Universitat de les Illes Balears

Escuela politécnica superior

Memoria del trabajo de final de grado

Aplicación iOS para rehabilitación cervical

Antonio Arenas Serra

Grado en Ingeniería Informática

Año académico 2018-19

DNI del alumno: 43460007N

Trabajo tutelado por: Cristina Manresa Yee y Maria Francesca Roig Maimó

Departamento de Ciencias Matemáticas e Informática

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

Índice de contenidos

Índice de Ilustraciones

Índice de tablas

Acrónimos

UIB Universitat de les Illes Balears

UGIVIA Unidad de Gráficos, Visión por Computador e Inteligencia Artificial

IDE Integrated Development Environment

API Application Programming Interface

ADE Api Development Environment

OMS Organización Mundial de la Salud

REST Representational State Transfer

JSON JavaScript Object Notation

HTTP HyperText Transfer Protocol

JPA Java Persistance API

Resumen

En este trabajo de final de grado se presenta un videojuego para motivar a los pacientes que sufren dolores cervicales a realizar ejercicio terapéutico para aliviar el dolor e incrementar la movilidad de cuello.

A lo largo del documento, se describe el proceso de diseño y desarrollo de una aplicación que implementa un videojuego cuyo objetivo final va más allá del entretenimiento. Este videojuego ha sido desarrollado para dispositivos móviles con sistema operativo iOS. Durante el proceso, se ha colaborado con el Departamento de Enfermería y Fisioterapia de la Universitat de les Illes Balears (UIB) para analizar los requisitos de la aplicación y validar el videojuego para su objetivo final de rehabilitación cervical.

El videojuego diseñado está basado en la recolección de elementos (estrellas) pilotando una nave espacial, y la particularidad de este videojuego es la interacción que ofrece, pues la dirección de la nave se controla mediante el movimiento vertical de la cabeza. Los objetivos se desplazan horizontalmente en la pantalla del dispositivo e irán reduciendo su tamaño y aumentando la velocidad durante el transcurso del juego. Estas características, junto con otras como la posición inicial de la nave o las características de los objetivos a recoger (posición, tamaño, velocidad), cuentan con un nivel de personalización muy elevado para que los fisioterapeutas sean capaces de adaptar el videojuego a cada paciente que haga uso de la aplicación.

La interacción se lleva a cabo a través de una interfaz basada en visión proporcionada por la Unidad de Gráficos, Visión por Computador e Inteligencia Artificial (UGIVIA) de la UIB: el framework *"FaceMe"*. *"FaceMe"* proporciona la posición de la nariz del usuario en cada fotograma capturado por la cámara frontal del dispositivo. A partir de las coordenadas proporcionadas por *"FaceMe"*, puede realizarse el seguimiento de los movimientos de la cabeza del usuario y transformarlos en acciones interactivas en el juego.

1. Introducción

La cervicalgia inespecífica se define como un tipo de dolor cervical causado por alteraciones musculoesqueléticas no específicas, en ocasiones de manera repetida pudiendo incluso derivar en dolor crónico (dolores que perduran más de 3 meses). Este trastorno musculoesquelético es muy común en la sociedad moderna, y no solo afecta a la calidad de vida de quien lo sufre, sino que también afecta a nivel familiar y a nivel económico en el sistema sanitario [1 , 2].

Los tratamientos para tratar la cervicalgia son: los ejercicios terapéuticos, movilizaciones, terapia manual, aplicación de agentes electro físicos, etc. Estos tratamientos requieren de constancia, implicación y adherencia[1] por parte del paciente y pueden suponer desplazamientos, esperas o personal dedicado [3]. Además, la realización continua y repetitiva de los ejercicios terapéuticos aconsejados para que el paciente los realice de forma individual es difícil de conseguir.

El Trabajo Final de Grado trabaja en la línea de motivar al paciente a realizar estos ejercicios terapéuticos de forma individual. A continuación, se presenta el proyecto RHBC, un proyecto del Departamento de Ciencias Matemáticas e Informática y del Departamento de Enfermería y Fisioterapia, cuyo objetivo general es diseñar aplicaciones para dispositivos móviles que implementen juegos serios. Los juegos serios son todos aquellos que no solo se diseñan e implementan con un propósito lúdico.

1.1. Proyecto RHBC

Con todos los indicadores expuestos en la sección anterior, entre el Departamento de Ciencias Matemáticas e Informática y el Departamento de Enfermería y Fisioterapia de la UIB surge el proyecto RHBC, enfocado al desarrollo de sistemas interactivos implementados en dispositivos móviles para la rehabilitación de la zona cervical. En particular, se decide diseñar una plataforma que integre varios videojuegos personalizables a las necesidades de cada paciente para motivarle a realizar ejercicio

[1] La Organización Mundial para la Salud (OMS) define la adherencia al tratamiento como el cumplimiento del mismo.

terapéutico y que almacene los resultados del uso para poder analizar el progreso de los pacientes.

La sinergia entre estos dos departamentos de la UIB es necesaria para el diseño y el desarrollo del sistema. Por un lado, el Departamento de Enfermería y Fisioterapia define los requisitos del sistema y valida su uso con pacientes analizando su estado y su progreso. Por el otro lado, el Departamento de Ciencias Matemáticas e Informática se encarga del desarrollo de los videojuegos y los integra en la plataforma a la que accederán tanto los pacientes para jugar como los fisioterapeutas para configurar los videojuegos y analizar los resultados.

El proyecto RHBC, busca ofrecer diferentes videojuegos que cumplan las siguientes características:

- Complementar el ejercicio terapéutico.
- Extraer medidas clínicas de la evolución de los pacientes para ser analizadas.

Los móviles y tabletas actuales cuentan con una gran cantidad de sensores como cámaras o acelerómetros. Es por ello, que se propuso hacer uso de la cámara frontal de estos dispositivos, para capturar los movimientos de la cabeza del usuario y utilizar esa información para interactuar con los videojuegos. En el caso de este proyecto, los movimientos se capturan mediante la detección de la posición de la nariz del usuario en las imágenes proporcionadas por la cámara frontal del dispositivo.

El uso de los videojuegos como complemento terapéutico puede resolver varios de los problemas que pueden presentar los tratamientos clásicos. Tanto la motivación para la realización de los ejercicios como su adherencia se ven incrementados con el uso de videojuegos [4,5].

A continuación, se listan los objetivos del proyecto RHBC:

Objetivo general:

- **OG1**: Diseñar juegos serios para dispositivos móviles destinados al tratamiento de la cervicalgia que sirvan como herramienta terapéutica para el alivio del dolor, la mejora de la funcionalidad y de la calidad de vida.

Objetivos específicos:

- **OE1**: Definir un conjunto de movimientos/ejercicios para el tratamiento clínico que vaya a complementar.
- **OE2**: Diseñar y desarrollar un videojuego para dispositivos móviles que motive a los pacientes a realizar los ejercicios definidos en su casa.
- **OE3**: Validar técnicamente el videojuego como conjunto de ejercicios terapéuticos adecuado para la rehabilitación cervical.
- **OE4**: Comprobar mediante evaluación clínica con pacientes reales de la validez del tratamiento definido.
- **OE5**: Demostrar mediante evaluación clínica que el uso del videojuego con dispositivo móvil mejora la adherencia al tratamiento en casa.
- **OE6**: Demostrar que los parámetros recogidos por el videojuego permiten la obtención de datos objetivos sobre la evolución de los pacientes.

1.2. Objetivos del TFG

Este trabajo de fin de grado se enmarca en el contexto del proyecto RHBC, con los siguientes objetivos propios.

