar Paciente*]
        4.a.a. EXTENDS Eliminar Permiso Grupo.
    [Fin si]
[Fin si]

Poscondiciones: Ninguna.

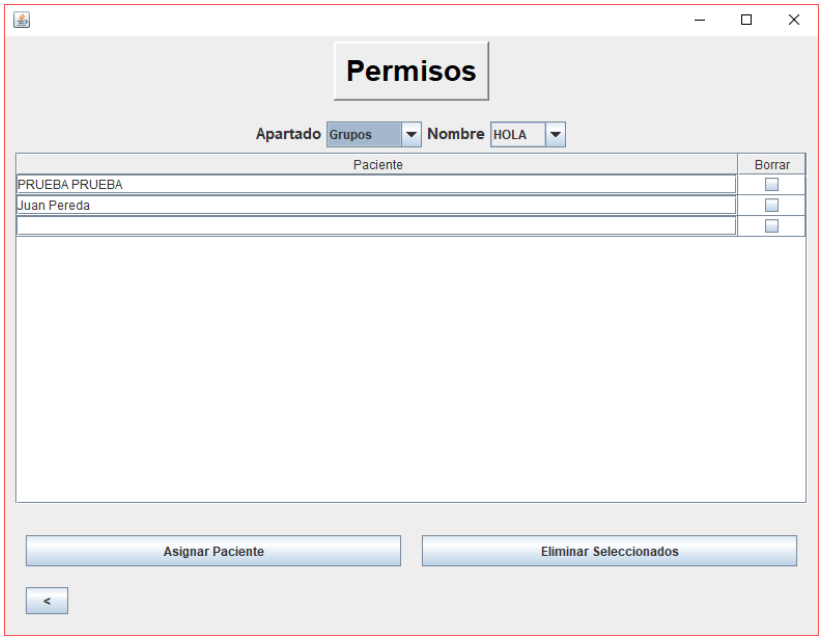


Figura 193: Casos de Uso Permisos

30. Asignar Grupo

Nombre: Asignar Grupo.

Descripción: Permite asignar un grupo a un paciente. Para ello, se deberá seleccionar el paciente al que se le quiere asignar el grupo.

Actores: Logopeda.

Precondición: Haber creado y seleccionado algún grupo. Haber creado algún paciente.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se selecciona un paciente en la última fila de la tabla y se pulsa el botón Asignar Paciente.
[Si se ha seleccionado un paciente]
    2.a. Se guarda el permiso del paciente en el grupo (Ver Figura 194).
[si no]
    2.b. Aparece un error de campos obligatorios (Ver Figura 195).
[Fin si]

Poscondiciones: Se asignará un permiso del grupo seleccionado a un paciente.

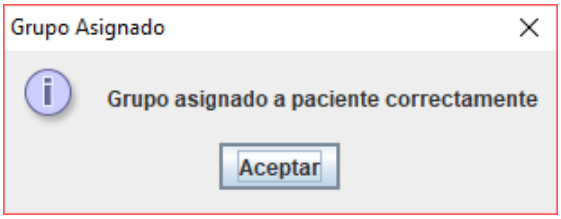


Figura 194: Casos de Uso Permisos Grupo Asignado

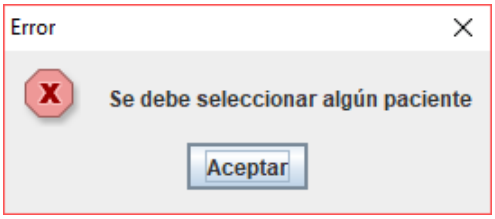


Figura 195: Casos de Uso Permisos Grupo Asignado Error

31. Eliminar Permiso Grupo

Nombre: Eliminar Permiso Grupo.

Descripción: Permite eliminar el permiso de un grupo a un paciente.

Actores: Logopeda.

Precondición: Haber creado y seleccionado algún grupo.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se hace click en la celda Eliminar de cada paciente que se desee eliminar y se pulsa el botón Eliminar Paciente.
[Si se ha seleccionado alguna fila]
    2.a. Se elimina el permiso del paciente en el grupo (Ver Figura 196).
[si no]
    2.b. Aparece un error de tener que seleccionar alguna fila (Ver Figura 197).
[Fin si]

Poscondiciones: Se eliminarán los permisos del grupo seleccionado a un paciente.

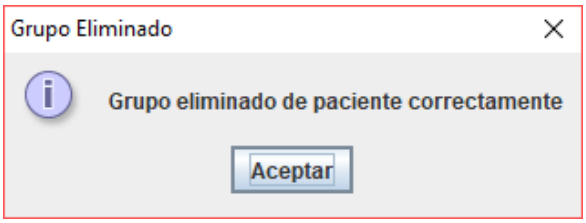


Figura 196: Casos de Uso Permisos Grupo Eliminado

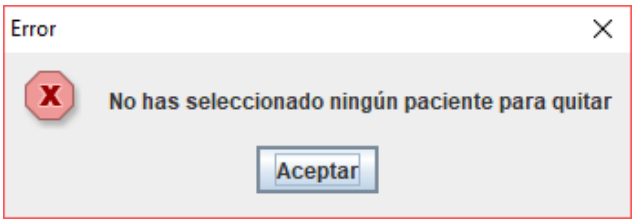


Figura 197: Casos de Uso Permisos Grupo Eliminado Error

32. Asignar Minijuego

Nombre: Asignar Minijuego.

Descripción: Permite asignar un minijuego a un paciente. Para ello, habrá que seleccionar el paciente al que se le quiere asignar el minijuego, los grupos máximos y mínimos que podrá visualizar y las vidas con las que comenzará.

Actores: Logopeda.

Precondición: Haber seleccionado algún minijuego. Haber creado algún paciente.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se selecciona un paciente, una cantidad máxima y mínima de grupos y las vidas en la última fila de la tabla y se pulsa el botón Asignar Paciente.
[Si se ha seleccionado un paciente]
2.a. Se guarda el permiso del paciente en el minijuego (Ver Figura 198).
[si no]
2.b. Aparece un error de campos obligatorios (Ver Figura 199).
[Fin si]

Poscondiciones: Se asignará un permiso del minijuego seleccionado a un paciente.

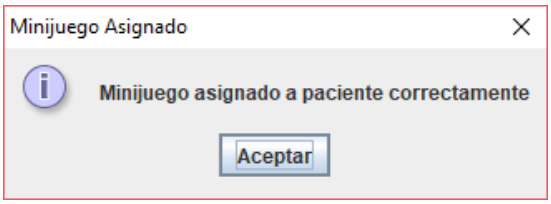


Figura 198: Casos de Uso Permisos Minijuego Asignado

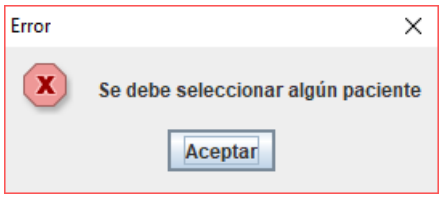


Figura 199: Casos de Uso Permisos Minijuego Asignado Error

33. Eliminar Permiso Minijuego

Nombre: Eliminar Permiso Minijuego.

Descripción: Permite eliminar el permiso de un minijuego a un paciente.

Actores: Logopeda.

Precondición: Haber creado y seleccionado algún minijuego.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se hace click en la celda Eliminar de cada paciente que se desee eliminar y se pulsa el botón Eliminar Paciente.
[Si se ha seleccionado alguna fila]
    2.a. Se elimina el permiso del paciente en el minijuego (Ver Figura 200).
[si no]
    2.b. Aparece un error de tener que seleccionar alguna fila (Ver Figura 201).
[Fin si]

Poscondiciones: Se eliminarán los permisos del minijuego seleccionado a un paciente.

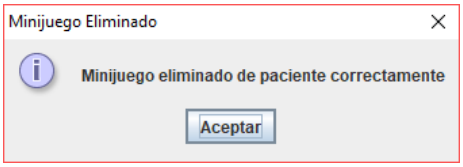


Figura 200: Casos de Uso Permisos Minijuego Eliminado

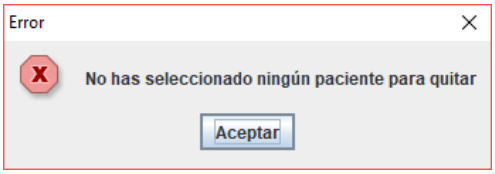


Figura 201: Casos de Uso Permisos Minijuego Eliminado Error

34. Ver Sesión

Nombre: Ver Sesión.

Descripción: Permite ver las sesiones que la Logopeda. le ha creado al paciente conectado. Para ver alguna sesión, la Logopeda. deberá haber creado alguna sesión y haberla puesto como visible por el tutor.

Actores: Paciente

Precondición: Haber creado alguna sesión para el paciente por la logopeda.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1.Accede a la ventana de Historial Sesión (Ver Figura 202).
2.Se visualizan las diferentes sesiones creadas para el paciente.

Poscondiciones: Se visualizarán las sesiones del paciente.

Figura 202: Casos de Uso Historial Sesión Paciente

35. Ver Mapas Con Botones

Nombre: Ver Mapas Con Botones.

Descripción: Permite visualizar los mapas de juego, junto con sus botones o niveles (los niveles activados, los que se han completado y los futuros).

Actores: Paciente

Precondición: Ninguna.

Requisitos no funcionales: Ninguno.

Flujo de eventos:

1.Se accede al mapa actual del videojuego que tiene el paciente conectado y los diferentes botones superiores (Ver Figura 203).
[Si mueve el mapa mediante desplazamientos del dedo sobre la pantalla]
2.a.Se visualizan los diferentes botones y mapas.
[Fin si]
[Si se pulsa sobre un botón rojo]
3.a.EXTENDS Minijuego.
[Fin si]
[Si se pulsa en el botón Premios]
4.a.EXTENDS Ver Premios Conseguidos.
[Fin si]
[Si se pulsa en el botón *Opciones* y luego en el botón *Desconectar*]
5.a.EXTENDS Desconectarse.
[Fin si]

Poscondiciones: Se visualizarán los mapas y sus botones.

Figura 203: Casos de Uso Mapa

36. Ver Premios Conseguidos

Nombre: Ver Premios Conseguidos.

Descripción: Permite visualizar los premios que ha conseguido. Estos premios se verán con su silueta si los ha conseguido, y la imagen en caso de no haberlo hecho.

Actores: Paciente

Precondición: Ninguna.

Requisitos no funcionales: Ninguno.

Flujo de eventos:

1.Se visualiza la ventana de premios (Ver Figura 204).
[Si un premio se ha conseguido]
    2.a. Se visualiza la imagen del premio.
[si no]
    2.b. Se visualiza la silueta de la imagen del premio.
[Fin si]

Poscondiciones: Se visualizarán los premios conseguidos y no conseguidos.

Figura 204: Casos de Uso Premios Conseguidos

37. Desconectarse

Nombre: Desconectarse.

Descripción: Permite desconectarse del videojuego. Para ello, será necesario estar conectado a Internet.

Actores: Paciente

Precondición: Ninguna.

Requisitos no funcionales: Conexión a Internet.

Flujo de eventos:

1. Se pulsa el botón Desconectar (Ver Figura 205).
2. Se muestra una ventana de desconexión.

[Si se dispone de Internet]

2.a. Se sincronizan los datos.

3.a. Se desconecta al paciente del videojuego.

4.a. Se vuelve a la pantalla de *Login*.

[si no]

2.b. Se cierra la ventana de desconexión y se vuelve al mapa.

[Fin si]

Poscondiciones: Se sincronizará y desconectará del videojuego.

Figura 205: Casos de Uso Desconectar

38. Jugar Minijuego

Nombre: Jugar Minijuego.

Descripción: Permite jugar a cualquier minijuego. Para ello deberá pulsar en cualquier botón activo del mapa, y el minijuego que se juegue dependerá del minijuego que esté asignado.

Actores: Paciente

Precondición: Ninguna.

Requisitos no funcionales: Ninguno**.**

Flujo de eventos:

1. Se pulsa sobre un botón activo (rojo) del mapa.
[Si en el botón está el minijuego *Buscar*]
    2.a. EXTENDS Buscar.
[Si en el botón está el minijuego *Sopa de Letras*]
    2.b. EXTENDS Sopa de Letras.
[Si en el botón está el minijuego *Parejas*]
    2.c. EXTENDS Parejas.
[Si en el botón está el minijuego *Palabras Incompletas*]
    2.d. EXTENDS Palabras Incompletas.
[Si en el botón está el minijuego *Frase Correcta*]
    2.e. EXTENDS Frase Correcta.
[Si en el botón está el minijuego *Seleccionar*]
    2.f. EXTENDS Seleccionar.
[Si en el botón está el minijuego *Laberinto Erróneo*]
    2.g. EXTENDS Laberinto Erróneo.
[Fin si]

Poscondiciones: Se mostrará algún minijuego.

39. Jugar Buscar

Nombre: Jugar Buscar.

Descripción: Permite jugar al minijuego *Buscar*. Para ello, la Logopeda. se lo tendrá que haber asignado, junto con los grupos correspondientes.

Actores: Paciente

Precondición: Tener asignado el minijuego *Buscar* y los grupos suficientes para poder jugarlo.

Requisitos no funcionales: Ninguno**.**

Flujo de eventos:

1. Se visualiza el minijuego *Buscar* (Ver Figura 206).
[Mientras no se haya terminado el juego]
    2. Pulsar casilla.
    [Si la casilla contiene el texto objetivo]
        3.a. Se marca la casilla como correcta (Ver Figura 207).
    [si no]
        3.b. Se resta una vida.
        4.b. Se marca la casilla como incorrecta (Ver Figura 208).
        [Fin si]
    [Fin si]
[Fin Mientras]
5. Se termina el juego.

Poscondiciones: Se completará el minijuego *Buscar.*

Figura 206: Casos de Uso Buscar

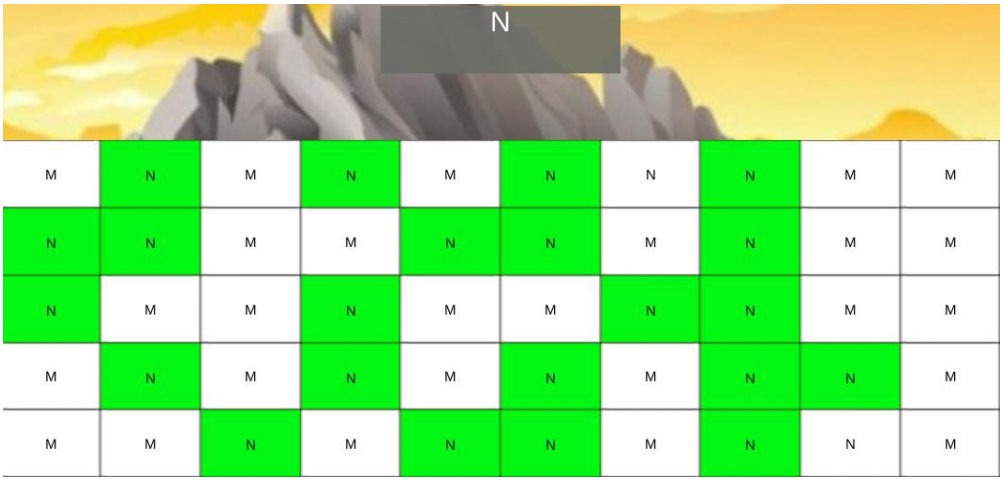


Figura 207: Casos de Uso Buscar Seleccionar

Figura 208: Casos de Uso Busca Fallo

40. Jugar Sopa de Letras

Nombre: Jugar Sopa de Letras.

Descripción: Permite jugar al minijuego *Sopa de Letras*. Para ello, la Logopeda. se lo tendrá que haber asignado, junto con los grupos correspondientes.

Actores: Paciente

Precondición: Tener asignado el minijuego *Sopa de Letras* y los grupos suficientes para poder jugarlo.

Requisitos no funcionales: Ninguno**.**

Flujo de eventos:

1. Se visualiza el minijuego *Sopa de Letras* (Ver Figura 209).
[Mientras no se haya terminado el juego]
    2. Pulsar casilla.
    3. Arrastrar en una dirección.
    4. Soltar casillas.
    [Si hay más de una casilla seleccionada]
        [Si la palabra que forman las letras es correcta]
            3.a.a. Se quedan marcadas las casillas (Ver Figura 210).
            4.a.a. Se oscurece la imagen o el texto.
        [si no]
            3.a.b. Se resta una vida.
            4.a.b. Se desmarcan las casillas.
        [Fin si]
    [Fin si]
[Fin Mientras]
5. Se termina el juego.

Poscondiciones: Se completará el minijuego *Sopa de Letras.*

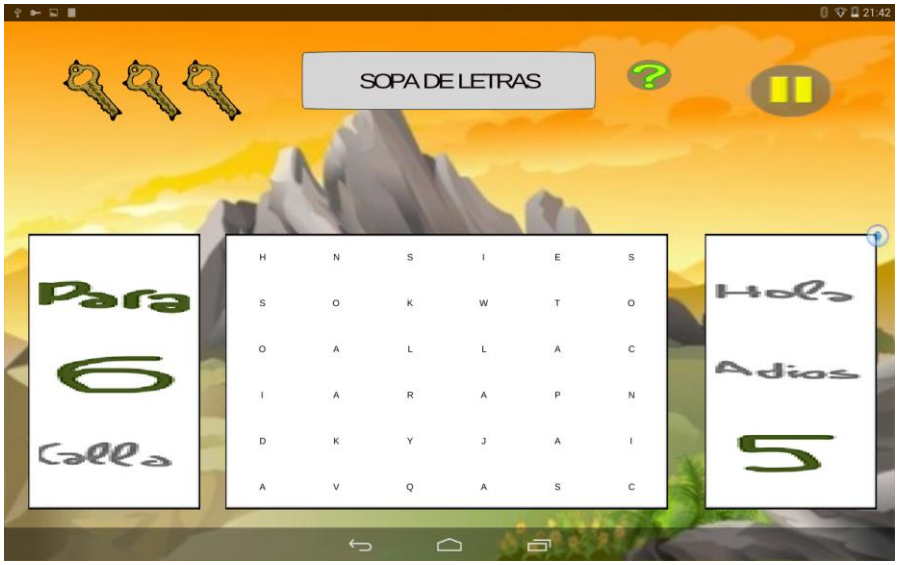


Figura 209: Casos de Uso Sopa de Letras

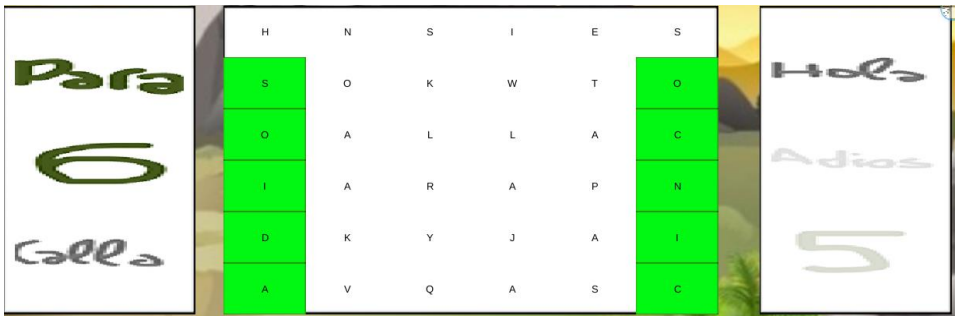


Figura 210: Casos de Uso Sopa de Letras Seleccionar

41. Jugar Parejas

Nombre: Jugar Parejas.

Descripción: Permite jugar al minijuego *Parejas*. Para ello, la Logopeda. se lo tendrá que haber asignado, junto con los grupos correspondientes.

Actores: Paciente

Precondición: Tener asignado el minijuego *Parejas* y los grupos suficientes para poder jugarlo.