Objetivo General:

Generar un nuevo juego serio para dispositivos iOS totalmente integrado en el proyecto RHBC, a partir de los requisitos definidos por el Departamento de Enfermería y Fisioterapia.

Objetivos Específicos:

- **OE1:** Definir un conjunto de movimientos/ejercicios (parámetros configurables del juego) para complementar el tratamiento clínico junto con los fisioterapeutas.
- **OE2:** Diseñar y desarrollar un videojuego para dispositivos móviles que motive a los pacientes a realizar los ejercicios definidos en OE1.
- **OE3:** Ampliar el back end ya existente para satisfacer los requisitos de la nueva aplicación.
- **OE4:** Integrar el framework *"FaceMe"*, que proporciona la posición de la nariz del usuario, como parámetro de entrada en la aplicación.

Objetivos personales:

- **OE5:** Desarrollar una aplicación desde cero con un lenguaje nuevo (Swift).
- **OE6:** Ampliar conocimientos de Java y del framework Spring usado en el desarrollo del back end .

2. Plataforma RHBC

Como se ha comentado en el capítulo anterior, en el proyecto RHBC participan el Departamento de Ciencias Matemáticas e Informática y el Departamento de Enfermería y Fisioterapia. En la ilustración 1 se ha definido un esquema con los diferentes participantes, las tareas de estos y los productos resultantes del proyecto.

Los dos primeros bloques representan estos dos departamentos y las tareas que éstos realizan para cumplir los objetivos del proyecto.

En el tercer bloque, aparece uno de los productos resultantes de este proyecto, un back end común. Este back end está desarrollado con el lenguaje Java, proporcionará y almacenará la información de todas las aplicaciones que estén integradas en este proyecto. Actualmente da soporte a una aplicación ya existente (juego de las dianas) y la aplicación desarrollada en este TFG.

Por último, el cuarto bloque representa los usuarios finales que harán uso de las diferentes aplicaciones desarrolladas.

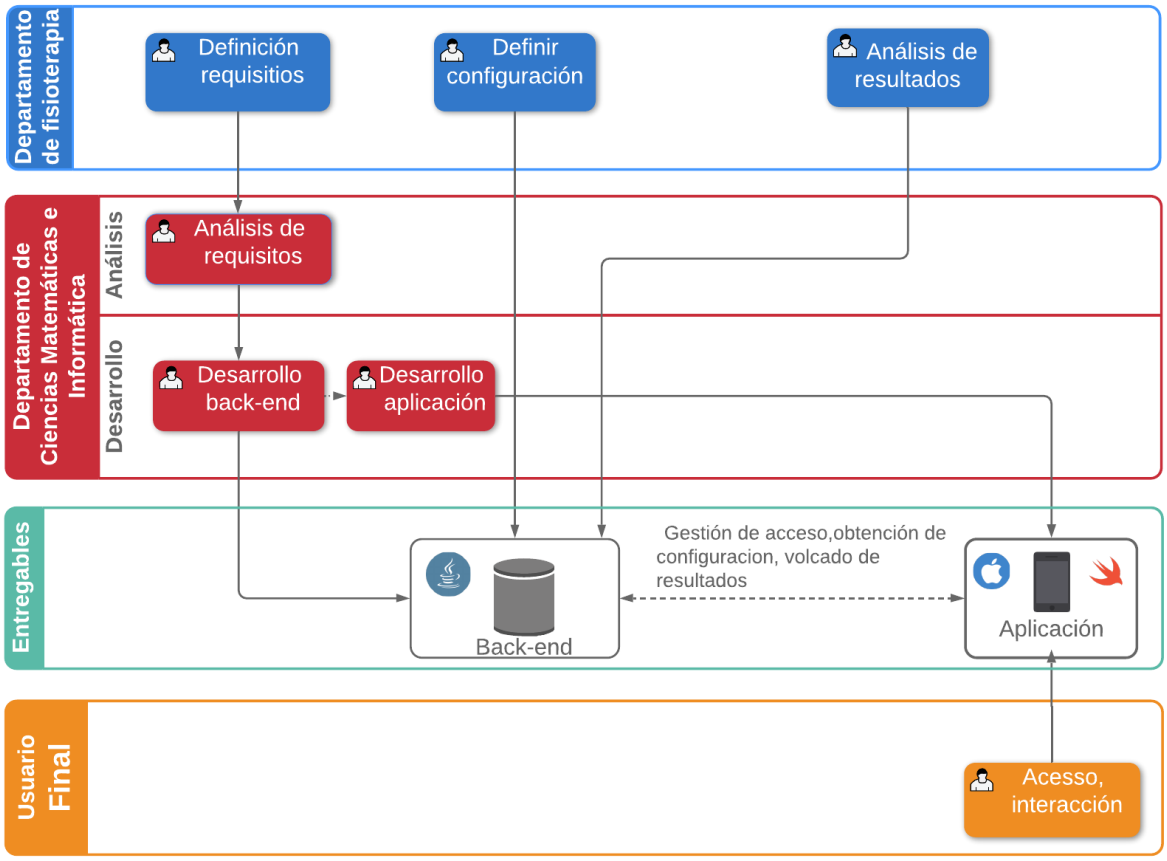

Ilustración 1. Esquema proyecto

Todos los videojuegos integrados dentro de la plataforma RHBC son accesibles para el usuario una vez introducidas y validadas las credenciales. En la ilustración 2 se puede ver el inicio de sesión, que hará uso de las funciones definidas en el back end, comprobando si el usuario y la contraseña han sido dados de alta previamente y, en caso afirmativo, recuperar el perfil del paciente y sus configuraciones personalizadas para cada uno de los videojuegos accesibles.

Actualmente ya existe un videojuego en la plataforma para entrenar los movimientos horizontales, verticales y diagonales, el juego de las dianas (ver ilustración 3). El videojuego utiliza el mismo sistema de interacción a través de los movimientos de la cabeza y el objetivo es mantener el cursor sobre las dianas durante periodos de tiempo que hayan sido definidos por los fisioterapeutas con el fin de aumentar el control cervical

del paciente, es decir, parámetros que se obtienen del back end y que han sido configurados previamente.