Requisitos no funcionales: Ninguno**.**

Flujo de eventos:

1. Se visualiza el minijuego *Parejas* (Ver Figura 211).
[Mientras no se haya terminado el juego]
    2. Pulsar casilla.
    3. Se muestra el contenido de la casilla (Ver Figura 212)..
    [Si hay otra casilla pulsada]
        [Si tienen el mismo contenido]
            3.a.a. Se quedan activas (Ver Figura 212).
        [si no]
            3.a.b. Se resta una vida.
            4.a.b. Se desactivan las dos casillas.
        [Fin si]
    [Fin si]
[Fin Mientras]
5. Se termina el juego.

Poscondiciones: Se completará el minijuego *Parejas.*

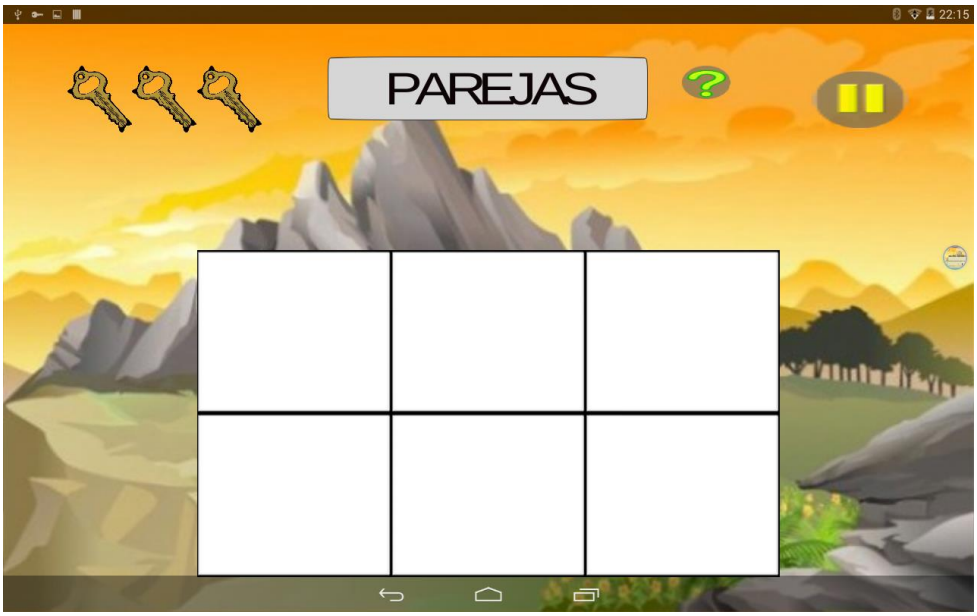


Figura 211: Casos de Uso Parejas

Figura 212: Casos de Uso Parejas Seleccionado

42. Jugar Palabras Incompletas

Nombre: Jugar Palabras Incompletas.

Descripción: Permite jugar al minijuego *Palabras Incompletas*. Para ello, la Logopeda. se lo tendrá que haber asignado, junto con los grupos correspondientes.

Actores: Paciente

Precondición: Tener asignado el minijuego *Palabras Incompletas* y los grupos suficientes para poder jugarlo.

Requisitos no funcionales: Ninguno**.**

Flujo de eventos:

1. Se visualiza el minijuego *Palabras Incompletas* (Ver Figura 213).
[Mientras no se haya terminado el juego]
    [Para todos los huecos]
        2. Pulsar hueco libre.
        3. Seleccionar una letra (Ver Figura 215).
    [Fin Para]
    4. Pulsar el botón *Validar.*
    [Si todos los huecos están correctos]
        5.a. Se termina el juego.
    [si no]
        5.b. Se muestran en rojo los huecos incorrectos (Ver Figura 214).
    [Fin si]
[Fin Mientras]

Poscondiciones: Se completará el minijuego *Palabras Incompletas.*

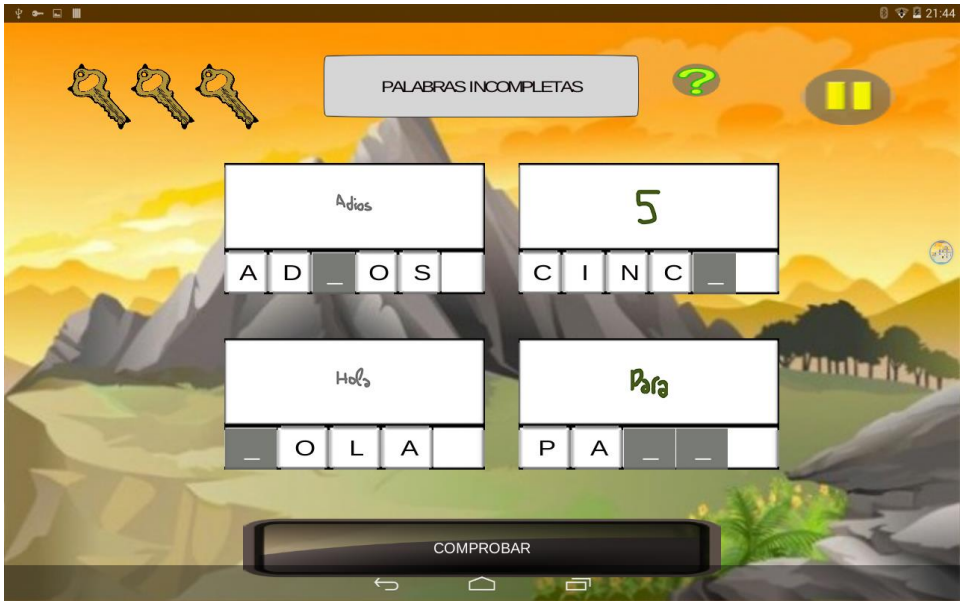


Figura 213:Casos de Uso Palabras Incompletas

Figura 214:Casos de Uso Palabras Incompletas Fallo

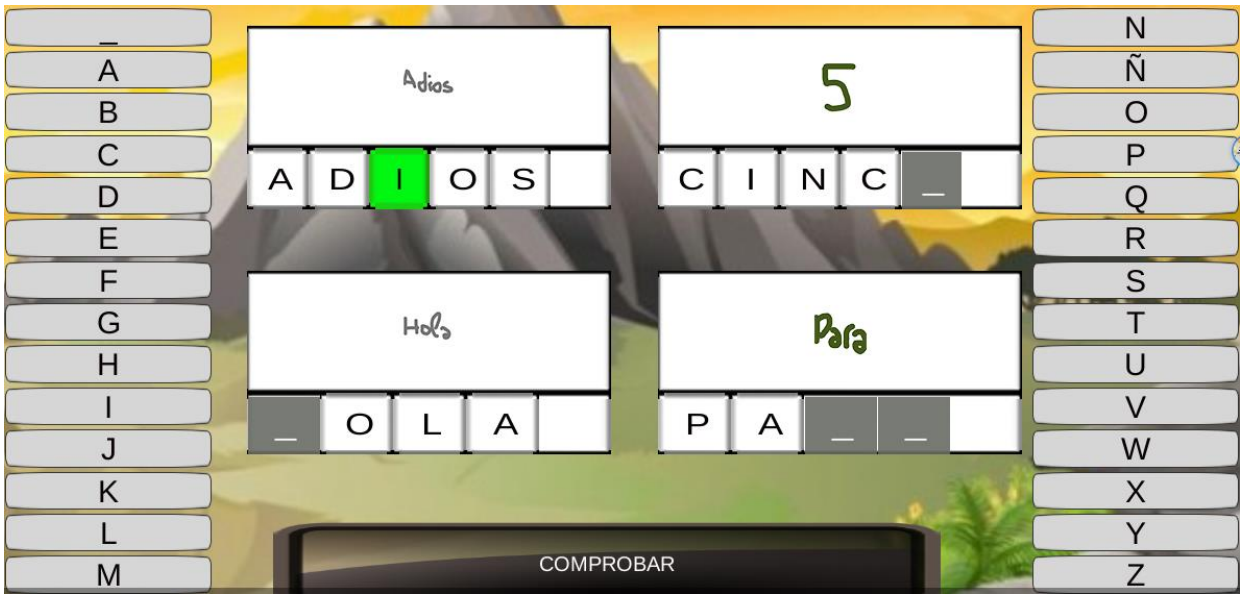


Figura 215:Casos de Uso Palabras Incompletas Letra

43. Jugar Frase Correcta

Nombre: Jugar Frase Correcta.

Descripción: Permite jugar al minijuego *Frase Correcta*. Para ello, la Logopeda. se lo tendrá que haber asignado, junto con los grupos correspondientes.

Actores: Paciente

Precondición: Tener asignado el minijuego *Frase Correcta* y los grupos suficientes para poder jugarlo.

Requisitos no funcionales: Ninguno**.**

Flujo de eventos:

1. Se visualiza el minijuego *Frase Correcta* (Ver Figura 216).
[Mientras no se haya terminado el juego]
    [Para todas las fases]
        2. Visualizar contenido.
        3. Pulsar sobre una opción. (Ver Figura 217).
        4. Pulsar el botón *Siguiente*.
    [Fin Para]
    4. Pulsar el botón *Comprobar.*
    [Si todas las fases están correctas]
      5.a.a. Se termina el juego.
    [si no]
      5.a.b. Se muestran en verde las fases correctas.
      6.a.b. Se muestran en rojo las fases incorrectas (Ver Figura 218).
      7.a.b. Se posiciona en la primera fase incorrecta.
    [Fin si]
[Fin Mientras]

Poscondiciones: Se completará el minijuego *Frase Correcta.*

Figura 216: Casos de Uso Frase Correcta

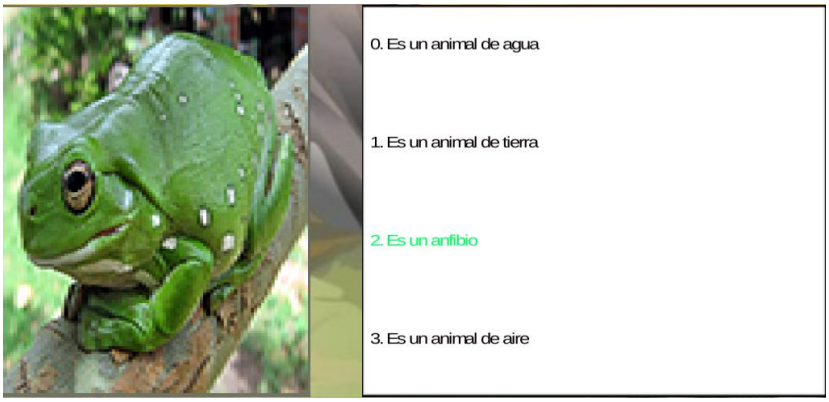


Figura 217: Casos de Uso Frase Correcta Seleccionado

Figura 218: Casos de Uso Frase Correcta Incorrecto

44. Jugar Seleccionar

Nombre: Jugar Seleccionar.

Descripción: Permite jugar al minijuego *Seleccionar*. Para ello, la Logopeda. se lo tendrá que haber asignado, junto con los grupos correspondientes.

Actores: Paciente

Precondición: Tener asignado el minijuego *Seleccionar* y los grupos suficientes para poder jugarlo.

Requisitos no funcionales: Ninguno**.**

Flujo de eventos:

1. Se visualiza el minijuego *Selecciona* (Ver Figura 219).

[Mientras no se haya terminado el juego]
  [Para todas las fases]
    2. Visualizar contenido.
    3. Pulsar sobre una opción. (Ver Figura 220).
    4. Deslizar verticalmente la mano sobre la pantalla.
  [Fin Para]
  5. Pulsar el botón *Comprobar.*
  [Si todas las fases están correctas]
    6.a.a. Se termina el juego.
  [si no]
    6.a.b. Se muestran en verde las fases correctas.
    7.a.b. Se muestran en rojo las fases incorrectas (Ver Figura 221).
    8.a.b. Se posiciona en la primera fase incorrecta.
  [Fin si]
[Fin Mientras]

Poscondiciones: Se completará el minijuego *Selecciona.*

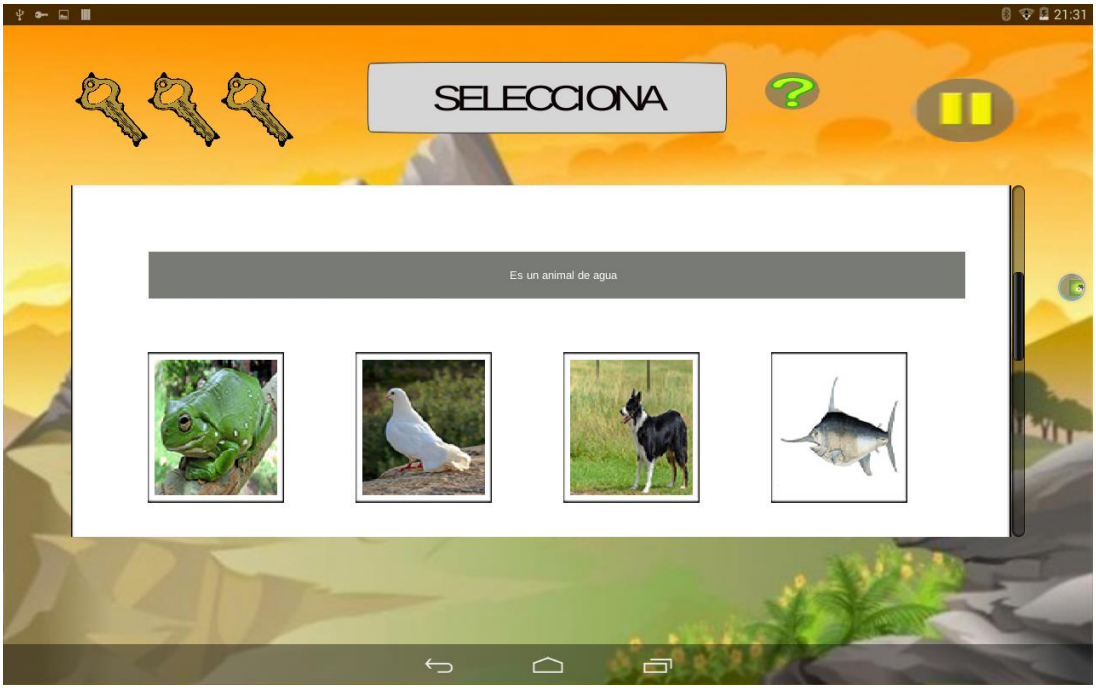


Figura 219: Casos de Uso Selecciona

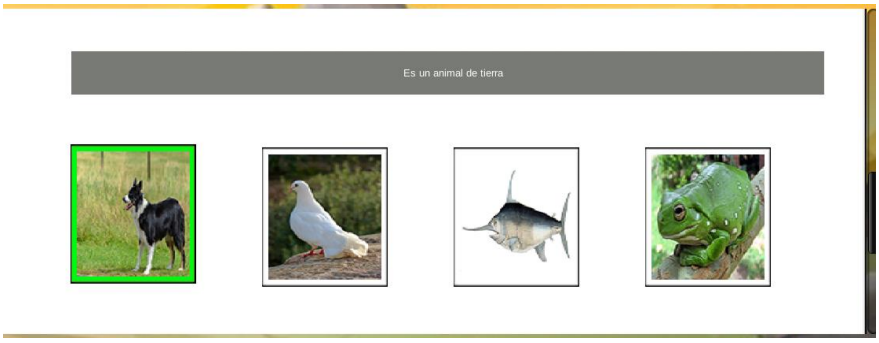


Figura 220: Casos de Uso Selecciona Opción

Figura 221: Casos de Uso Selecciona Fallo

45. Jugar Laberinto Erróneo

Nombre: Jugar *Laberinto Erróneo*.

Descripción: Permite jugar al minijuego *Laberinto Erróneo*. Para ello, la Logopeda. se lo tendrá que haber asignado.

Actores: Paciente

Precondición: Tener asignado el minijuego *Laberinto Erróneo*.

Requisitos no funcionales: Ninguno**.**

Flujo de eventos:

1. Se visualiza el minijuego *Laberinto Erróneo* (Ver Figura 222).
[Mientras no se haya terminado el juego]
    2. Se pulsa sobre una casilla visible.
    [Si es casa]
      3.a. Se muestra las casillas colindantes (arriba, abajo, derecha e izquierda (Ver Figura 223).
    [Si es llave]
      3.b. Se termina el juego (Ver Figura 224).
    [Si es correcta]
      3.c. Se muestra las casillas colindantes (arriba, abajo, derecha e izquierda (Ver Figura 224).
    [si no]
      3.d. Se resta una vida.
    [Fin si]
[Fin Mientras]