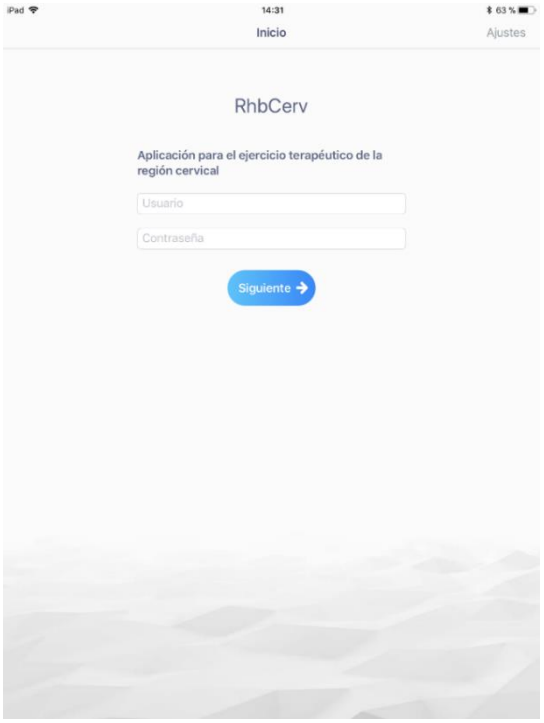

Ilustración 2. Aplicación RHBC - Inicio Sesión

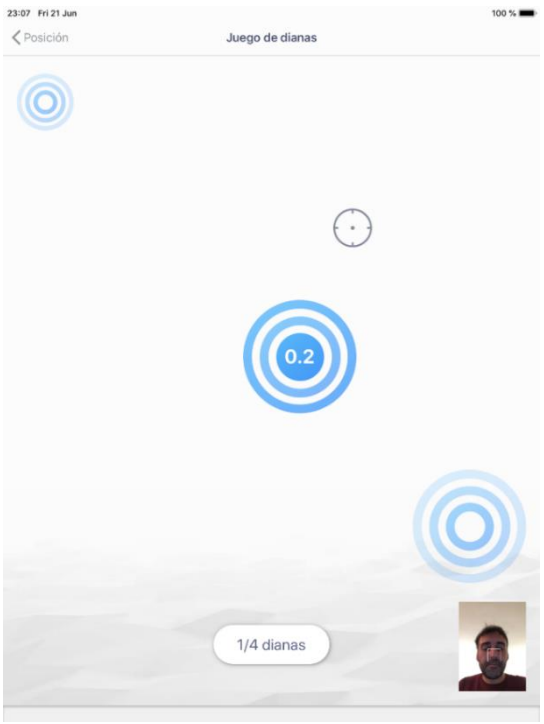

Ilustración 3. Juego Dianas - Interacción

3. Análisis

En el este capítulo se definen los requisitos para el nuevo videojuego y las ampliaciones que deberán realizarse en el back end para garantizar la coexistencia de la nueva aplicación en la plataforma existente. Los requisitos del back end se separarán en: aquellos necesarios para la implementación de los web services (funciones accesibles vía web desde otra plataforma, en nuestro caso la aplicación) y los propios del back end.

Los requisitos aquí descritos han sido identificados mediante reuniones con los tutores, los fisioterapeutas del Departamento de Enfermería y Fisioterapia y el equipo encargado del desarrollo del proyecto RHBC.

Se ha definido la siguiente nomenclatura para listar todos los requisitos:

- R.F.A.X, requisito funcional aplicación más identificador numérico.
- R.N.A.X, requisito no funcional aplicación más identificador numérico.
- R.F.B.X, requisito funcional back end más identificador numérico.
- R.F.WS.X, requisito funcional web service ofrecido por el back end más identificador numérico.
- R.N.B.X, requisito no funcional back end más identificador numérico.

3.1. Requisitos funcionales aplicación

Identificador Requisito	Descripción
R.F.A 1	Control acceso, notificar credenciales incorrectas o permitir el acceso a la aplicación.
R.F.A 2	Obtener la configuración para el usuario que haya introducido las credenciales de acceso al juego.
R.F.A 3	Capturar los movimientos de la cabeza para trasladarlos a posiciones de la pantalla haciendo uso del framework "*"FaceMe"*".

R.F.A 4	Gestión estado del juego, pausado, reanudar, game over, juego superado.
R.F.A 5	Visualización de puntos, enseñar en todo momento los puntos disponibles y actualizar sus valores.
R.F.A 6	Visualización de vidas, enseñar en todo momento las vidas disponibles y actualizar sus valores.
R.F.A 7	Sistema de niveles

Tabla 1. Requisitos funcionales videojuego

3.2. Requisitos no funcionales aplicación

Identificador Requisito	Descripción
R.N.A. 1	Lenguaje de implementación Swift o Objective-C para poder integrar "*"FaceMe"*"
R.N.A. 2	Soporte para múltiples idiomas
R.N.A. 3	Diseño adaptativo, mantener la interfaz del videojuego para los diferentes dispositivos iOS (Apple *iPhone*, Apple iPad)

Tabla 2. Requisitos no funcionales videojuego

3.3. Requisitos back end

Identificador Requisito	Descripción
R.F.B. 1	Nueva tabla para la configuración de los usuarios en el juego.
R.F.B. 2	Nueva tabla para almacenar los resultados.
R.F.B. 3	Permitir crear una configuración (username, gain, yship, xship, speed, starsize, totalstars, levelup, lives, nextstar[2]) por usuario para este videojuego
R.F.WS. 1	Dado un usuario devolver la configuración definida para este.
R.F.WS. 2	Guardar los resultados (success, timestamp, level, score, lives, fails[3]) obtenidos por el usuario en la base de datos del back end

Tabla 3. Requisitos funcionales back end

3.4. Requisitos no funcionales back end

Identificador Requisito	Descripción
R.N.B.1	Lenguaje de implementación Java y el framework Spring para poder integrarlo en el back end existente.
R.N.B.2	Uso del motor de base de datos H2 para mantener la compatibilidad con la base de datos ya existente.

Tabla 4. Tabla Requisitos no funcionales back end

[2] Estos parámetros se detallan en la sección del diseño de las tablas de la base de datos

[3] Estos parámetros se detallan en la sección del diseño de las tablas de la base de datos

4. Diseño

4.1. Videojuego propuesto

Debido a que los fisioterapeutas sólo deseaban entrenar los movimientos cervicales en el eje vertical, se propone el diseño de un videojuego centrado en el control de una nave espacial cuyos únicos movimientos permitidos sean en el eje y, es decir, movimientos verticales.

Dicha nave será controlada por el usuario a través de los movimientos de la cabeza con el objetivo de recoger todos los objetivos (estrellas) que vayan apareciendo en pantalla con un número máximo de fallos que fijarán los fisioterapeutas por parámetro en función del perfil del usuario.

Para añadir dinamismo al videojuego, a media que se van capturando los objetivos se subirá de nivel. Al subir de nivel, el tamaño de los objetivos será reducido y su velocidad se verá aumentada.

Esta recolección de objetivos será una experiencia totalmente diferente en función del usuario que la realice, ya que, para cada usuario, los fisioterapeutas pueden definir una configuración acorde al paciente que afectará a la hora de interactuar con la aplicación. A continuación, se añade una pequeña tabla informativa (ver tabla 5) en la que se pueden consultar los parámetros configurables de la aplicación, en el apartado 4.3 se pueden ver detallados todos estos parámetros.

Parámetro	Tipo de Dato
Id	Numérico
Username	Cadena de caracteres
Gain	Numérico
Yship	Numérico

Xship	Numérico
Ystar	Numérico
Speed	Numérico
StarSize	Numérico
TotalStars**.**	Numérico
LevelUp	Numérico
NextStar	Numérico
Lives	Numérico

Tabla 5. Parámetros Configurables

4.2. Diseño flujo videojuego

A partir de los requisitos obtenidos, se presenta el siguiente flujo (ver ilustración 4) para el videojuego, donde se ven identificadas las vistas que serán necesarias, así como las diferentes operaciones que se realizarán entre la transición de estas. En nuestro caso, definimos *tratamiento* como el conjunto de los diferentes parámetros configurados para un usuario (paciente) que haga uso de la aplicación.

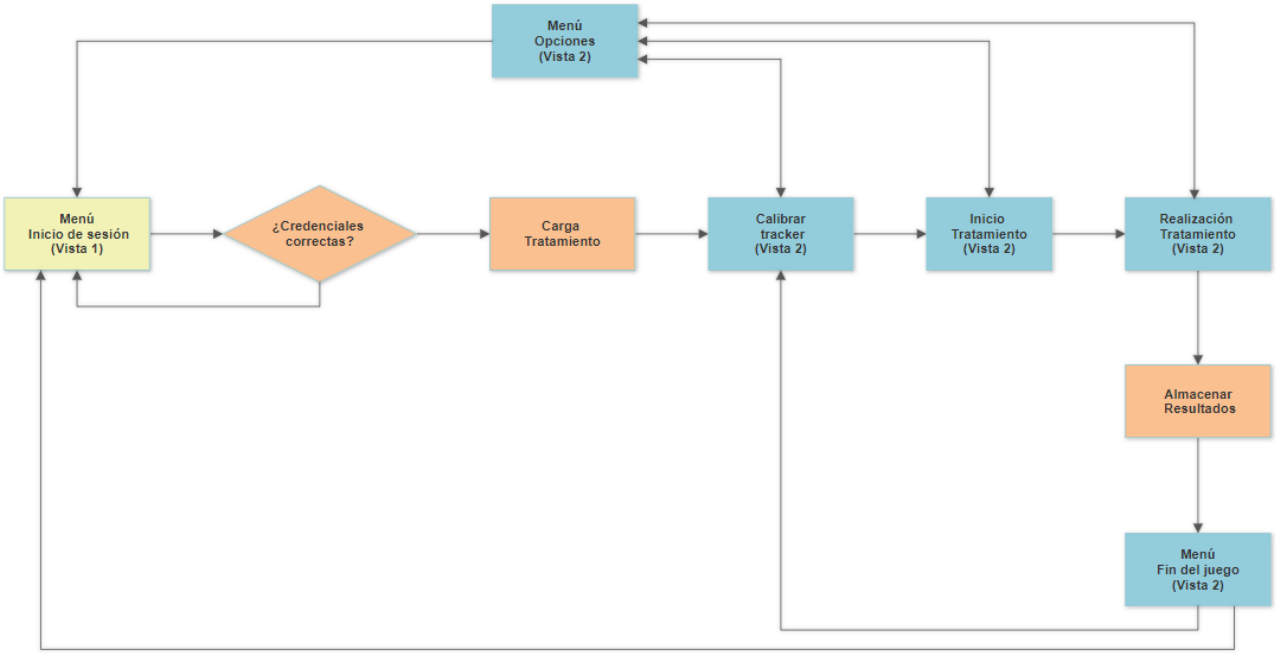

Ilustración 4. Diagrama Flujo Vistas

4.3. Diseño interfaz gráfica aplicación

La vista principal al ejecutar la aplicación presenta el formulario clásico de inicio de sesión para acceder a la aplicación, compuesta por dos componentes que recogerán el nombre de usuario y la contraseña proporcionadas al paciente. Por último, se dispone de un botón que permitirá acceder a la aplicación con la configuración del paciente si las credenciales son correctas (ver Ilustración 5).

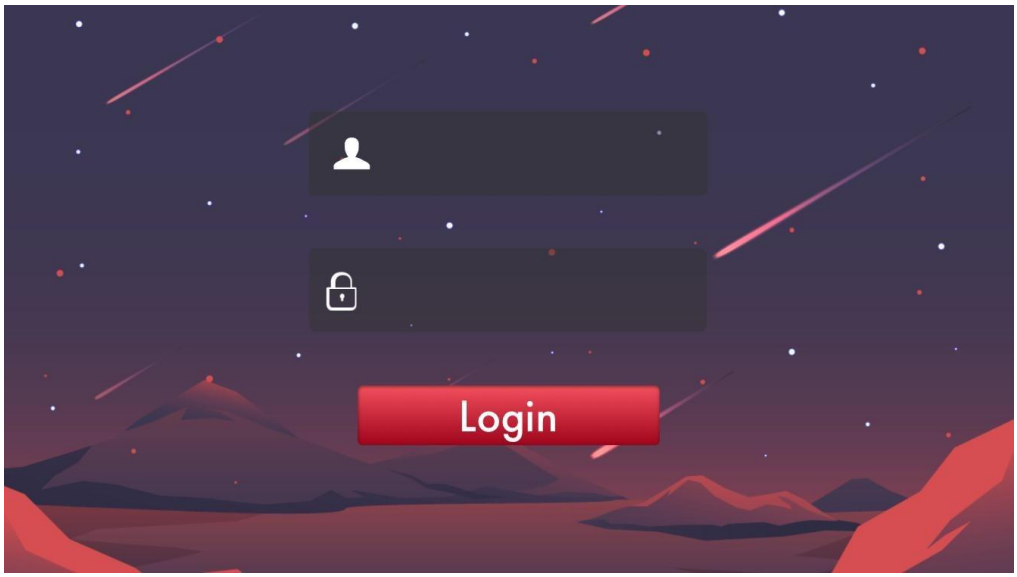

Ilustración 5. Interfaz Inicio de sesión

La siguiente vista, es la vista principal donde se ejecutará el juego (ver Ilustración 6). La dividiremos en 2 secciones para explicar su contenido.

En la sección superior, están definidas las vidas actuales (número de objetivos que puede perder un usuario) del usuario, el nivel en el que se encuentra y la puntuación total obtenida. Todos estos valores se irán actualizando a medida que el paciente progrese en el juego.

En la sección inferior derecha, encontramos el botón con forma engranaje que servirá para pausar el juego y enseñar las diferentes opciones del menú (ver ilustraciones 7 y 8).

- **Restart:** Se reiniciará y se cargará de nuevo el tratamiento del usuario que ha accedido a la aplicación.
- **Exit**: Saldremos a la vista principal desconectando al usuario actual de la aplicación dejando otra vez la vista de inicio de sesión. En caso de finalización del juego, ya sea porque ha conseguido recoger todos los objetivos y aún dispone de vidas o bien por no haber completado el juego, aparecerá un nuevo botón central con la opción de reiniciar; este botón será de color rojo, en caso de que haya perdido todas las vidas de las que disponía, o bien de color verde, indicando el éxito.