Poscondiciones: Se completará el minijuego *Laberinto Erróneo.*

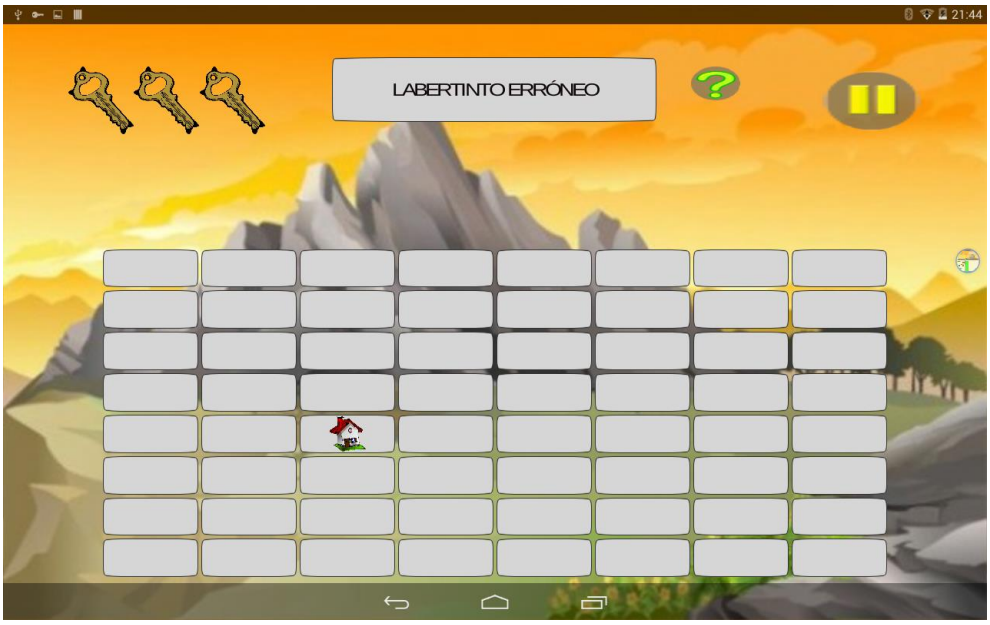


Figura 222: Casos de Uso Laberinto Erróneo

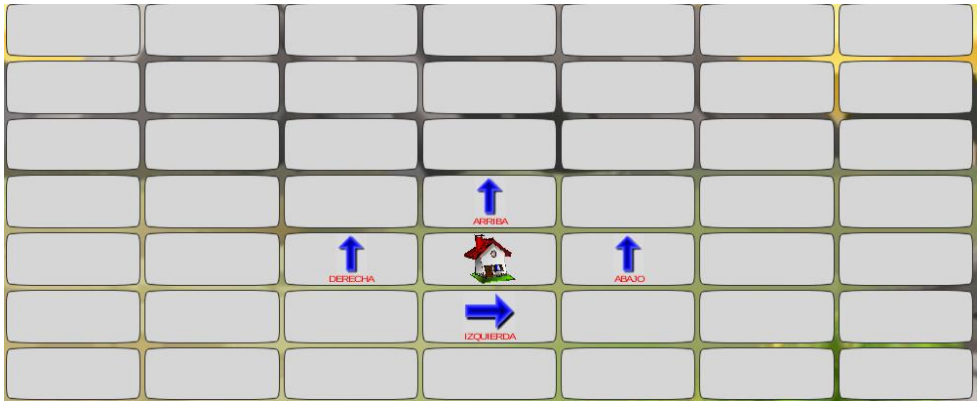


Figura 223: Casos de Uso Laberinto Erróneo Casa

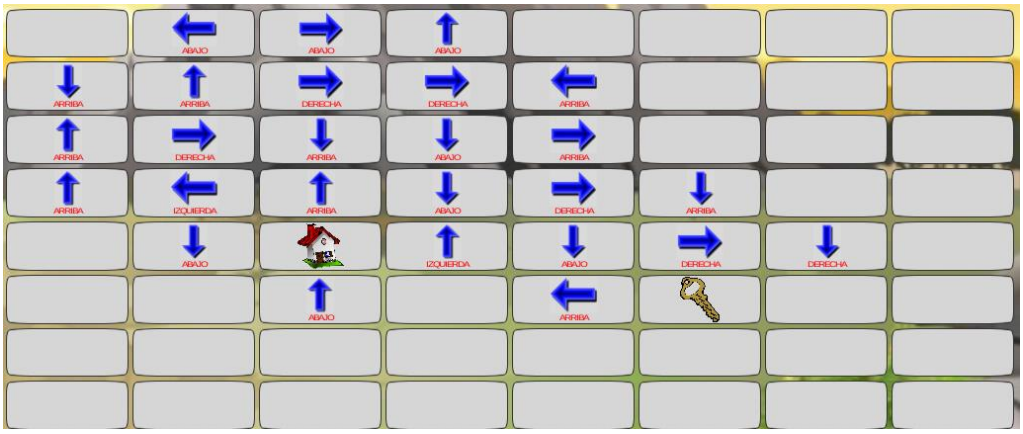


Figura 224: Casos de Uso Laberinto Erróneo Fin

ANEXO II

Diagramas de Secuencia

Login Gestor

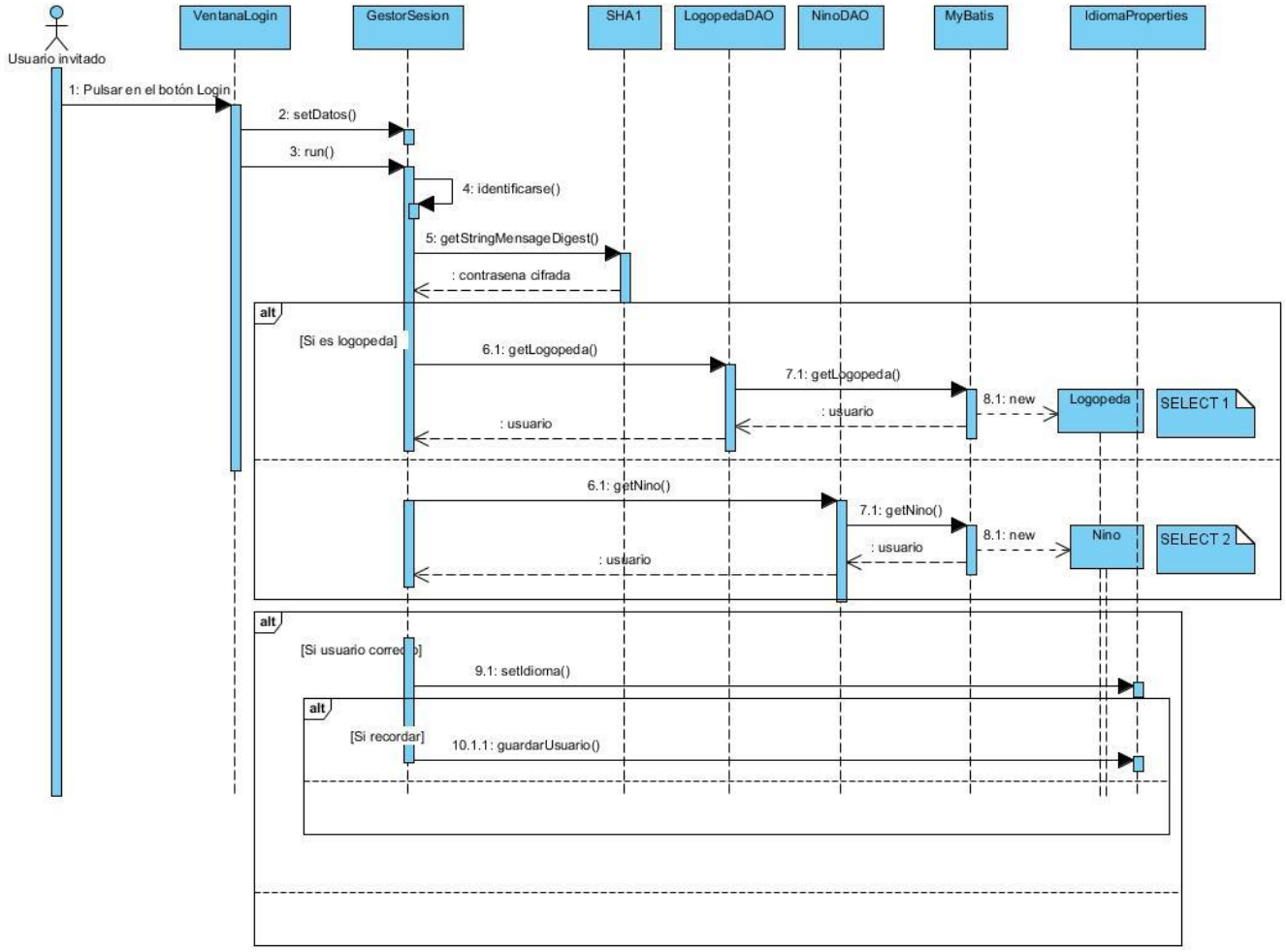


Figura 225: Diagrama de Secuencia Login Gestor

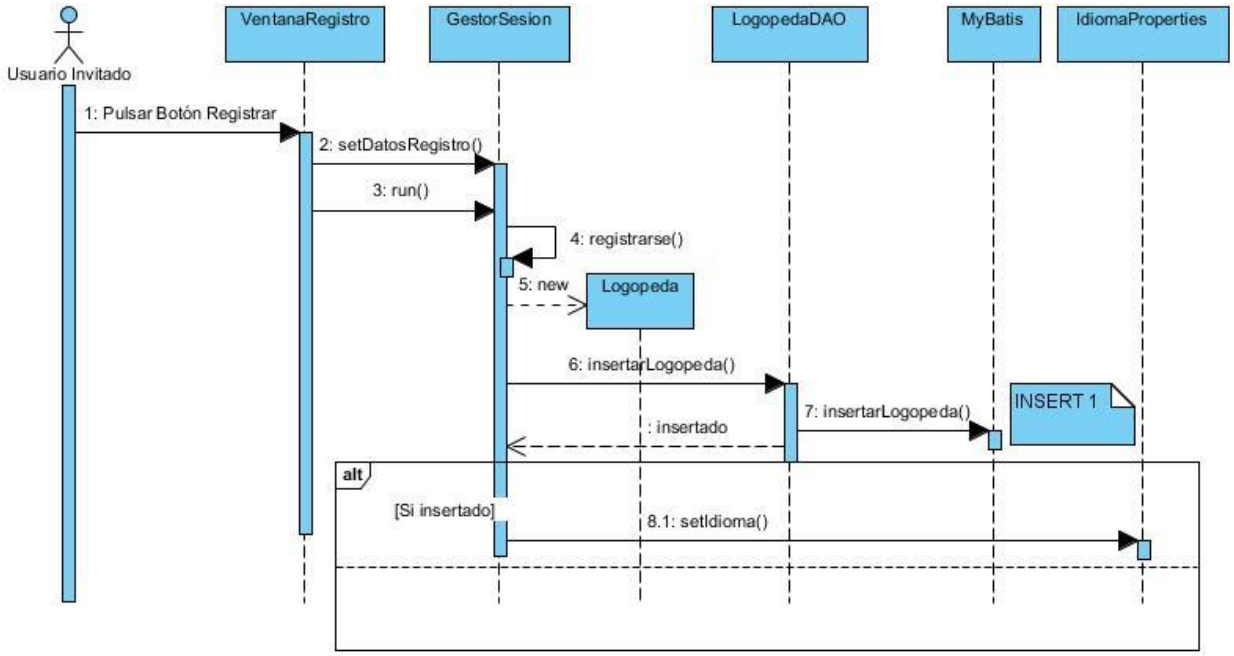
SELECT 1

```
SELECT L_ID_LOGOPEDA, L_USUARIO, L_NOMBRE, L_APELLIDOS, L_CONTRASENA, L_IDIOMA
FROM LOGOPEDA
WHERE L_USUARIO = #{usuario}
  AND L_CONTRASENA = #{contrasena}
```

SELECT 2

```
SELECT *
FROM NINO
LEFT JOIN ANAMNESIS_NINO AN ON N_ID_NINO=AN.AN_ID_NINO
LEFT JOIN AUTONOMIA_NINO AU ON N_ID_NINO=AU.AN_ID_NINO
LEFT JOIN SOCIALIZACION_NINO ON N_ID_NINO=SN_ID_NINO
LEFT JOIN LENGUAJE_NINO ON N_ID_NINO=LN_ID_NINO
WHERE N_USUARIO = #{usuario}
  AND N_CONTRASENA = #{contrasena}
```

Registrar



INSERT 1

```
INSERT INTO LOGOPEDA
(
L_ID_LOGOPEDA, L_USUARIO, L_NOMBRE, L_APELLIDOS, L_CONTRASENA, L_IDIOMA
)
(
	SELECT IFNULL(MAX(L_ID_LOGOPEDA),0)+1, #{usuario}, #{nombre}, #{apellidos},
		#{contrasena},#{idioma}
	FROM LOGOPEDA
)
```

ObtenerNinos

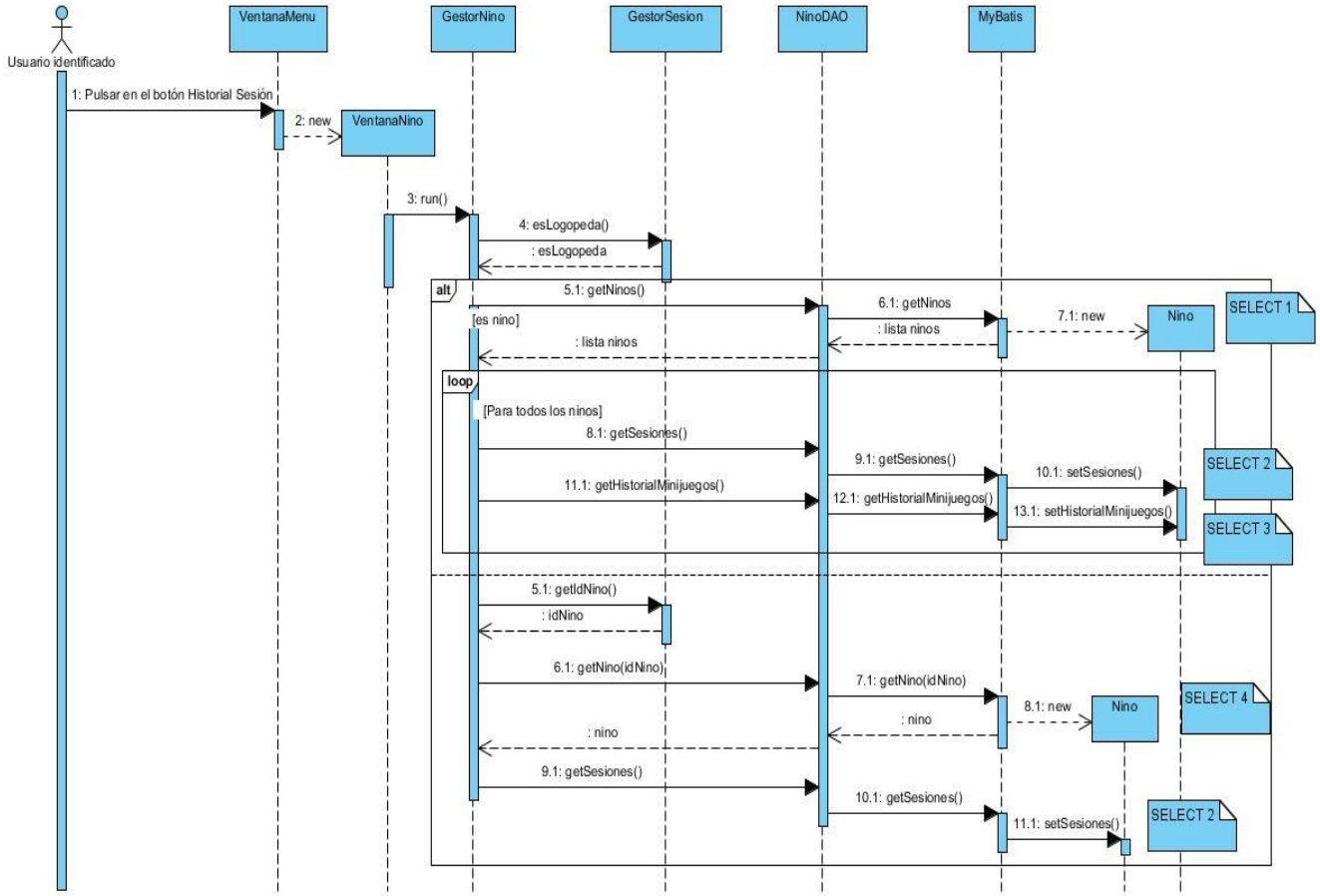


Figura 226: Diagrama de Secuencia ObtenerNino

SELECT 1

```
SELECT *
FROM NINO
LEFT JOIN ANAMNESIS_NINO AN ON N_ID_NINO=AN.AN_ID_NINO
LEFT JOIN AUTONOMIA_NINO AU ON N_ID_NINO=AU.AN_ID_NINO
LEFT JOIN SOCIALIZACION_NINO ON N_ID_NINO=SN_ID_NINO
LEFT JOIN LENGUAJE_NINO ON N_ID_NINO=LN_ID_NINO
```

SELECT 2

```
SELECT S_ID_NINO, S_ID_SESION, S_VER, S_COMENTARIO, S_FECHA
FROM SESION_NINO
WHERE S_ID_NINO = #{idUsuario}
ORDER BY S_FECHA DESC
```

SELECT 3

```
SELECT M_NOMBRE, HJ_FECHA, HJ_COMPLETO, HJ_VIDAS, HJ_ERRORES, HJ_TIEMPO
FROM HISTORIAL_JUEGO
INNER JOIN MINIJUEGO ON HJ_ID_MINIJUEGO = M_ID_MINIJUEGO
WHERE HJ_ID_NINO = #{idUsuario}
ORDER BY HJ_FECHA DESC
```

SELECT 4

```
SELECT *
FROM NINO
LEFT JOIN ANAMNESIS_NINO AN ON N_ID_NINO=AN.AN_ID_NINO
LEFT JOIN AUTONOMIA_NINO AU ON N_ID_NINO=AU.AN_ID_NINO
LEFT JOIN SOCIALIZACION_NINO ON N_ID_NINO=SN_ID_NINO
LEFT JOIN LENGUAJE_NINO ON N_ID_NINO=LN_ID_NINO
WHERE N_ID_NINO = #{idNino}
```

InsertarPaciente

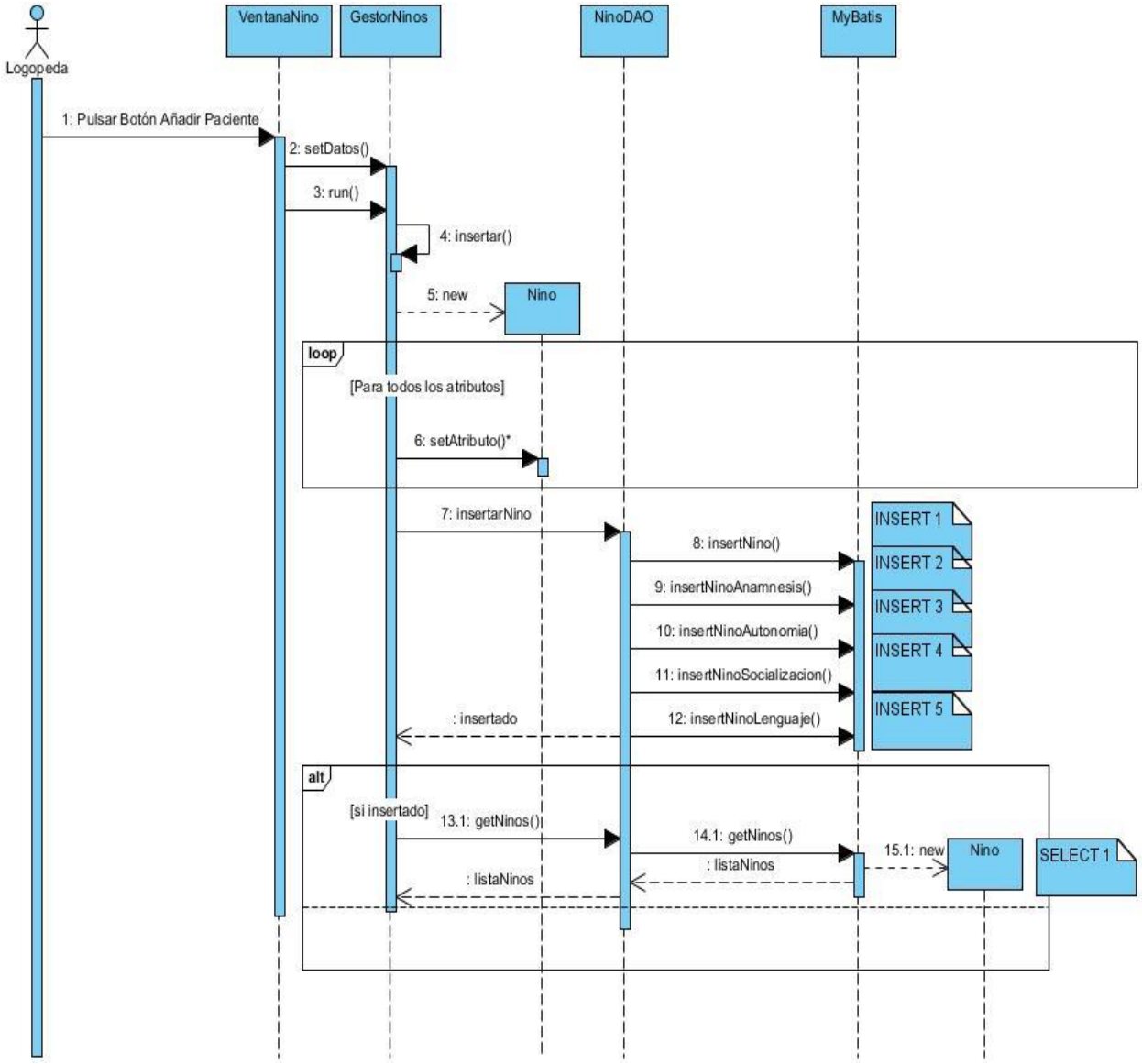


Figura 227: Diagrama de Secuencia Insertar Paciente

La llamada 6 (*setAtributo()*) no existe como tal. Se ha puesto de esta forma para evitar poner los setters de los 60 atributos, y globalizar, pero habría que quitar el *loop* y poner cada *setter* para ser exacto.