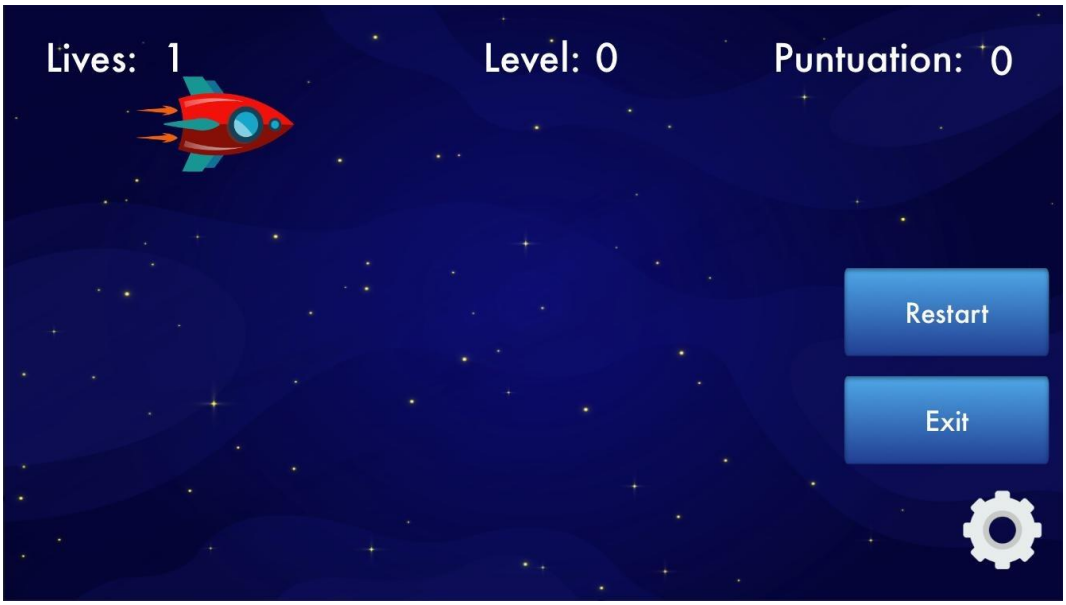

Ilustración 6. Interfaz Menú pausado

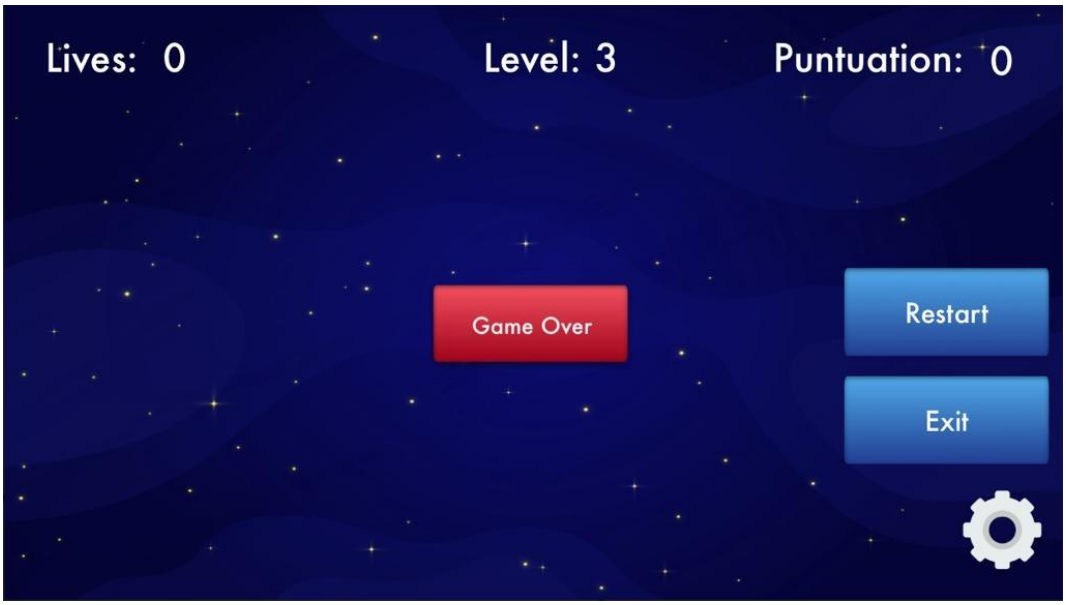

Ilustración 7. Interfaz: Menú fin del juego

El resto de la vista será el espacio utilizado por la nave, de la cual el usuario tendrá el control vertical, y el espacio en el que los objetivos irán apareciendo para ser recogidos, ver ilustración 8. Recordemos que la nave siempre estará fijada en el eje x.

Ilustración 8. Interfaz: Interacción con objetivos

4.4. Diseño sistema de coordenadas

Para ser capaz de gestionar las posiciones de los objetivos (estrellas) y la posición del jugador (nave) se ha diseñado el siguiente sistema de coordenadas (ver ilustración 9). Se divide la altura y anchura de la pantalla del dispositivo que esté ejecutando la aplicación para obtener cinco secciones. De esta manera se obtienen las posibles posiciones de inicio de los objetivos y la nave.

El origen de este sistema de coordenadas está fijado en la parte inferior izquierda. Siendo la coordenada (x,y: 0,0) la posición inferior izquierda y la coordenada (x,y: 5,5) la posición superior derecha.

Según el parámetro de configuración "invert" detallado en la sección 4.3, si invert=0, el lanzamiento de estrellas tiene su origen en el eje x=5. Por otro lado, si invert=1 el origen de las estrellas será x=0. Con esto se resuelve uno de los requisitos de la aplicación era que la captura de objetivos se hiciese tanto de derecha a izquierda como de izquierda a derecha.

Por otro lado, la nave del jugador siempre queda fijada en el eje x durante el transcurso del juego, tomando como valor aquel que venga definido por parámetro.

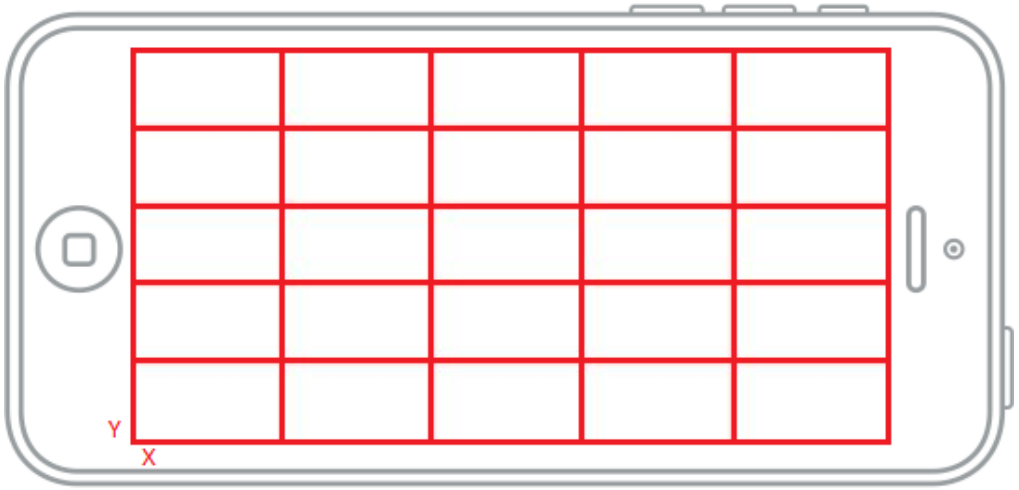

Ilustración 9. Sistema de Coordenadas

4.5. Traducción de posiciones

A continuación, se detallan las diferentes operaciones necesarias para realizar el seguimiento de los movimientos del jugador a partir de los valores proporcionado por el TrackingManager del framework "*"FaceMe'"*". Con estas operaciones seremos capaces de actualizar la posición de la nave en la pantalla del dispositivo en función de los movimientos de la cabeza del usuario.

El primer paso, es identificar el recorrido (R) que ha realizado el usuario a partir de las posiciones (x, y) retornadas por el framework en dos fotogramas consecutivos. Para ello se resta la posición previa (retornada en la iteración anterior) a la posición actual.

$$R = Posicion\ Actual - Posicion\ Previa$$

El segundo paso consiste en escalar el desplazamiento en función de la dimensión de la pantalla del dispositivo. Se comprobó que, para una imagen de la cámara de 192x144 pixeles (imagen de baja resolución de un Apple *iPhone* 7), los usuarios podían alcanzar un rango de movimiento de 55 píxeles tanto en el eje vertical como en el eje horizontal. Por lo tanto, para que los usuarios puedan alcanzar los límites inferiores y superiores de la pantalla sin realizar sobreesfuerzos, se hace uso del siguiente factor de escalado para el eje y (FE):

$$\text{FE} = \frac{\text{altura dispositivo}}{55}$$

El último paso para la traducción del desplazamiento es el factor de ganancia (FG), este factor de ganancia como se ha visto en el apartado 4.3 es configurable por los fisioterapeutas. El factor de ganancia representa la cantidad de movimiento sobre el dispositivo (movimiento de la nave) en respuesta a un movimiento unitario del usuario en la imagen de la cámara. El factor de ganancia es uno de los parámetros configurables por parte de los fisioterapeutas y, por lo tanto, llega por parámetro a la aplicación (todos los parámetros configurables de la aplicación se detallan en el siguiente apartado de este capítulo). Con este factor se puede facilitar la movilidad al usuario si se usan valores mayores a 1, o bien, requerir más esfuerzo si este factor tiene valores inferiores a 1.

Con todos estos valores, somos capaces de calcular la nueva posición de la nave (y) en el eje y, esta nueva posición se calcula mediante la siguiente operación [4]:

$$y = yActual - (R * FE * FG)$$

4.6. Diseño modelo de datos back end

Para el diseño de la base de datos ha sido necesario ampliar el modelo ya existente del proyecto RHBC. Para ello, se han creado dos tablas nuevas relacionadas con la tabla paciente, la cual ya estaba definida en el modelo, se puede consultar en la ilustración 10.

La primera tabla, *"Config_TFG",* definirá el tratamiento del paciente, es decir los parámetros que recibirá la aplicación para cargar su tratamiento. Relación 1 a 1, solo se almacenará una configuración por usuario.

La segunda tabla*, "Results_TFG"*, almacenará los resultados obtenidos una vez se haya realizado el tratamiento. Relación 1 a n, de manera que se almacenen todos los resultados de un usuario.

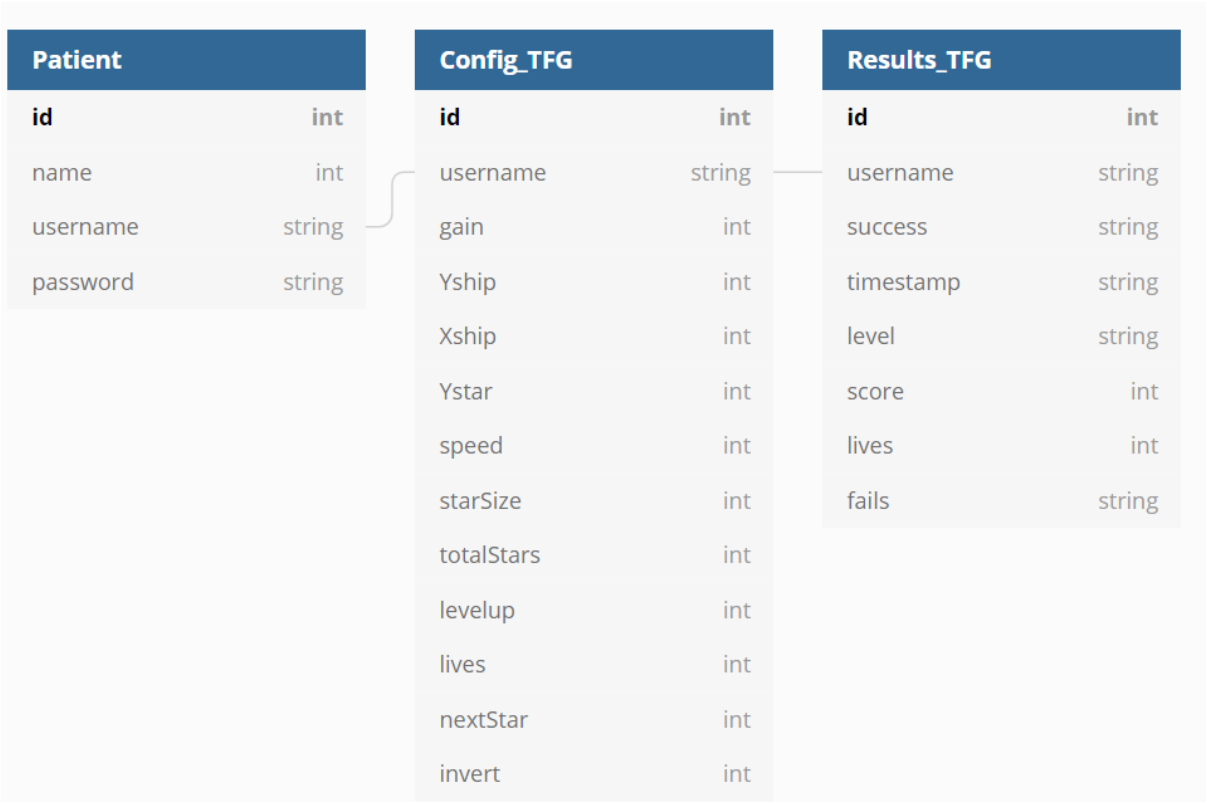

Ilustración 10. Diseño Base de datos

Los campos de la tabla "*config_TFG*" se detallan a continuación, ver ejemplo en la ilustración 11.

- **Id**: Número entero para identificar el tratamiento, clave primaria.

- **Username**: Clave extranjera con el nombre del usuario que tendrá esta configuración.
- **Gain**: Factor de ganancia (FG) para el eje vertical. Un valor más elevado implica menos esfuerzo por parte del usuario, es decir, mayor recorrido de la nave en respuesta al movimiento realizado por el usuario; un valor menor implica un mayor movimiento por parte del usuario para provocar el mismo desplazamiento en la nave. Se puede ver en detalle el uso de este parámetro en la sección 4.5.
- **Yship**: Posición inicial de la nave en el eje vertical de la pantalla. Los valores aceptados por este parámetro son los definidos en el sistema de coordenadas diseñado en la sección 4.4 (1,2,3,4,5).
- **Xship**: Posición inicial de la nave en el eje horizontal de la pantalla y que se mantendrá durante todo el juego. Los valores aceptados por este parámetro son los definidos en el sistema de coordenadas (1,2,3,4,5)
- **Ystar**: Posición inicial de la estrella (objetivo) en el eje vertical de la pantalla. Los valores aceptados por este parámetro son los definidos en el sistema de coordenadas (1,2,3,4,5).
- **Speed**: Tiempo que tarda un objetivo en realizar su recorrido, es decir, tiempo desde que aparece en pantalla y desaparece en el lado contrario. Este parámetro se mide en segundos.
- **StarSize**: Tamaño de los objetivos en pixeles.
- **TotalStars**: Total de objetivos que aparecerán en pantalla para ser capturados.
- **LevelUp**: Con este entero se define cuantos objetivos son necesarios para subir de nivel. Al subir de nivel, se aumentará la velocidad de las estrellas y se reducirá el tamaño de estas.
- **NextStar**: Este campo define el tiempo de aparición en segundos entre objetivos. Este parámetro se mide en segundos.
- **Lives**: Total de vidas de las que dispone el usuario para completar el juego.
- **Invert**: Si se configura con valor 1, cambiará la orientación del juego. Por defecto, la nave se sitúa en la parte izquierda de la pantalla y los objetivos avanzan de derecha a izquierda; con este campo a -1, se realizará la inversión: con las estrellas avanzando de izquierda a derecha y la nave posicionada a la derecha de la pantalla.

ID	XSHIP	YSHIP	YSTAR	GAIN	INVERT	LEVELUP	NEXT_STAR	SPEED	STAR_SIZE	TOTAL_STARS	USERNAME	VIDAS
1	1	1	2	1.0	0	4	1	4	30	10	toni	3
2	4	4	2	1.0	1	4	0	4	90	10	iosune	3

(2 rows, 0 ms)

Ilustración 11. Ejemplo Configuración

Detalles de los campos en *"Results_tfg",* ver ejemplo en ilustración 12.

- **Id**: Llave primaria, número entero.
- **Username**: Llave extranjera con el nombre del usuario que ha generado este resultado.
- **Success**: Campo que almacenará si ha completado el ejercicio con éxito, es decir, no se ha quedado sin vidas y han aparecido todos los objetivos.
- **TimeStamp**: Instante de tiempo formado por la hora y la fecha en la que se ha realizado el ejercicio.
- **Level**: Nivel al que ha llegado el usuario.
- **Score**: Número de objetivos conseguidos al finalizar ejercicio.
- **Lives**: Número de vidas restantes al finalizar el ejercicio.
- **Fails**: Se almacenarán todos los objetivos que no hayan sido capturados por el jugador, así como la posición en el eje vertical de estos objetivos.