INSERT 1

```
INSERT INTO NINO
(
	N_ID_NINO, N_USUARIO, N_NOMBRE, N_APELLIDOS, N_CONTRASENA, N_IDIOMA,
	N_IDIOMA_JUEGO, N_FECHA_NACIMIENTO, N_TELEFONO_1, N_TELEFONO_2,
	N_NOMBRE_PADRE, N_NOMBRE_MADRE, N_PROFESION_PADRE, N_PROFESION_MADRE,
	N_HERMANOS_EDADES, N_OTRAS_CONVIVENCIAS, N_MOTIVO_CONSULTA, N_ENTREVISTADOS
)
VALUES
(
	#{idUsuario}, #{usuario}, #{nombre}, #{apellidos}, #{contrasena}, #{idioma},
	#{idiomaJuego}, #{fechaNacimiento}, #{telefono1}, #{telefono2},
	#{nombrePadre}, #{nombreMadre}, #{profesionPadre}, #{profesionMadre},
	#{hermanosEdades}, #{otrasConvivencias}, #{motivoConsulta}, #{entrevistados}
)
```

INSERT 2

```
INSERT INTO ANAMNESIS_NINO
(
	AN_ID_NINO, AN_EMBARAZO_PARTO, AN_DES_FISICO, AN_DES_MOTOR, AN_DES_OROFACIAL,
	AN_DES_LENGUAJE, AN_ANTECEDENTES_FAMILIARES, AN_DATOS_MEDICOS,
	AN_HISTORIA_ESCOLAR, AN_ATEN_FUERA_CENTRO
)
VALUES
(
	#{idUsuario}, #{embarazoParto}, #{desFisico}, #{desMotor}, #{desOrofacial},
	#{desLenguaje}, #{antecedentesFamiliares}, #{datosMedicos},
	#{historialEscolar}, #{atenFueraCentro}
)
```

INSERT 3

```
INSERT INTO AUTONOMIA_NINO
(
AN_ID_NINO, AN_TIPO_ALIMENTOS, AN_ALIMENTOS_PREFERIDOS, AN_RECONOCE_COMER,
AN_PIDE_ALIMENTO, AN_BIBERON_CHUPETE, AN_MASTICA, AN_SUCCIONA_TRAGA, AN_MUERDE_COSAS,
AN_SALIVA, AN_HORAS_SUENO, AN_INTENCION_DORMIR, AN_RECONOCE_DORMIR, AN_RONCA,
AN_CONTROL_ESFINTERES, AN_AVISA_SUCIO, AN_PIDE_WC, AN_RECONOCE_ROPA,
AN_NECESIDAD_CAMBIO_ROPA, AN_MALESTAR_SUCIO, AN_SONARSE_NARIZ
)
VALUES
(
	#{idUsuario}, #{tipoAlimentos}, #{alimentosPreferidos}, #{reconoceComer},
	#{pideAlimento}, #{biberonChupete}, #{mastica}, #{succionaTraga},
	#{muerdeCosas}, #{saliva}, #{horasSueno}, #{intencionDormir},
	#{reconoceDormir}, #{ronca}, #{controlEsfinteres}, #{avisaSucio}, #{pideWC},
	#{reconoceRopa}, #{necesidadCambioRopa}, #{malestarSucio}, #{sonarseNariz}
)
```

INSERT 4

```
INSERT INTO SOCIALIZACION_NINO
(
	SN_ID_NINO, SN_A_CARGO, SN_MAYOR_APEGO, SN_SIN_ATENDER,
	SN_RECONOCE_FAMILIARES, SN_COMPORTAMIENTO_DESCONOCIDOS, SN_JUEGO_NINOS,
	SN_RELACION_NINOS, SN_JUEGO_FUNCIONAL, SN_IMITA_ACCIONES,
	SN_OBJETOS_PREFERIDOS, SN_RECHAZA_OBJETO, SN_JUEGA_EN_CASA,
	SN_RECONOCE_JUGUETE
)
VALUES
(
	#{idUsuario}, #{aCargo}, #{mayorApego}, #{sinAtender}, #{reconoceFamiliares},
	#{comportamientoDesconocidos}, #{juegaNinos}, #{relacionNinos},
	#{juegoFunional}, #{imitaAcciones}, #{objetosPreferidos}, #{rechazaObjetos},
	#{juegaEnCasa}, #{reconoceJuguetes}
)
```

INSERT 5

```
INSERT INTO LENGUAJE_NINO
(
	LN_ID_NINO, LN_INTENCIONALIDAD_COMUNICATIVA, LN_QUIEN_COMUNICA,
	LN_COMO_COMUNICA, LN_LLAMA_ATENCION, LN_INTENTA_ATENCION, LN_PIDE_NECESITA,
	LN_GESTO_RECHAZO, LN_AGRADO, LN_ESTADOS_EMOCION, LN_GESTO_ADIOS,
	LN_ORDENES_SENCILLAS, LN_ORDENES_COMPLEJAS, LN_SE_ENTIENDE
)
VALUES
(
	#{idUsuario}, #{intencionalidadComunicativa}, #{quienComunica},
	#{comoComunica}, #{llamaAtencion}, #{intenaAtencion}, #{pideNecesita},
	#{gestoRechazo}, #{agrado}, #{estadosEmocion}, #{gestoAdios},
	#{ordenesSencillas}, #{ordenesComplejas}, #{seEntiende}
)
```

SELECT 1

```
SELECT *
FROM NINO
LEFT JOIN ANAMNESIS_NINO AN ON N_ID_NINO=AN.AN_ID_NINO
LEFT JOIN AUTONOMIA_NINO AU ON N_ID_NINO=AU.AN_ID_NINO
LEFT JOIN SOCIALIZACION_NINO ON N_ID_NINO=SN_ID_NINO
LEFT JOIN LENGUAJE_NINO ON N_ID_NINO=LN_ID_NINO
```

CrearSesión

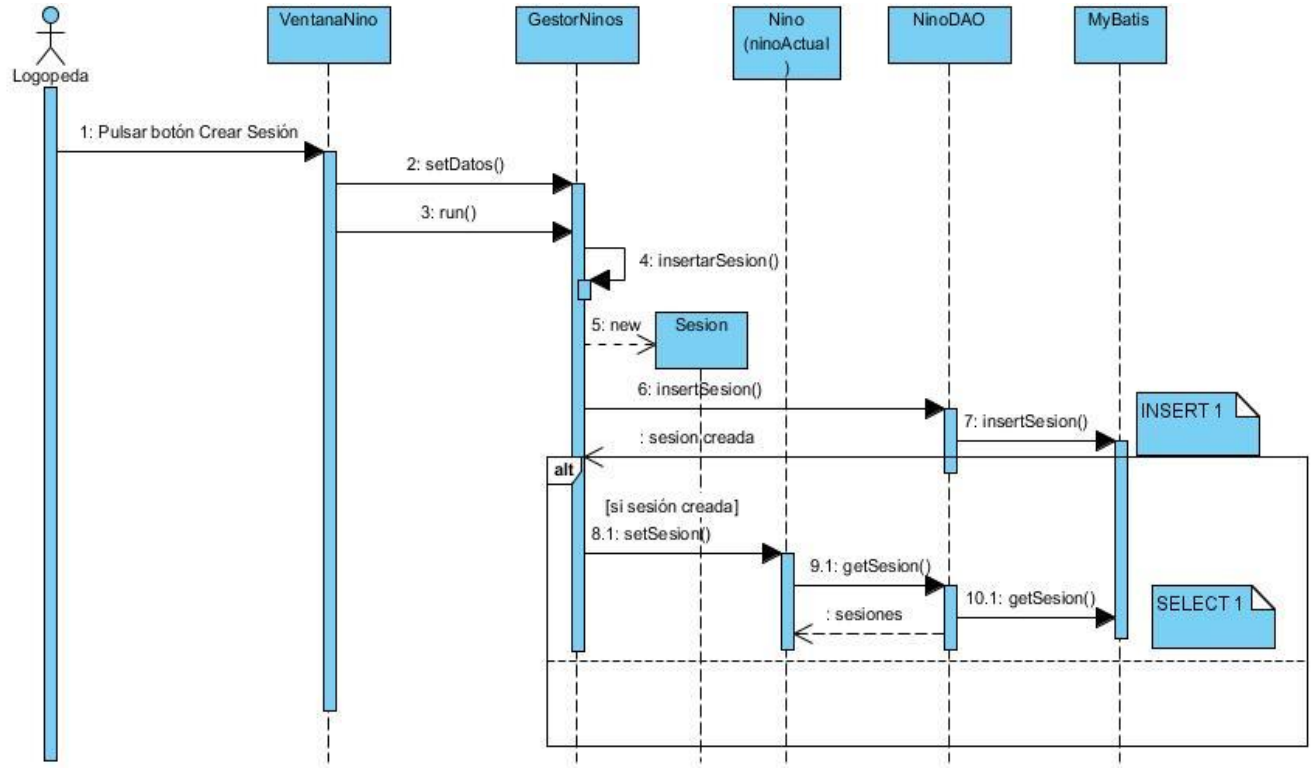


Figura 228: Diagrama de Secuencia Crear Sesión

INSERT 1

```
INSERT INTO SESION_NINO
(S_ID_NINO, S_ID_SESION, S_VER, S_FECHA, S_COMENTARIO)
VALUES
(#{idNino}, #{idSesion}, #{ver}, #{fecha}, #{comentario})
```

SELECT 1

```
SELECT S_ID_NINO, S_ID_SESION, S_VER, S_COMENTARIO, S_FECHA
FROM SESION_NINO
WHERE S_ID_NINO = #{idUsuario}
ORDER BY S_FECHA DESC
```

ObtenerFondos

El diagrama de secuencia de la Figura 229 se trata del diagrama de secuencia de la ventana de *GestorFondo*. Aun así, podría tratarse de la ventana *GestorPremio*, ya que el diagrama será el mismo (cambiando sólo los nombres de las funciones). Por esto, este diagrama se asumirá para ambas ventanas.

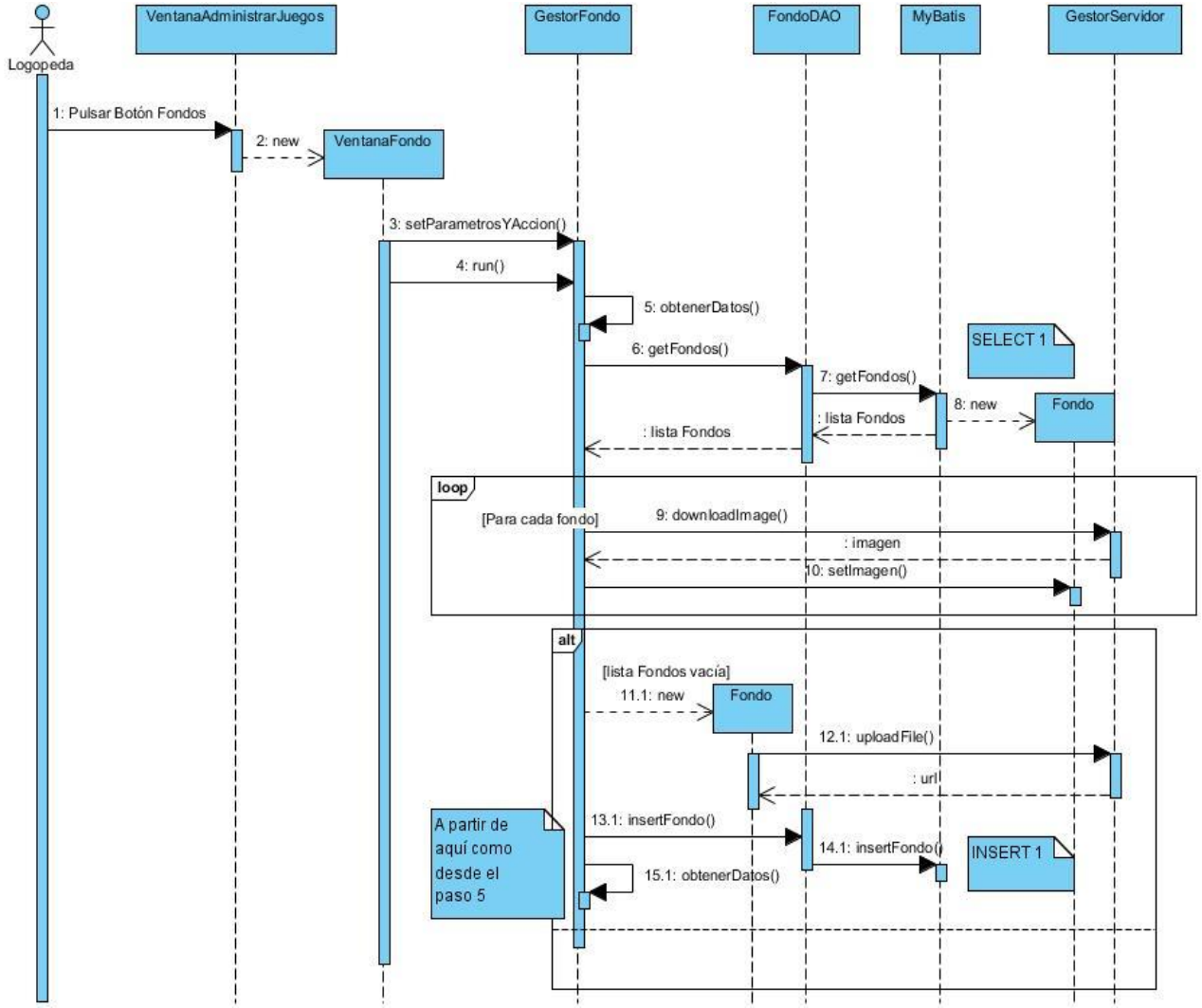


Figura 229: DIagrama de Secuencia Gestor Fondos

SELECT 1

```
SELECT F_ID_FONDO, F_NOMBRE, F_IMAGEN
FROM FONDO
```

INSERT 1

```
INSERT INTO FONDO
(F_ID_FONDO, F_NOMBRE, F_IMAGEN, F_FECMOD)
VALUES
(#{idFondo}, RTRIM(#{nombre}), #{url}, SYSDATE())
```

InsertarFondo

Este diagrama ocurre lo mismo que con el diagrama anterior, por lo que servirá tanto para insertarFondo como insertarPremio.

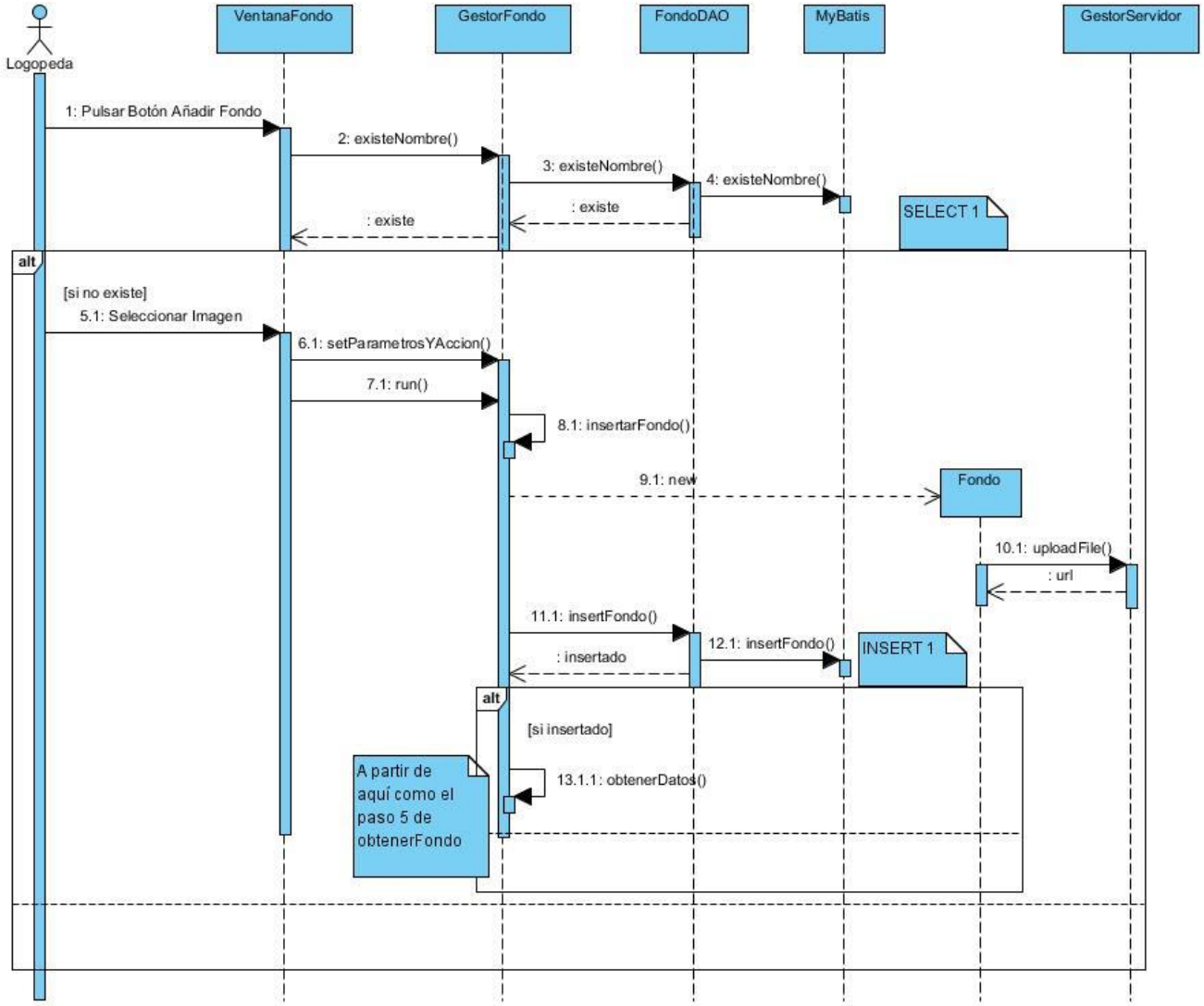


Figura 230: Diagrama de Secuencia Crear Fondo

SELECT 1

```
SELECT COUNT(1)
FROM FONDO
WHERE F_NOMBRE = RTRIM(#{nombre})
```

INSERT 1

```
INSERT INTO FONDO
(F_ID_FONDO, F_NOMBRE, F_IMAGEN, F_FECMOD)
VALUES
(#{idFondo}, RTRIM(#{nombre}), #{url}, SYSDATE())
```

ObtenerMapas

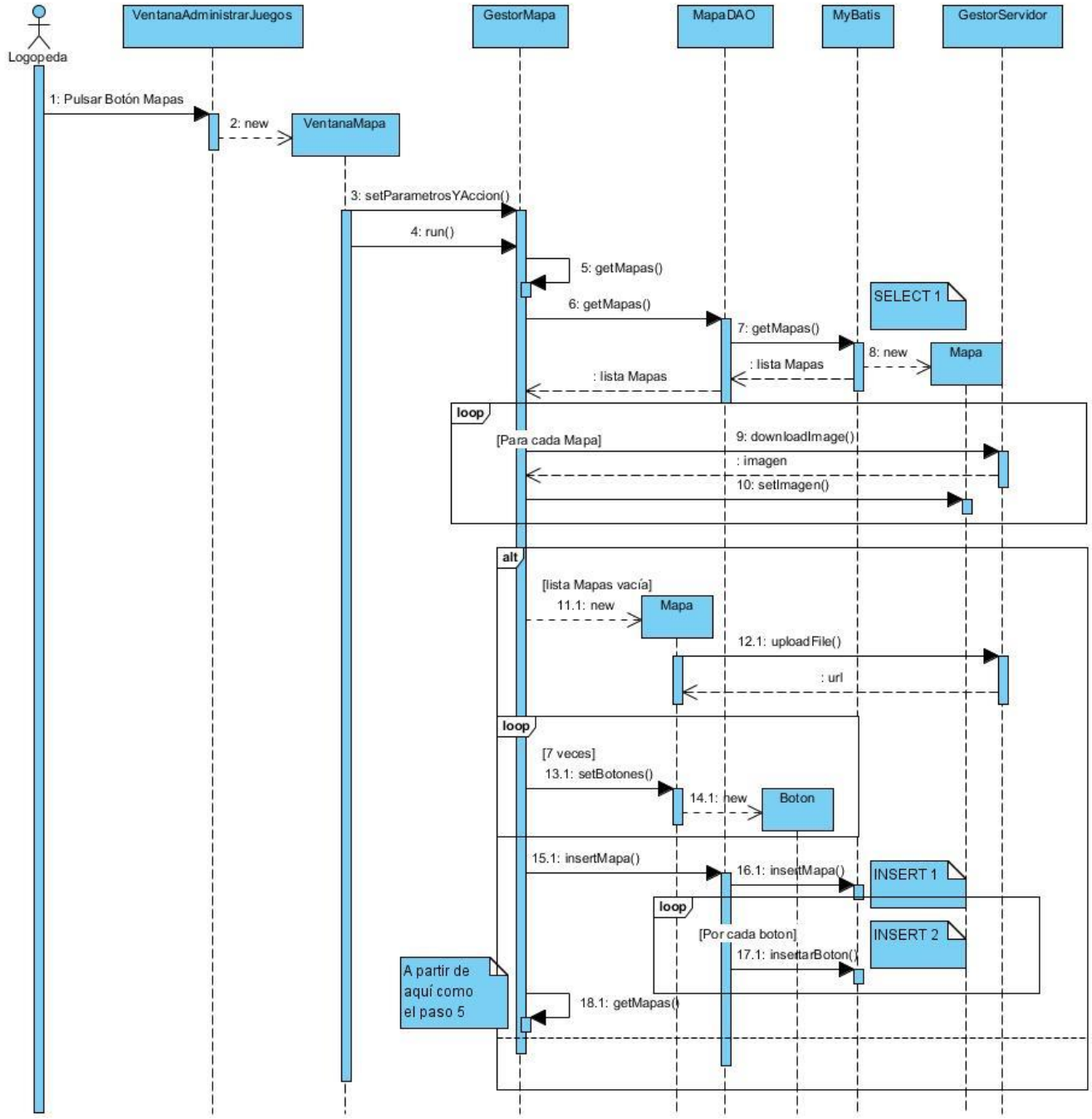


Figura 231: DIagrama de Secuencia ObtenerMapas

SELECT 1

```
SELECT MA_ID_MAPA, MA_NOMBRE, MA_IMAGEN, BM_ID_MAPA, BM_ID_BOTON, BM_POSICION_X,
      BM_POSICION_Y, BM_WIDTH, BM_HEIGHT
FROM MAPA
INNER JOIN BOTONES_MAPA ON BM_ID_MAPA = MA_ID_MAPA
```

INSERT 1

```
INSERT INTO MAPA
(MA_ID_MAPA, MA_NOMBRE, MA_IMAGEN, MA_FECMOD)
VALUES
(#{idMapa}, RTRIM(#{nombre}), #{url}, SYSDATE());
```

INSERT 2

```
INSERT INTO BOTONES_MAPA
(BM_ID_MAPA, BM_ID_BOTON, BM_POSICION_X, BM_POSICION_Y, BM_WIDTH, BM_HEIGHT)
VALUES
(#{idMapa}, #{idBoton}, #{posicionX}, #{posicionY}, #{width}, #{height});
```

InsertMapa

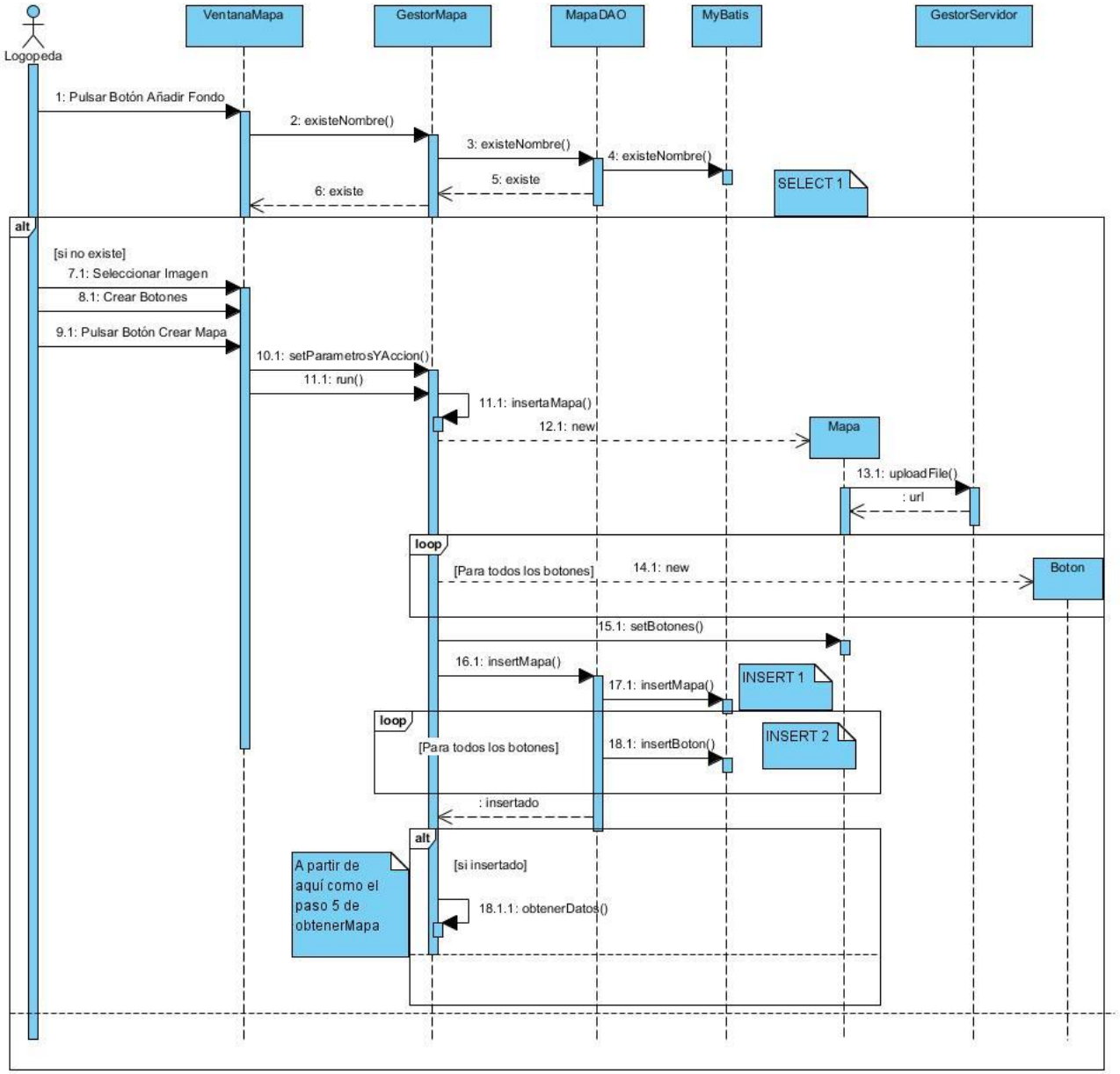


Figura 232: DIagrama de Secuencia CrearMapa

SELECT 1

```
SELECT COUNT(1)
FROM MAPA
WHERE MA_NOMBRE = RTRIM(#{nombre})
```

INSERT 1

```
INSERT INTO MAPA
(MA_ID_MAPA, MA_NOMBRE, MA_IMAGEN, MA_FECMOD)
VALUES
(#{idMapa}, RTRIM(#{nombre}), #{url}, SYSDATE());
```

INSERT 2

```
INSERT INTO BOTONES_MAPA
(BM_ID_MAPA, BM_ID_BOTON, BM_POSICION_X, BM_POSICION_Y, BM_WIDTH, BM_HEIGHT)
VALUES
(#{idMapa}, #{idBoton}, #{posicionX}, #{posicionY}, #{width}, #{height});
```

UpdateMapa

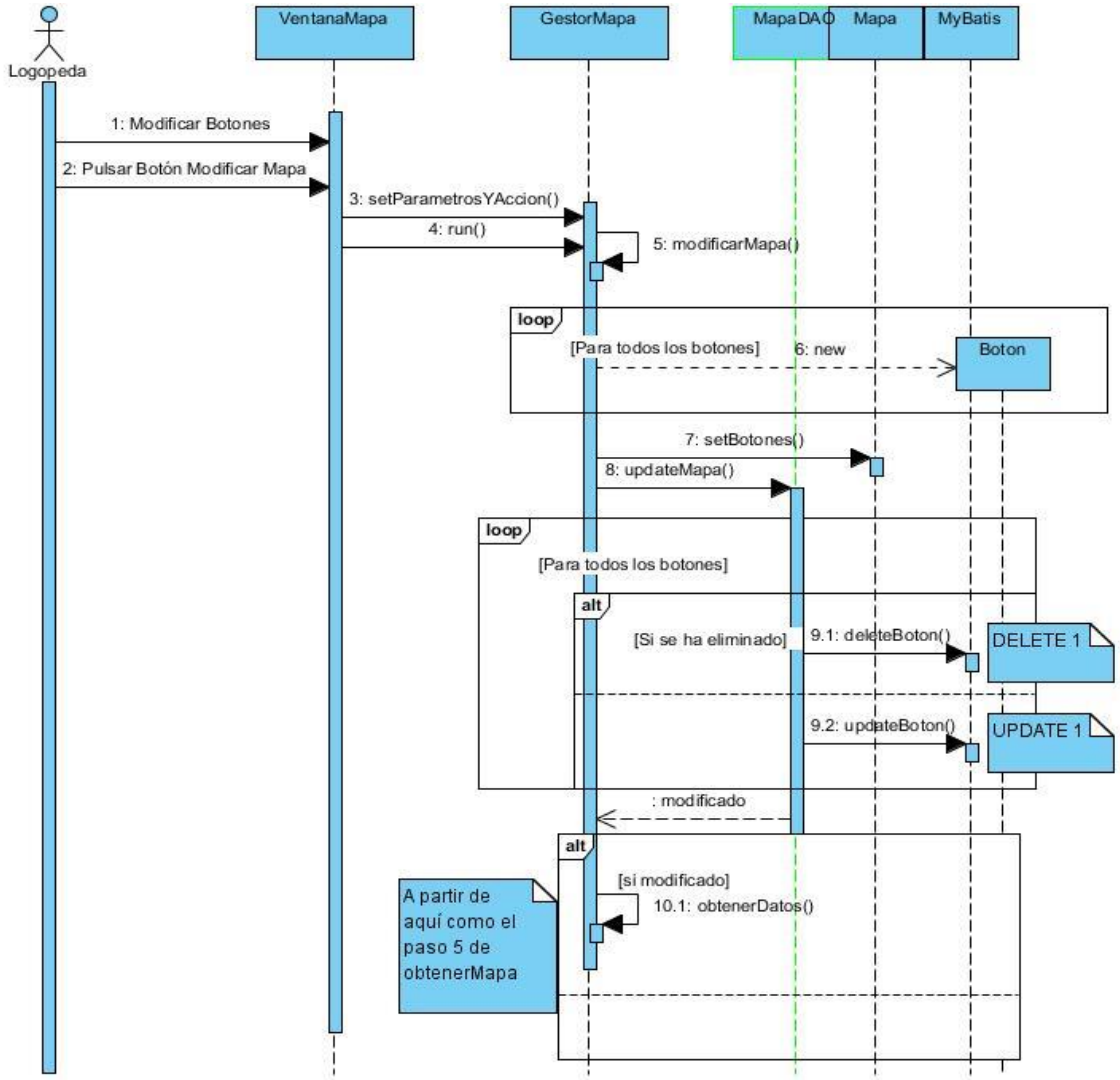


Figura 233: DIagrama de Secuencia ModificarMapa

DELETE 1

```
SELECT COUNT(1)
FROM MAPA
WHERE MA_NOMBRE = RTRIM(#{nombre})
```

UPDATE 1

```
UPDATE BOTONES_MAPA
SET BM_POSICION_X= #{posicionX},
    BM_POSICION_Y= #{posicionY},
    BM_WIDTH= #{width},
    BM_HEIGHT= #{height}
WHERE BM_ID_MAPA= #{idMapa}
  AND BM_ID_BOTON = #{idBoton}
```

ObtenerContenido

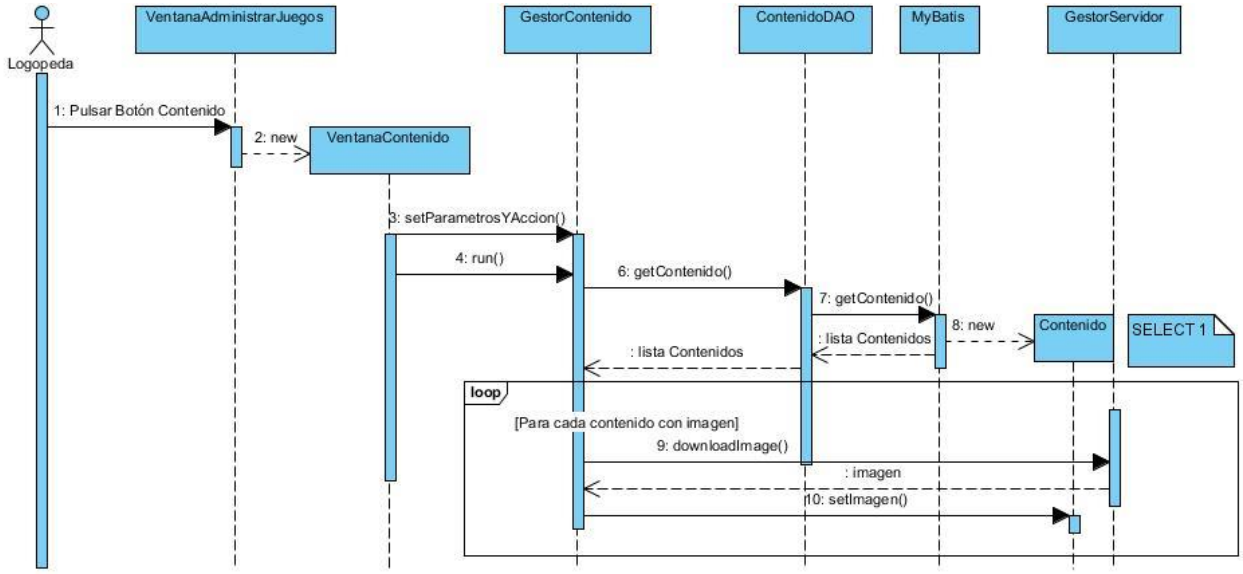


Figura 234: Diagrama de Secuencia ObtenerContenido

SELECT 1

```
SELECT C_ID_CONTENIDO, C_NOMBRE, C_CASTELLANO, C_EUSKERA, C_IMAGEN
FROM CONTENIDO
```

InsertarContenido

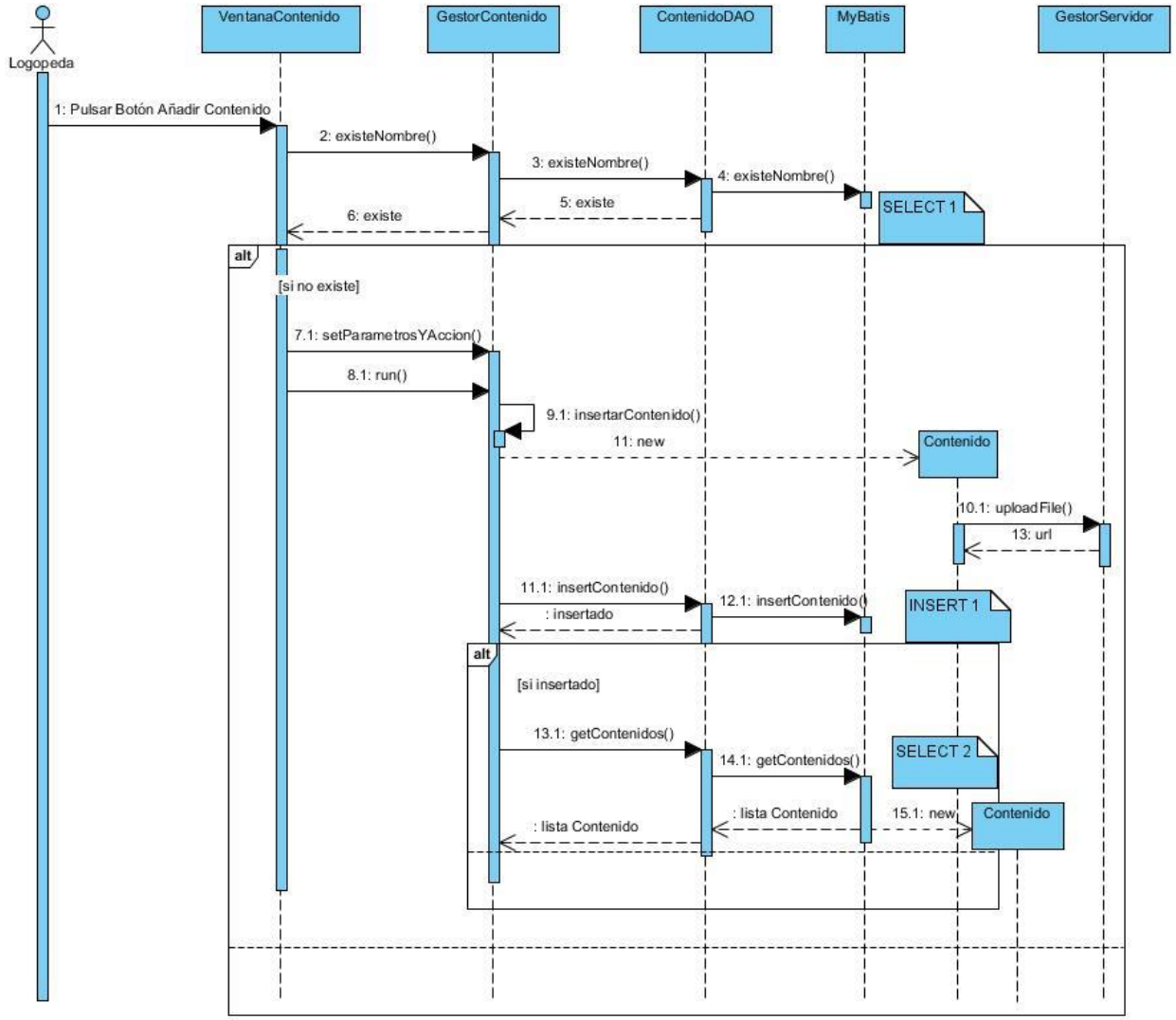


Figura 235: Diagrama de Secuencia Insertar Contenido

SELECT 1

```
SELECT C_ID_CONTENIDO, C_NOMBRE, C_CASTELLANO, C_EUSKERA, C_IMAGEN
FROM CONTENIDO
```

SELECT 2

```
SELECT COUNT(1)
FROM CONTENIDO
WHERE C_NOMBRE = RTRIM(#{nombre})
```

INSERT 1

```
INSERT INTO CONTENIDO
(C_ID_CONTENIDO, C_NOMBRE, C_CASTELLANO, C_EUSKERA, C_IMAGEN)
VALUES
(#{idContenido}, RTRIM(#{nombre}), #{castellano}, #{euskera}, #{url})
```