ID	FALLOS	LEVEL	SCORE	SUPERADO	TIMESTAMP	USERNAME	VIDAS
1		2	10	si	2019-05-25 12:57:36	toni	3
2	Pos:2,Pos:2,Pos:3,	1	0	NO	2019-05-25 12:58:33	toni	0

(2 rows, 5 ms)

Ilustración 12. Ejemplo Resultados

A continuación, se adjuntan una comparativa de dos configuraciones, en las que se puede ver como la nave aparece fijada en distintos valores del eje x y tanto el número de vidas como el tamaño de las estrellas también son diferentes. (ver ilustración 13).

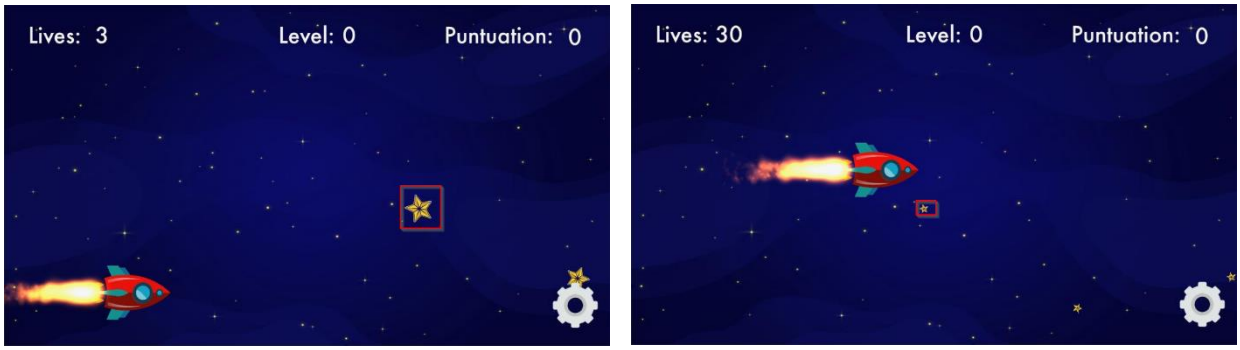

Ilustración 13. Comparativa Configuraciones

Se puede ver esta comparativa con más detalle en el anexo (A.2) donde también se pueden consultar las configuraciones en formato JSON y su representación en la aplicación.

5. Tecnología e Infraestructura

5.1 Lenguajes

El siguiente apartado contiene la información referente a los lenguajes de programación utilizados para llevar a cabo el desarrollo de la aplicación y del back end.

5.1.1. Swift

Para el desarrollo de aplicaciones iOS con Xcode, los dos lenguajes disponibles son Objective-C y Swift [6], la opción escogida ha sido Swift ya que presenta una sintaxis más simple y es más similar a los lenguajes orientados a objetos, como por ejemplo Java y C#, en los que ya contaba con experiencia.

Swift es un lenguaje multiparadigma y compilado desarrollado por Apple Inc para el desarrollo de aplicaciones tanto iOS como macOS, lanzado en 2014.

5.1.2. Java

El proyecto ya contaba con un back end existente y uno de los objetivos de este TFG era integrar el juego desarrollado dentro del proyecto RHBC. Para ello, se ha ampliado el proyecto del back end haciendo uso del lenguaje con el que ya había sido implementado, Java [7].

Java es un lenguaje compilado, orientado a objetos y concurrente cuyo objetivo es reducir el número de dependencias ganando así en compatibilidad, siendo necesaria una sola compilación para diferentes plataformas.

5.2. Herramientas utilizadas

A continuación, se detallan las herramientas utilizadas durante todo el desarrollo de este TFG.

5.2.1. Xcode

Xcode [8] es un entorno de desarrollo gratuito ofrecido por Apple y pensado para ejecutarse en macOS, que permite la creación de aplicaciones nativas para iOS, WatchOS, etc.

Xcode puede presentar problemas de compatibilidad en función de su versión y la versión de iOS para la que se quiere desarrollar. En este caso, ya que el material disponible era un iPhone 7 con la versión iOS 12.1, era necesario el SDK (Kit de desarrollo de software) iOS 12.1 y, para ello, se instaló la versión 10.1 de Xcode.

5.2.2. Eclipse

El IDE seleccionado por los desarrolladores del proyecto para el back end fue Eclipse [9], por lo que las nuevas funcionalidades del back end también se añadieron haciendo uso de Eclipse para evitar problemas de configuración y compatibilidades.

5.2.3. Postman

Postman [10] es una aplicación diseñada para el testeo de APIs, este tipo de aplicaciones se denominan ADE, es decir, permite generar llamadas a los servicios generados con diferentes parámetros y ver las respuestas de estas sin necesidad de crear un entorno específico para ser comprobado.

Esto es muy útil durante el desarrollo de nuestros servicios, puesto que se pueden hacer llamadas a estos sin necesidad de ejecutar la aplicación en nuestro dispositivo. Para ello, siempre se han probado primero desde Postman para verificar el resultado y, posteriormente, implementar estas operaciones desde la aplicación.

5.2.4. Sketch

Sketch [11] es un editor de gráficos vectoriales para macOS, el cual sido muy útil a la hora de generar el diseño gráfico de la aplicación. Una vez diseñada la interfaz, solo ha sido necesaria la exportación de los diferentes componentes a formato .PNG para su integración en el proyecto de Xcode.

5.3. Frameworks

5.3.1. FaceMe

Uno de los objetivos principales de la aplicación era ofrecer una manera de interactuar al paciente a través de los movimientos de la cabeza y monitorizarlos para ayudarle en su rehabilitación cervical. Para ello, esta interacción se lleva a cabo a través de una interfaz basada en visión proporcionada por la Unidad de Gráficos, Visión por Computador e Inteligencia Artificial (UGIVIA) de la UIB a través del *framework "FaceMe"*.[4]

"FaceMe" detecta y sigue los movimientos de la cabeza a través de los fotogramas que nos proporciona la cámara frontal del dispositivo. *"FaceMe"* identifica la cara del usuario, selecciona características sobre ambos lados de la nariz que son buenas para realizar su seguimiento (por ejemplo, fosas nasales o bordes) y, finalmente, calcula la media de todas esas características para obtener un único punto centrado sobre la nariz del usuario. Se puede ver el proceso de este algoritmo en detalle en la ilustración 14. Este punto proporcionado por *"FaceMe"* será el utilizado para gestionar los movimientos del jugador en la aplicación.

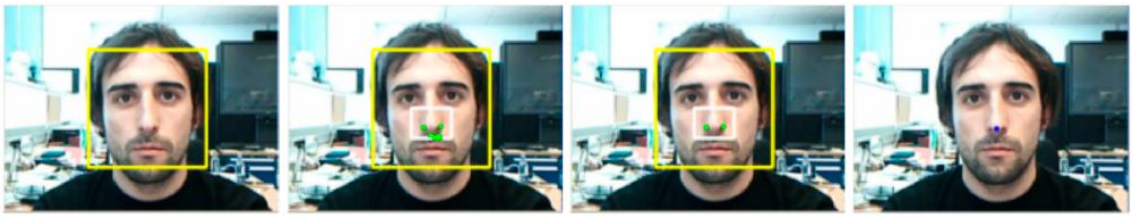

Ilustración 14. (a) Identificación cara; (b) Identificación características; (c) Identificación de características simétricas (d) Punto centrado.

Con el objetivo de traducir la posición retornada por *"FaceMe"* en cada fotograma, aparecen los conceptos de escalado y ganancia que se han detallado en la sección 4.3.