ModificarContenido

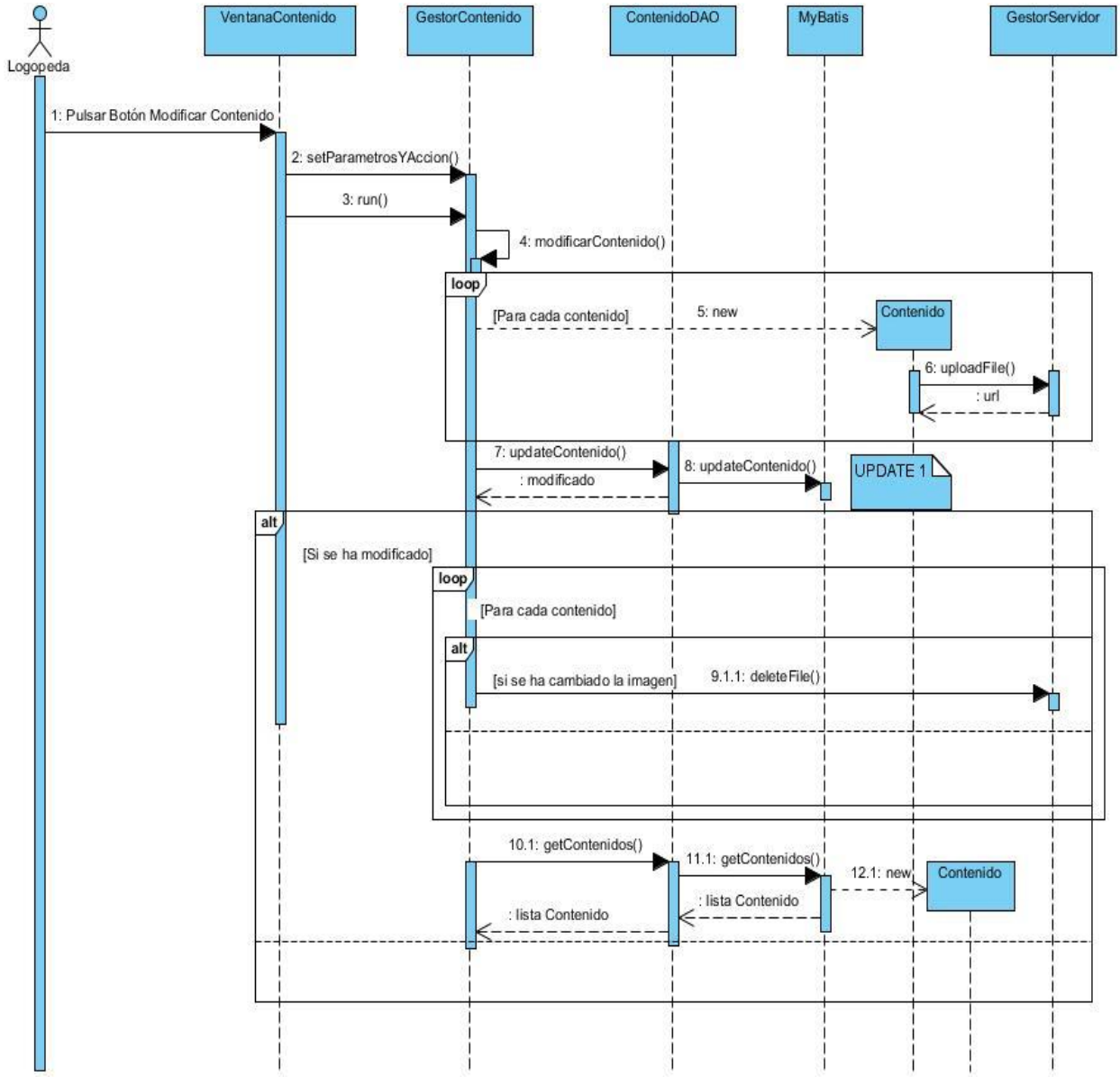


Figura 236: Diagrama de Secuencia ModificarContenido

UPDATE 1

```
UPDATE CONTENIDO
SET C_CASTELLANO = #{castellano},
    C_EUSKERA = #{euskera}
    <if test="url != null">,C_IMAGEN = #{url}</if>
WHERE C_ID_CONTENIDO = #{idContenido}
```

EliminarContenido

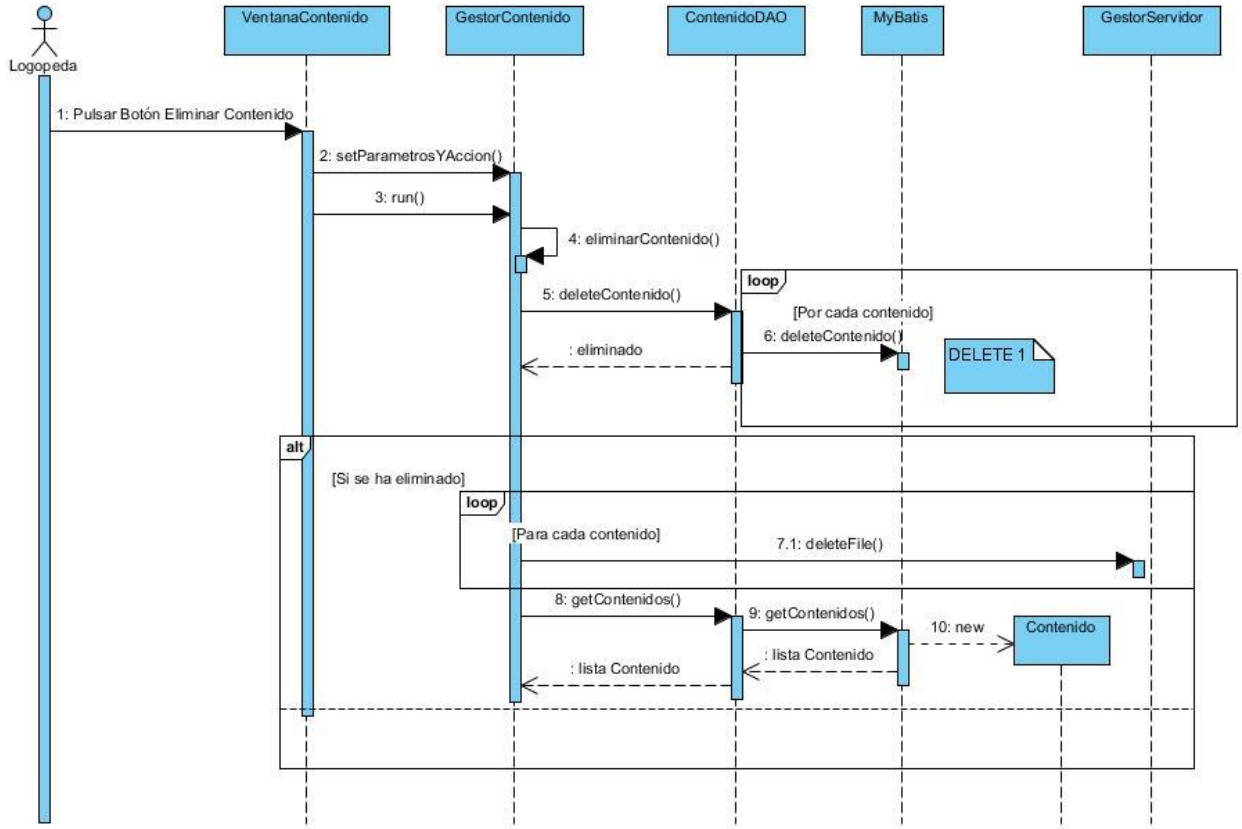


Figura 237: Diagrama de Secuencia EliminarContenido

DELETE 1

```
DELETE FROM CONTENIDO WHERE C_ID_CONTENIDO = #{idContenido}
```

ObtenerGrupos

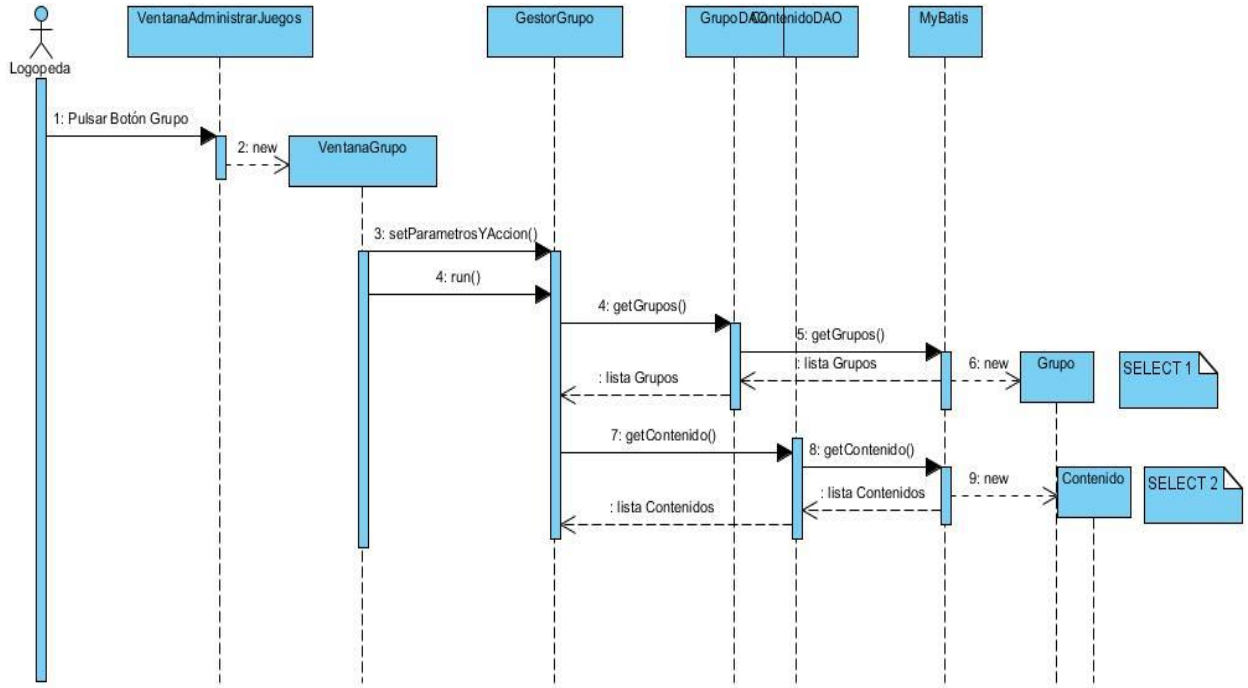


Figura 238: Diagrama de Secuencia ObtenerGR

SELECT 1

```
SELECT G_ID_GRUPO, G_NOMBRE, G_CONTENIDO_SOLUCION, S.C_NOMBRE AS S_NOMBRE,
      S.C_CASTELLANO AS S_CASTELLANO, S.C_EUSKERA AS S_EUSKERA, S.C_IMAGEN AS
      S_IMAGEN,G_CONTENIDO_OPCION, O.C_NOMBRE AS O_NOMBRE, O.C_CASTELLANO AS
      O_CASTELLANO, O.C_EUSKERA AS O_EUSKERA, O.C_IMAGEN AS O_IMAGEN
FROM GRUPO
INNER JOIN CONTENIDO S ON G_CONTENIDO_SOLUCION = S.C_ID_CONTENIDO
INNER JOIN CONTENIDO O ON G_CONTENIDO_OPCION = O.C_ID_CONTENIDO
ORDER BY G_ID_GRUPO ASC
```

SELECT 1

```
SELECT C_ID_CONTENIDO, C_NOMBRE, C_CASTELLANO, C_EUSKERA, C_IMAGEN
FROM CONTENIDO
```

InsertarGrupo

Debido a que son similares, en este apartado se referirá a la inserción de grupos, permisos y asignación grupos en minijuegos.

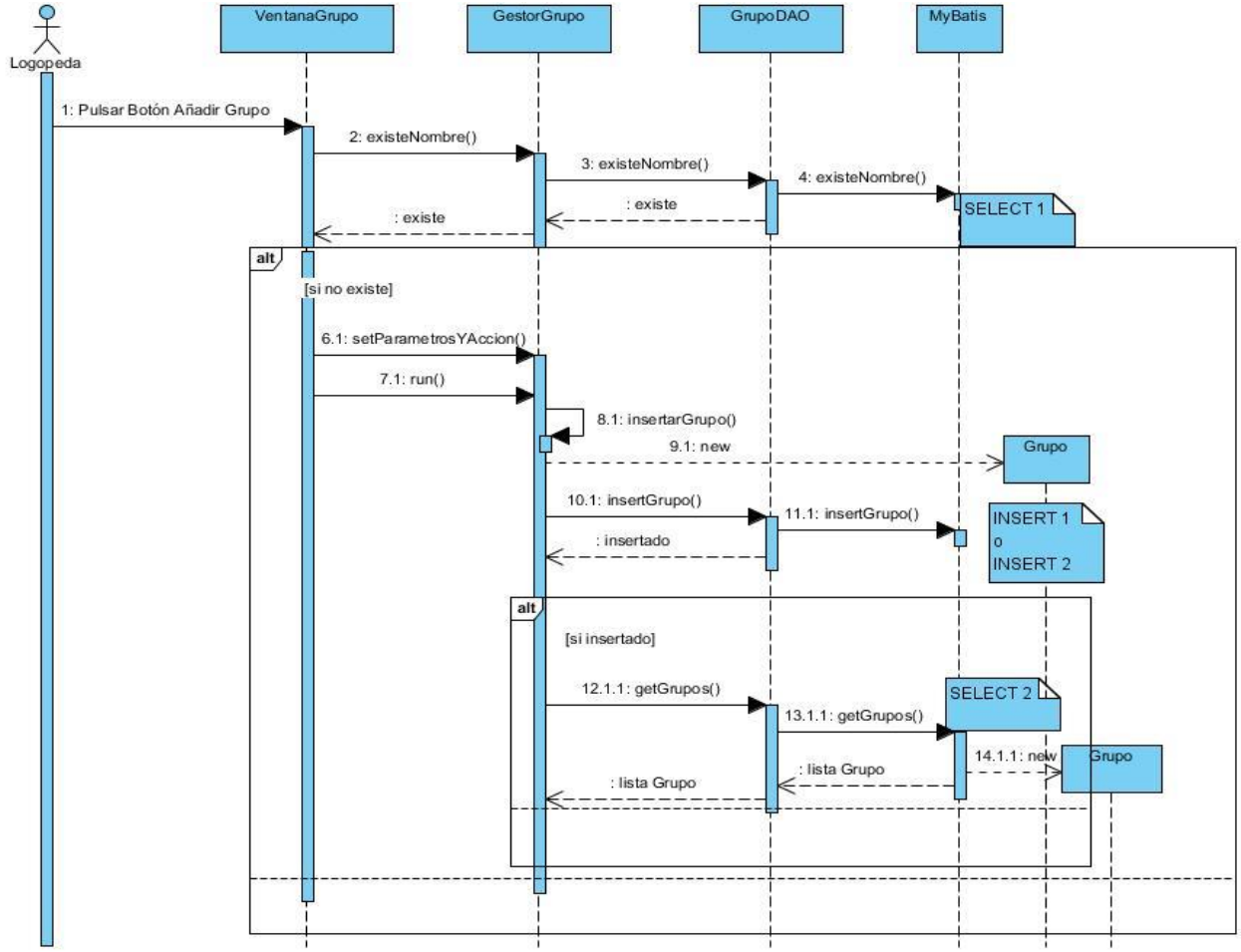


Figura 239: Diagramas de Secuencia Crear Grupo

La opción de elegir INSERT 1 o INSERT 2 dependerá de si se le pasa un id de grupo o no (si se le pasa, el grupo ya existe).

SELECT 1

```
SELECT COUNT(1)
FROM GRUPO
WHERE G_NOMBRE = RTRIM(#{nombre})
```

SELECT 2

```
SELECT G_ID_GRUPO, G_NOMBRE, G_CONTENIDO_SOLUCION, S.C_NOMBRE AS S_NOMBRE,
       S.C_CASTELLANO AS S_CASTELLANO, S.C_EUSKERA AS S_EUSKERA, S.C_IMAGEN AS
       S_IMAGEN,G_CONTENIDO_OPCION, O.C_NOMBRE AS O_NOMBRE, O.C_CASTELLANO AS
       O_CASTELLANO, O.C_EUSKERA AS O_EUSKERA, O.C_IMAGEN AS O_IMAGEN
FROM GRUPO
INNER JOIN CONTENIDO S ON G_CONTENIDO_SOLUCION = S.C_ID_CONTENIDO
INNER JOIN CONTENIDO O ON G_CONTENIDO_OPCION = O.C_ID_CONTENIDO
ORDER BY G_ID_GRUPO ASC
```

INSERT 1

```
INSERT INTO GRUPO
(G_ID_GRUPO, G_NOMBRE, G_CONTENIDO_SOLUCION, G_CONTENIDO_OPCION)
VALUES
(#{idGrupo}, RTRIM(#{nombre}), #{contenidoSolucion.idContenido},
#{contenidoOp.idContenido}) //Esto se hace mediante el foreach del xml
```

INSERT 2

```
INSERT INTO GRUPO
(G_ID_GRUPO, G_NOMBRE, G_CONTENIDO_SOLUCION, G_CONTENIDO_OPCION)
VALUES
(#{idGrupo}, RTRIM(#{nombre}), #{contenidoSolucion.idContenido},
#{contenidoOp.idContenido}) //Esto se hace mediante el foreach del xml
ON DUPLICATE KEY UPDATE
	G_NOMBRE= VALUES(G_NOMBRE),
	G_CONTENIDO_SOLUCION=VALUES(G_CONTENIDO_SOLUCION),
	G_CONTENIDO_OPCION=VALUES(G_CONTENIDO_OPCION)
```

EliminarGrupo

Debido a que son similares, en este apartado se referirá a la eliminación de grupos, permisos y de grupos de minijuegos.

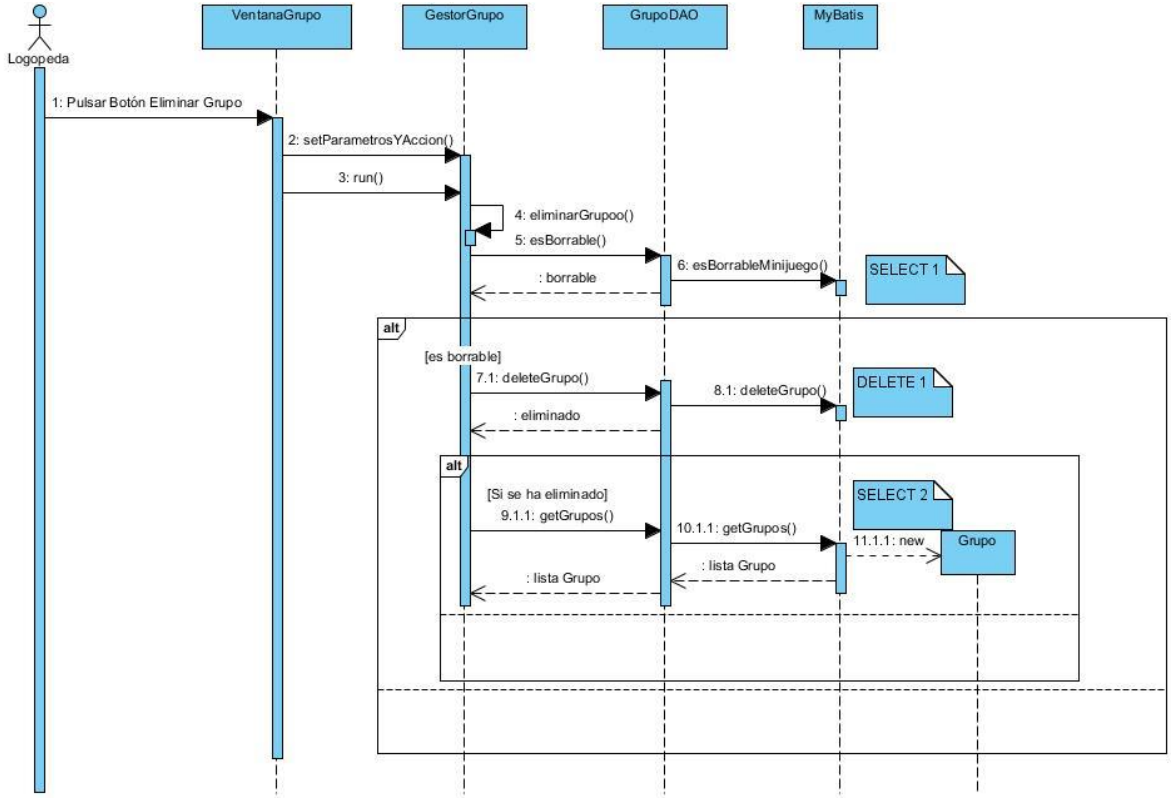


Figura 240: Diagrama de Secuencia EliminarGrupo

SELECT 1

```
SELECT COUNT(1)
FROM PERMISOS_GRUPOS
WHERE PG_ID_GRUPO = #{idGrupo}
```

SELECT 2

```
SELECT G_ID_GRUPO, G_NOMBRE, G_CONTENIDO_SOLUCION, S.C_NOMBRE AS S_NOMBRE,
       S.C_CASTELLANO AS S_CASTELLANO, S.C_EUSKERA AS S_EUSKERA, S.C_IMAGEN AS
       S_IMAGEN,G_CONTENIDO_OPCION, O.C_NOMBRE AS O_NOMBRE, O.C_CASTELLANO AS
       O_CASTELLANO, O.C_EUSKERA AS O_EUSKERA, O.C_IMAGEN AS O_IMAGEN
FROM GRUPO
INNER JOIN CONTENIDO S ON G_CONTENIDO_SOLUCION = S.C_ID_CONTENIDO
INNER JOIN CONTENIDO O ON G_CONTENIDO_OPCION = O.C_ID_CONTENIDO
ORDER BY G_ID_GRUPO ASC
```

DELETE 1

```
DELETE FROM GRUPO
WHERE G_ID_GRUPO = #{idGrupo}
  AND G_CONTENIDO_SOLUCION = #{solucion}
  AND G_CONTENIDO_OPCION = #{opcion}
```

Login Videojuego

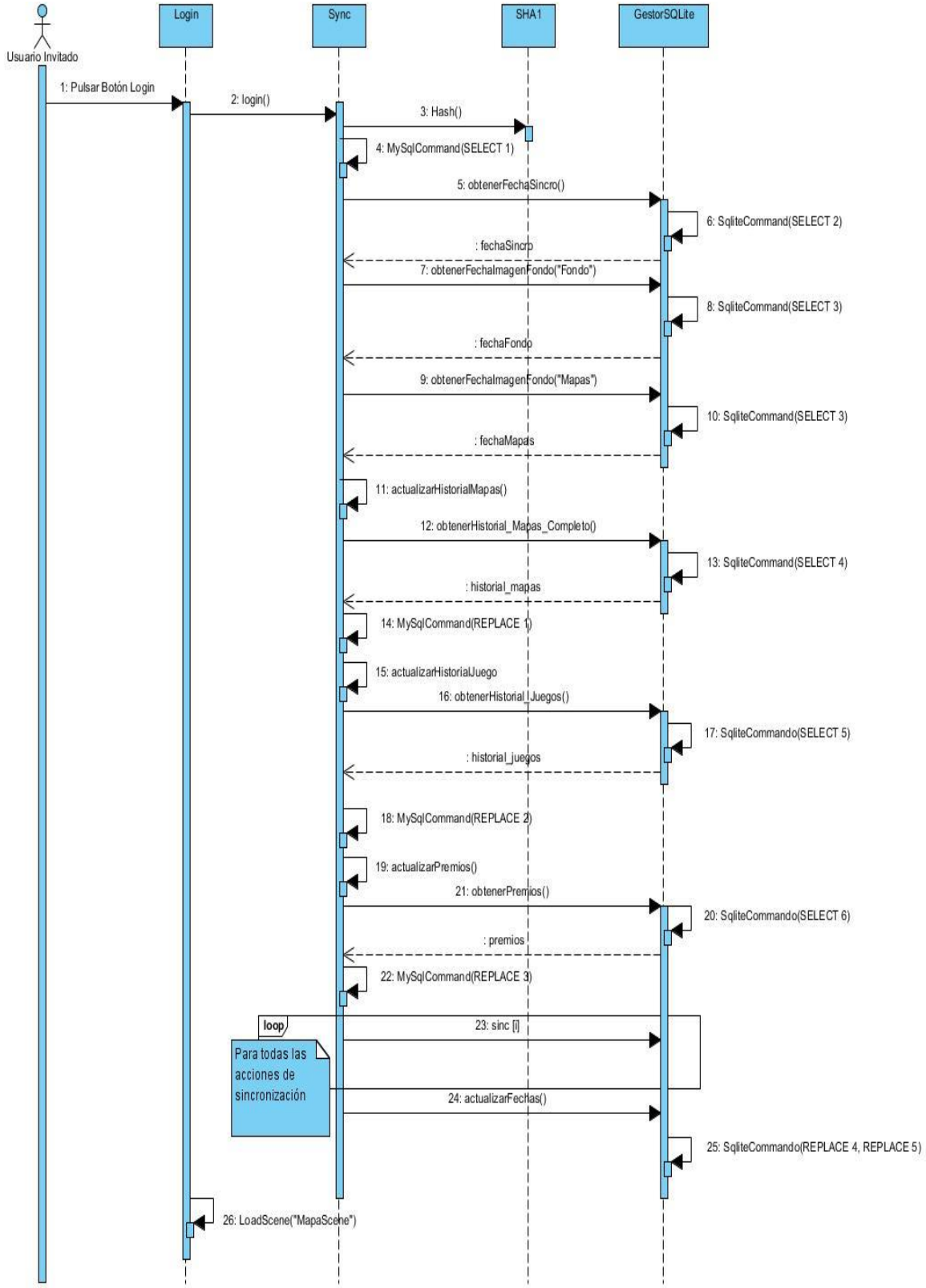


Figura 241: Diagrama de Secuencia Login Videojuego

SELECT 1

```
SELECT LOGIN(@nombre, @contrasena)
```

SELECT 2

```
SELECT DATETIME(U_ULT_SINCRO)
FROM USUARIO;
```

SELECT 3

```
SELECT TI_FECMOD
FROM TIEMPOS_IMAGENES
WHERE TI_NOMBRE_TABLA = :nombre_tabla;
```

SELECT 4

```
SELECT HM_ID_MAPA, HM_POS_MAPA, HM_ID_MINIJUEGO, HM_ID_BOTON, HM_ESTADO, HM_FECHA
FROM V_HISTORIAL_MAPAS
```

REPLACE 1

```
SINCRONIZAR_HISTORIAL_MAPA
```

SELECT 5

```
SELECT HJ_ID_MINIJUEGO, HJ_COMPLETO, HJ_VIDAS, HJ_ERRORES, HJ_TIEMPO, HJ_FECHA
FROM V_HISTORIAL_JUEGO
```

REPLACE 2

```
REPLACE INTO HISTORIAL_JUEGO
(HJ_ID_NINO, HJ_ID_MINIJUEGO, HJ_FECHA, HJ_COMPLETO, HJ_VIDAS, HJ_ERRORES, HJ_TIEMPO)
VALUES
(@id_usuario, @id_minijuego, @fecha, @completo, @vidas, @errores, @tiempo);
```

SELECT 6

```
SELECT PR_ID_PREMIO, PR_CONSEGUIDO
FROM PREMIO
WHERE PR_FECMOD >= (SELECT U_ULT_SINCRO FROM USUARIO)
```

REPLACE 3

```
UPDATE USUARIO
SET U_ULT_SINCRO = :fecha
WHERE U_ID_USUARIO = :id_usuario;
```

SELECT 7

Esta *select* dependerá de si es la primera sincronización del usuario o no. Por otro lado, también dependerá de si es la primera vez que se juega o no (por cualquier usuario)

REPLACE 4

```
REPLACE INTO PREMIOS_USUARIOS
(PU_ID_PREMIO, PU_ID_NINO, PU_CONSEGUIDO, PU_FECMOD)
VALUES
(@id_premio, @id_usuario, 'Si', SYSDATE());
```

REPLACE 4

```
INSERT OR REPLACE INTO TIEMPOS_IMAGENES
(TI_NOMBRE_TABLA, TI_FECMOD)
VALUES
(:nombre_tabla, :fecha);
```

Desconectar

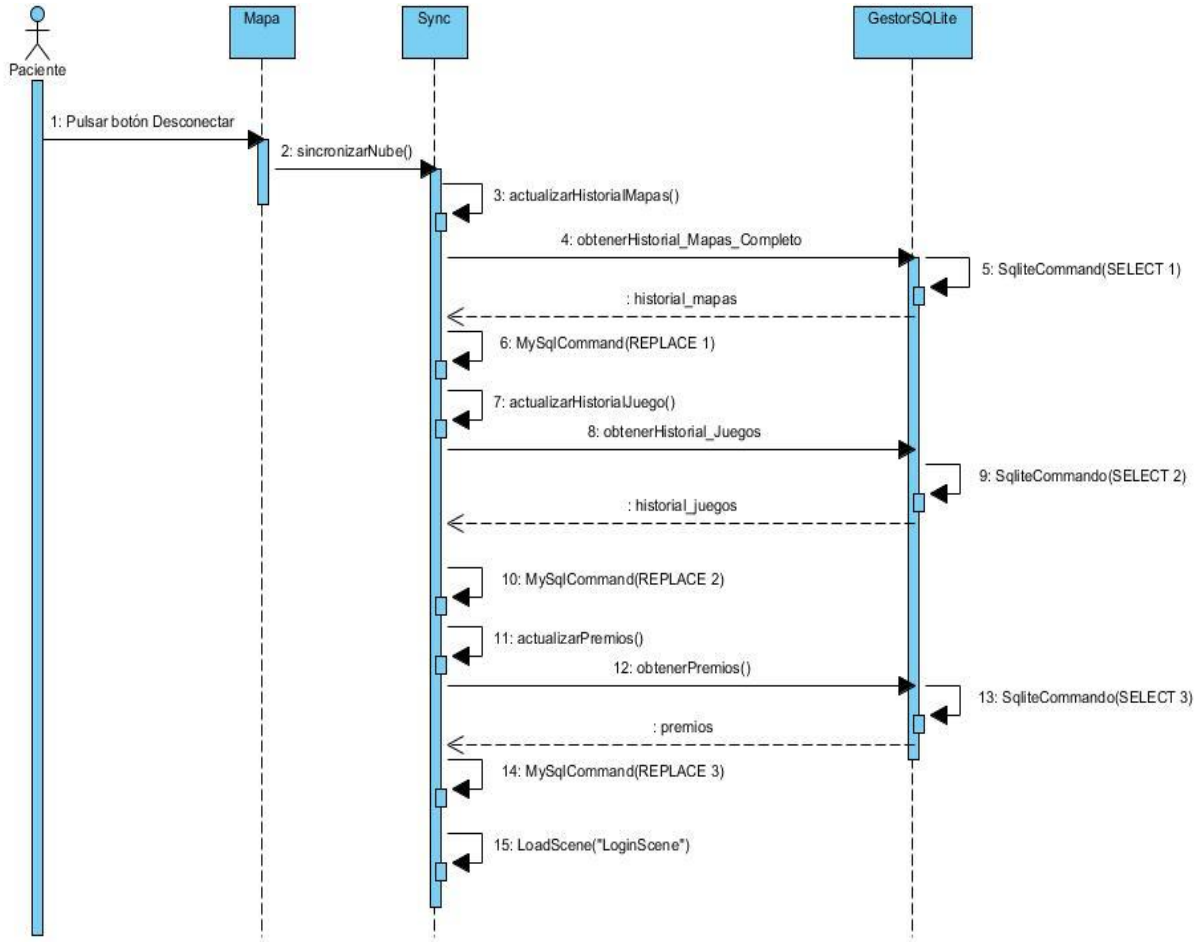


Figura 242: Diagrama de Secuencia Desconectar

SELECT 1

```
SELECT HM_ID_MAPA, HM_POS_MAPA, HM_ID_MINIJUEGO, HM_ID_BOTON, HM_ESTADO, HM_FECHA
FROM V_HISTORIAL_MAPAS
```

REPLACE 1

```
SINCRONIZAR_HISTORIAL_MAPA
```

SELECT 2

```
SELECT HJ_ID_MINIJUEGO, HJ_COMPLETO, HJ_VIDAS, HJ_ERRORES, HJ_TIEMPO, HJ_FECHA
FROM V_HISTORIAL_JUEGO
```

REPLACE 2

```
REPLACE INTO HISTORIAL_JUEGO
(HJ_ID_NINO, HJ_ID_MINIJUEGO, HJ_FECHA, HJ_COMPLETO, HJ_VIDAS, HJ_ERRORES, HJ_TIEMPO)
VALUES
(@id_usuario, @id_minijuego, @fecha, @completo, @vidas, @errores, @tiempo);
```

SELECT 3

```
SELECT PR_ID_PREMIO, PR_CONSEGUIDO
FROM PREMIO
WHERE PR_FECMOD >= (SELECT U_ULT_SINCRO FROM USUARIO)
```

REPLACE 3

```
REPLACE INTO HISTORIAL_JUEGO
(HJ_ID_NINO, HJ_ID_MINIJUEGO, HJ_FECHA, HJ_COMPLETO, HJ_VIDAS, HJ_ERRORES, HJ_TIEMPO)
VALUES
(@id_usuario, @id_minijuego, @fecha, @completo, @vidas, @errores, @tiempo);
```

TerminarJuego

A pesar de que este método es utilizado en cada minijuego, se utilizará la clase que lo implementa (*Minijuego*) para globalizar.

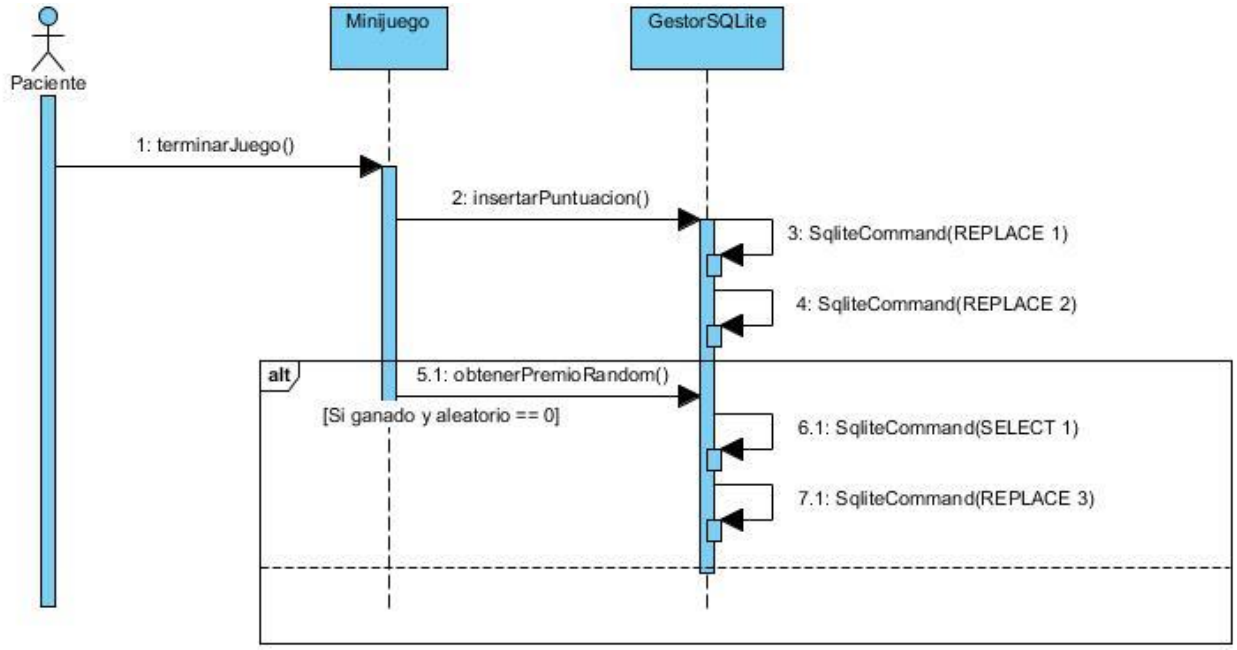


Figura 243: Diagrama de Secuencia terminarJuego

REPLACE 1

```
INSERT INTO HISTORIAL_JUEGO
(HJ_ID_MINIJUEGO, HJ_COMPLETO, HJ_VIDAS, HJ_ERRORES, HJ_TIEMPO, HJ_FECHA)
VALUES
(:id_minijuego, :completo, :vidas, :errores, :tiempo, DATETIME(CURRENT_TIMESTAMP,
'localtime'));
```

REPLACE 2

```
UPDATE HISTORIAL_MAPAS
SET HM_ESTADO = 'Pasado',
    HM_FECHA = DATETIME(CURRENT_TIMESTAMP, 'localtime')
WHERE HM_ID_MAPA = :id_mapa
  AND HM_POS_MAPA = :pos_mapa
  AND HM_ID_MINIJUEGO = :id_minijuego
  AND HM_ID_BOTON = :id_boton;
```

REPLACE 3

```
UPDATE PREMIO
SET PR_CONSEGUIDO = 'Si',
    PR_FECMOD = DATETIME(CURRENT_TIMESTAMP, 'localtime')
WHERE PR_ID_PREMIO = :id_premio;
```

SELECT 1

```
SELECT PR_ID_PREMIO, PR_IMAGEN
FROM PREMIO
ORDER BY RANDOM()
LIMIT 1;
```

Buscar

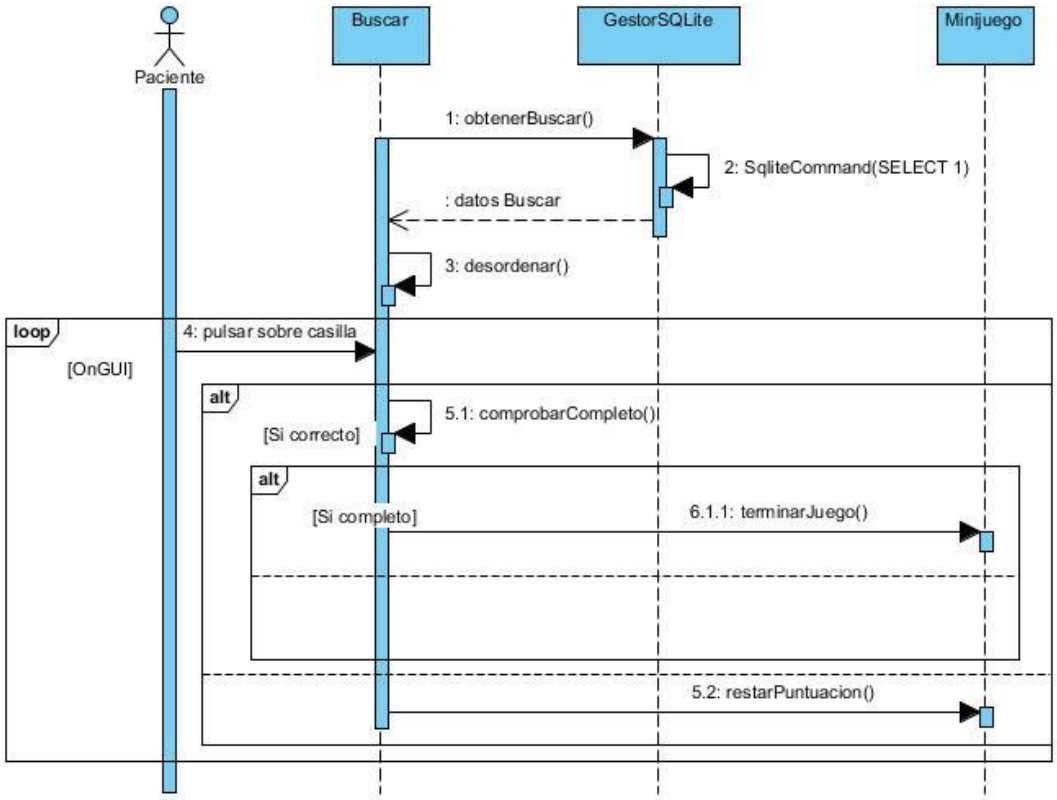


Figura 244: Diagrama de Secuencia Buscar

SELECT 1

```
SELECT TEXTO_SOLUCION, TEXTO_OPCION
FROM V_CONTENIDO
WHERE GM_ID_MINIJUEGO = :id_minijuego
  AND TEXTO_SOLUCION IS NOT NULL");
  AND TEXTO_OPCION NOT NULL
GROUP BY G_ID_GRUPO
ORDER BY RANDOM()
LIMIT 0,1;
```

Parejas

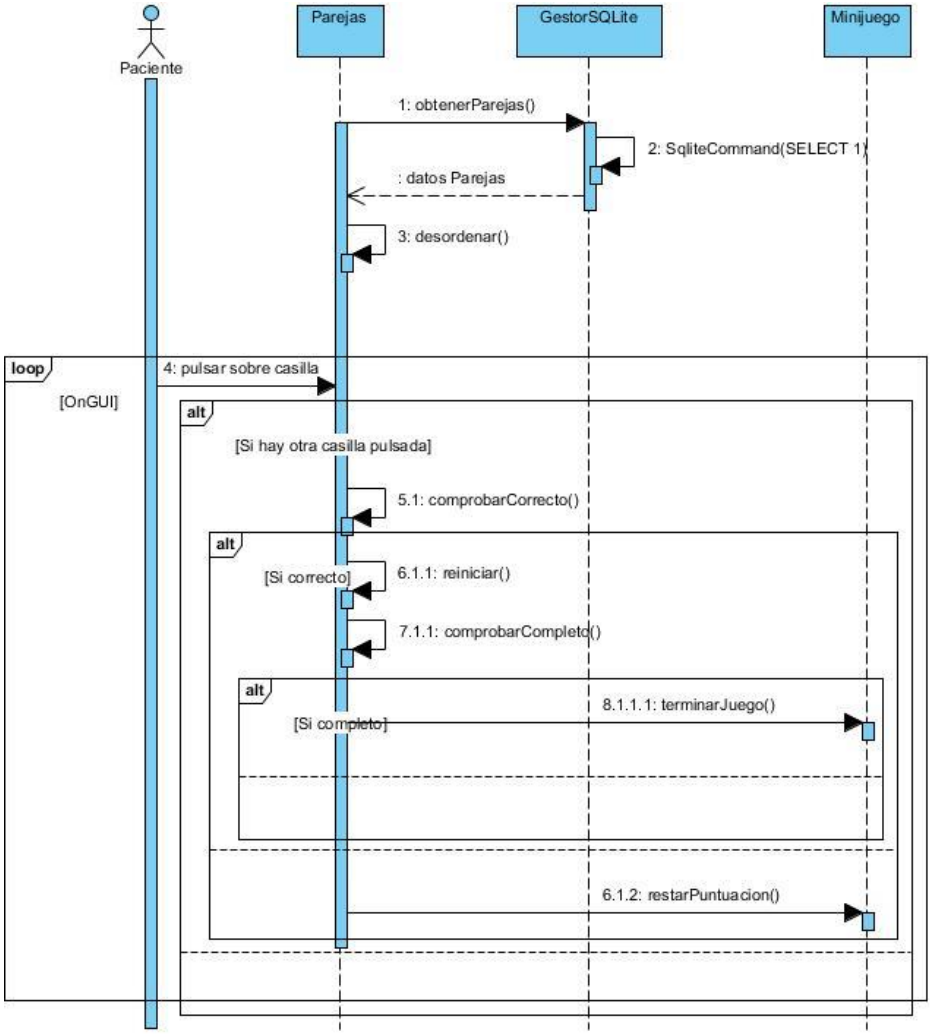


Figura 245: Diagrama de Secuencia Parejas

SELECT 1

```
SELECT TEXTO_SOLUCION, IMAGEN_SOLUCION
FROM V_CONTENIDO
WHERE GM_ID_MINIJUEGO = :id_minijuego
  AND TEXTO_SOLUCION IS NOT NULL
GROUP BY G_ID_GRUPO
ORDER BY RANDOM();
```

Palabras Incompletas

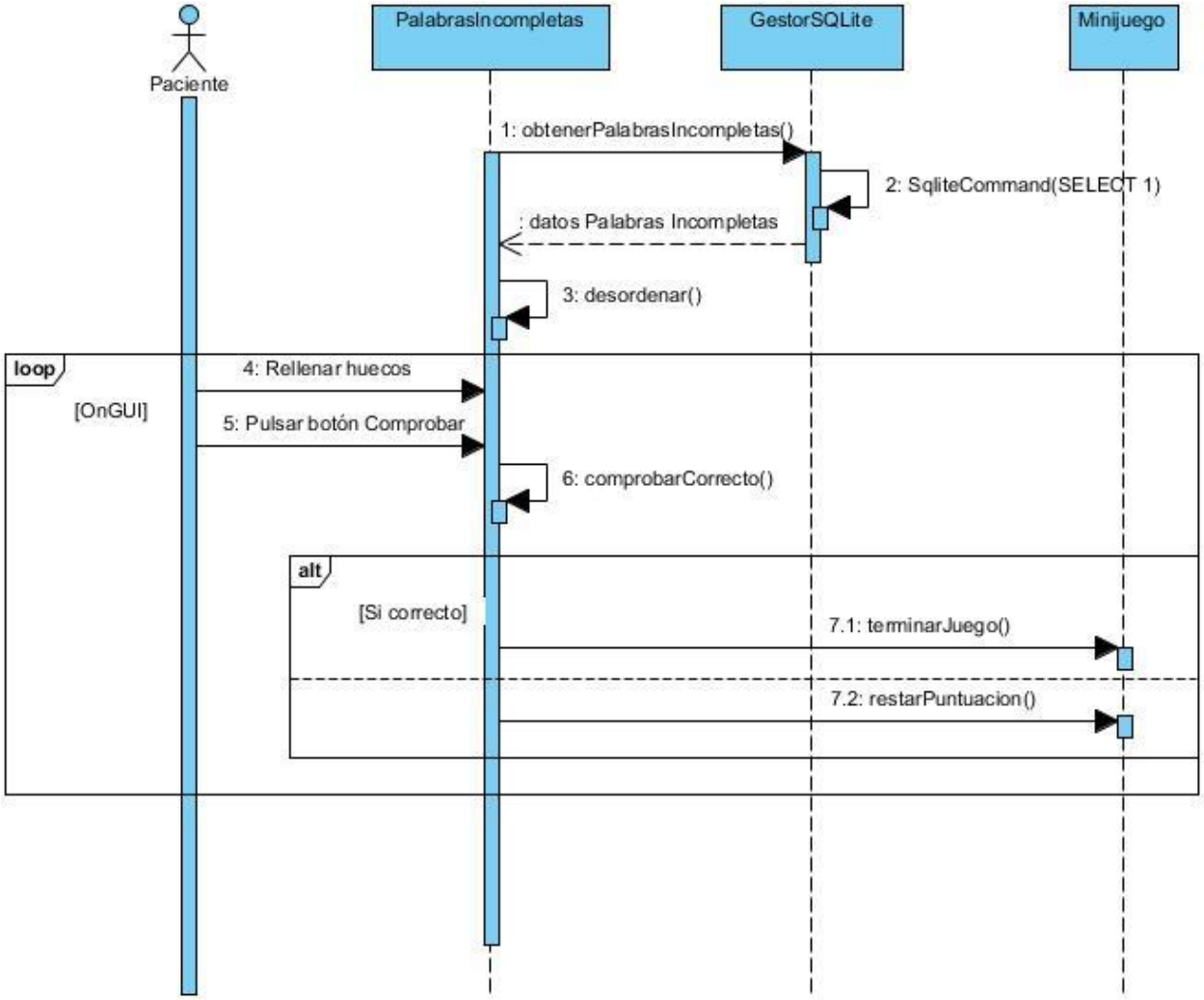


Figura 246: Diagrama de Secuencia Palabras Incompletas

SELECT 1

```
SELECT TEXTO_SOLUCION, IMAGEN_SOLUCION,
       (SELECT TEXTO_OPCION
       FROM V_CONTENIDO VC1
       WHERE VC1.G_ID_GRUPO = VC.G_ID_GRUPO ORDER BY RANDOM()) AS TEXTO_OPCION
FROM V_CONTENIDO VC
WHERE GM_ID_MINIJUEGO = :id_minijuego
  AND IMAGEN_SOLUCION IS NOT NULL
  AND TEXTO_SOLUCION IS NOT NULL
  AND TEXTO_OPCION IS NOT NULL
  AND TEXTO_OPCION LIKE '%\_%' ESCAPE '\'
GROUP BY G_ID_GRUPO
ORDER BY RANDOM();
```

Sopa de Letras

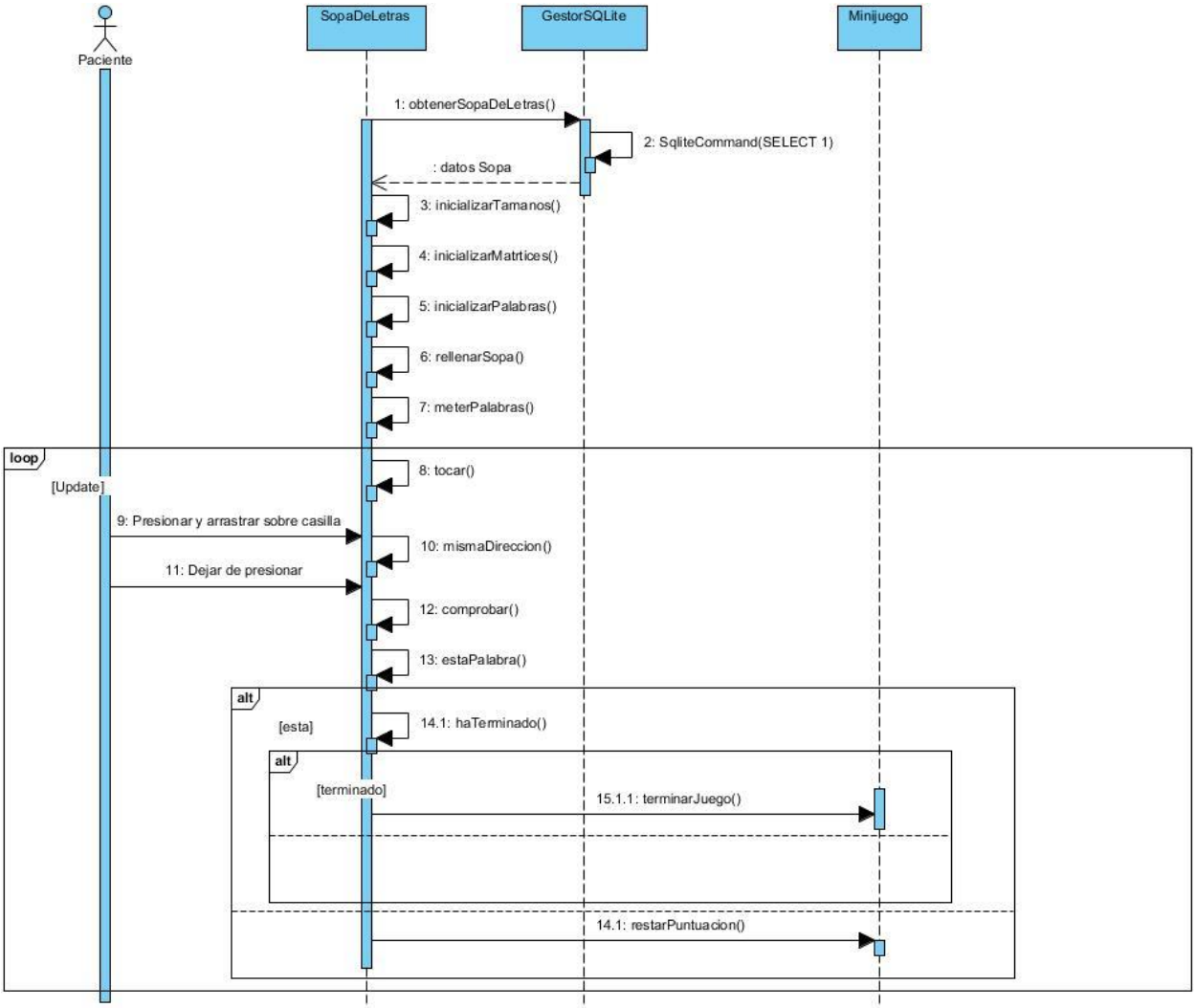


Figura 247: Diagrama de Secuencia Sopa de Letras

SELECT 1

```
SELECT TEXTO_SOLUCION, IMAGEN_SOLUCION
FROM V_CONTENIDO
WHERE GM_ID_MINIJUEGO = :id_minijuego
  AND TEXTO_SOLUCION IS NOT NULL
GROUP BY G_ID_GRUPO
ORDER BY RANDOM();
```

Laberinto Erróneo

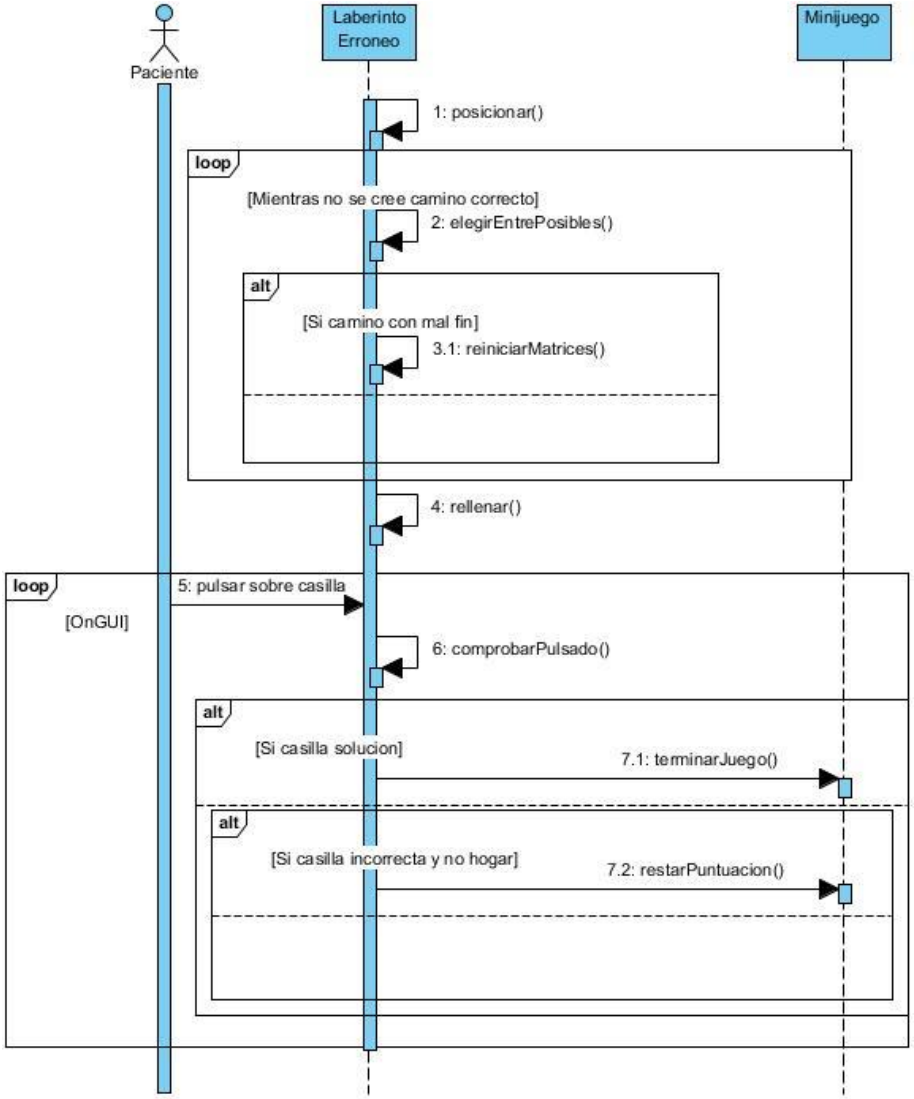


Figura 248: Diagrama de Secuencia Laberinto Erróneo

Seleccionar y Frase Correcta

En este caso, el diagrama se ha realizado sobre el minijuego *Seleccionar*. A pesar de esto, el diagrama de secuencia se refiere tanto al minijuego *Seleccionar* como *Frase Correcta*, que se han agrupado debido a la similitud en los diagramas (para no repetir).

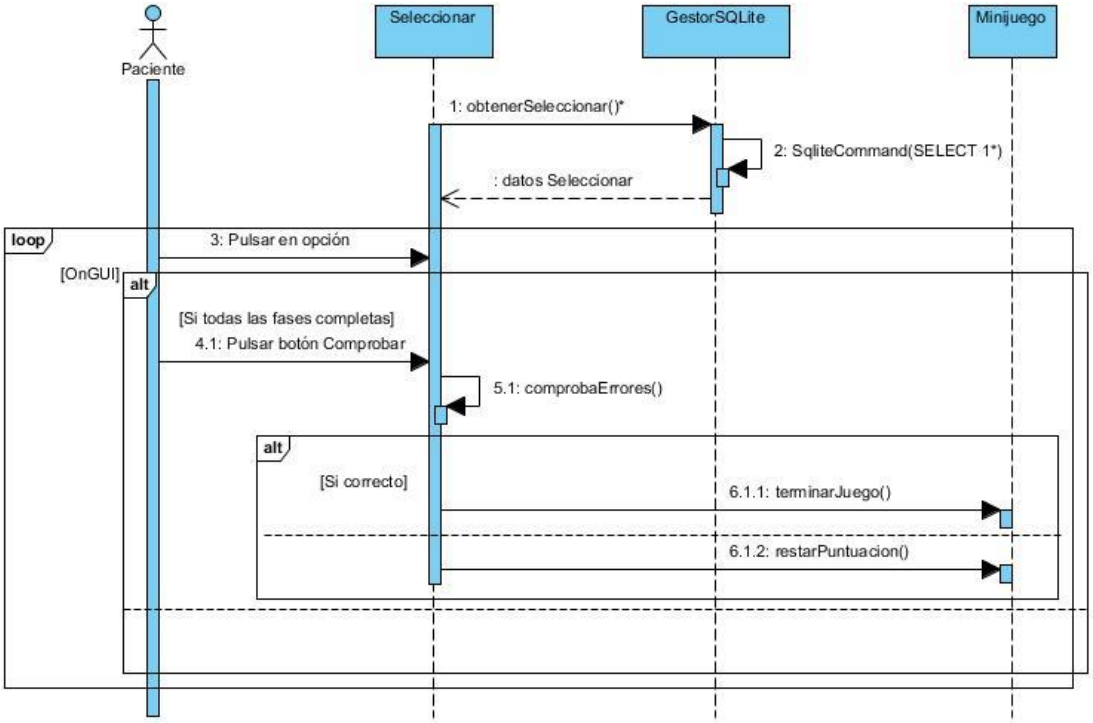


Figura 249: Diagrama de Secuencia Seleccionar

En el caso de la función 1 (*obtenerSeleccion()*), en caso del minijuego *FraseCorrecta* será *obtenerFraseCorrecta().*

SELECT 1

```
SELECT G_ID_GRUPO, ID_CONTENIDO_SOLUCION, TEXTO_SOLUCION, ID_CONTENIDO_OPCION,
      IMAGEN_OPCION
FROM V_CONTENIDO
WHERE GM_ID_MINIJUEGO = :id_minijuego
  AND TEXTO_SOLUCION IS NOT NULL
  AND IMAGEN_OPCION IS NOT NULL
ORDER BY RANDOM();
```