5.3.2. Spritekit

Spritekit es un *framework* desarrollado por Apple que ofrece la posibilidad de añadir contenido 2D a las aplicaciones de iOS, por ejemplo, partículas, texto, imágenes, video y efectos; así como la gestión de eventos para todos estos elementos citados previamente [12].

5.3.3. H2

El sistema administrador de la base de datos elegido sido H2, actualmente este entorno no ha sido desplegado y por lo tanto siempre se ejecuta en memoria una vez se ha desplegado el back end. Este tipo de ejecución se trata de una solución temporal que ofrece mayor flexibilidad durante el desarrollo ya que el conjunto de tablas y las relaciones finales del proyecto aún pueden sufrir cambios durante el desarrollo.

5.3.4. Spring

Spring es un framework que se utiliza para el desarrollo de aplicaciones, es de código abierto y utiliza Java como lenguaje de programación. Originalmente fue desarrollado para la plataforma J2EE de Java dicha plataforma estaba orientada al desarrollo de

aplicaciones web. Actualmente Spring se ha convertido un framework modular, pues la compañía detrás de Spring ofrece diferentes frameworks en función de las necesidades del desarrollador. Uno de ellos y que además ha sido utilizado en este proyecto ha sido, SpringBoot que permite la construcción de aplicaciones de manera rápida y sencilla, integrando las características que seleccione el desarrollador, en nuestro caso H2 y Spring [13,14].

6. Implementación

6.1. Implementación back end

6.1.1. Servicio web

Los servicios Web son todos aquellos sistemas de sistemas software con la finalidad de proporcionar interacción entre máquinas sobre una red. Los servicios Web suelen ser APIs Web que pueden ser accedidas dentro de una red y que se ejecutan en el servidor en el que se encuentran alojadas [15].

Una de las características principales de los servicios web, es su disponibilidad en la web y para ello debe hacer uso del protocolo estándar de transporte HTTP, detallado en la sección 6.1.2.

En nuestro caso, se realizó una implementación basada en REST, esta arquitectura genera una API haciendo uso de los diferentes métodos del protocolo HTTP (GET, POST...) permitiendo así interactuar a las aplicaciones con la aplicación principal que ofrece el servicio web [16].

6.1.2. Protocolo HTTP

El protocolo HTTP es probablemente el protocolo más utilizado en internet. La información que obtendrá la aplicación y la información que esta devolverá al back end será mediante las diferentes operaciones que proporciona el protocolo HTTP.

El esquema general que sigue dicho protocolo y el implementado para este proyecto se puede ver en la siguientes ilustraciones 15,16 [17].

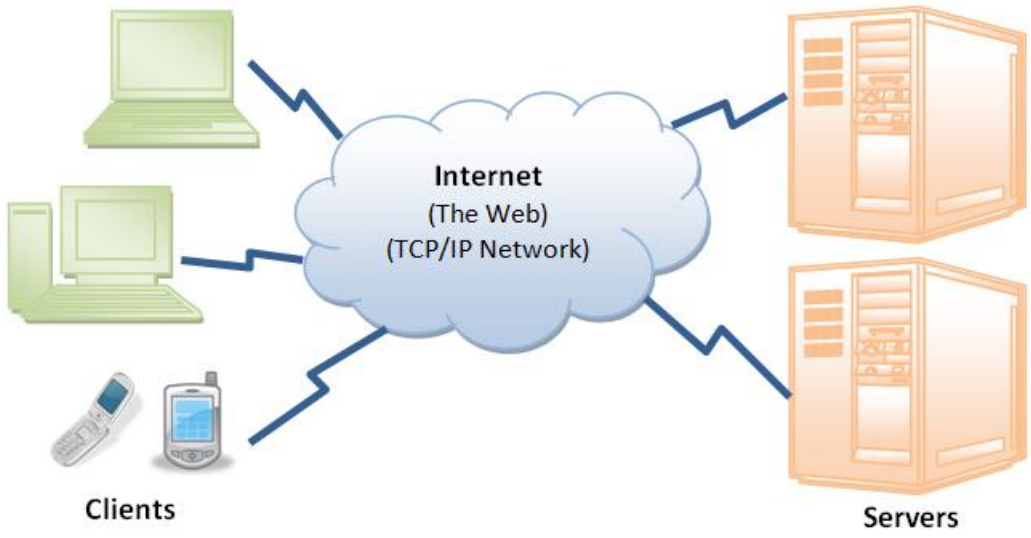

Ilustración 15. Esquema Red [17]

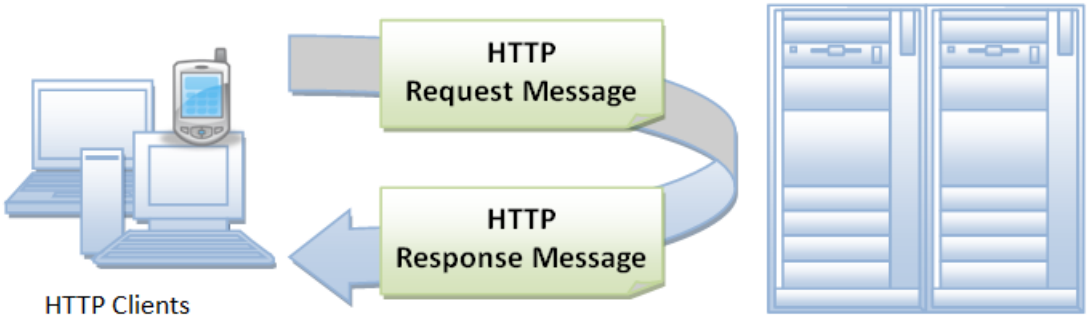

Ilustración 16. Esquema peticiones HTTP [17]

Existen diferentes tipos de operaciones disponibles (GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, PATCH). Estas se clasifican en dos categorías: seguras e idempotentes. Las operaciones seguras son aquellas utilizadas para recuperar información mientras que las operaciones idempotentes son aquellas que independientemente del número de ejecuciones deben generar el mismo resultado y mantener el mismo estado en el servidor [18].

Las operaciones más comunes son:

- GET
    - Método para adquirir información/recurso del servidor.
- POST
    - Método para enviar información/recurso al servidor

- DELETE
    - Método para eliminar información/recurso del servidor.

Para este TFG, solo ha sido necesaria la implementación de los métodos GET y POST.

6.1.3. Arquitectura

La arquitectura de este back end está formada por una base de datos, que se carga en memoria (H2) al ejecutar el back end, y la API web implementada con Java (SpringBoot) que permite realizar las diferentes operaciones sobre nuestras tablas a través del protocolo HTTP[15][16].

Para añadir el soporte del back end a este TFG ha sido necesario: creación de las nuevas tablas, inserción de registros en las nuevas tablas y añadir a la API las nuevas operaciones: una para realizar la obtención de la configuración y otra para guardar los resultados.

6.1.4. Implementación

Para las ampliaciones del back end existente ha sido necesario mantener la estructura de este., la estructura está formada por los siguientes elementos:

- La clase principal, también llamada entidad, que define a la tabla.
- Un interfaz, que será el repositorio de la clase
- Una clase de servicios que implementará las funcionalidades.
- Controlador, encargado de gestionar las peticiones HTTP.

Para gestionar la base de datos Spring hace uso del framework de Java JPA que permite establecer relaciones entre una base de datos relacional y los objetos que definamos.

En la ilustración 17 se pueden observar todas las clases nuevas que han sido creadas a excepción de la clase initialLoader, pues era una clase existente que se encarga introducir datos en la base de datos al desplegar el back end.

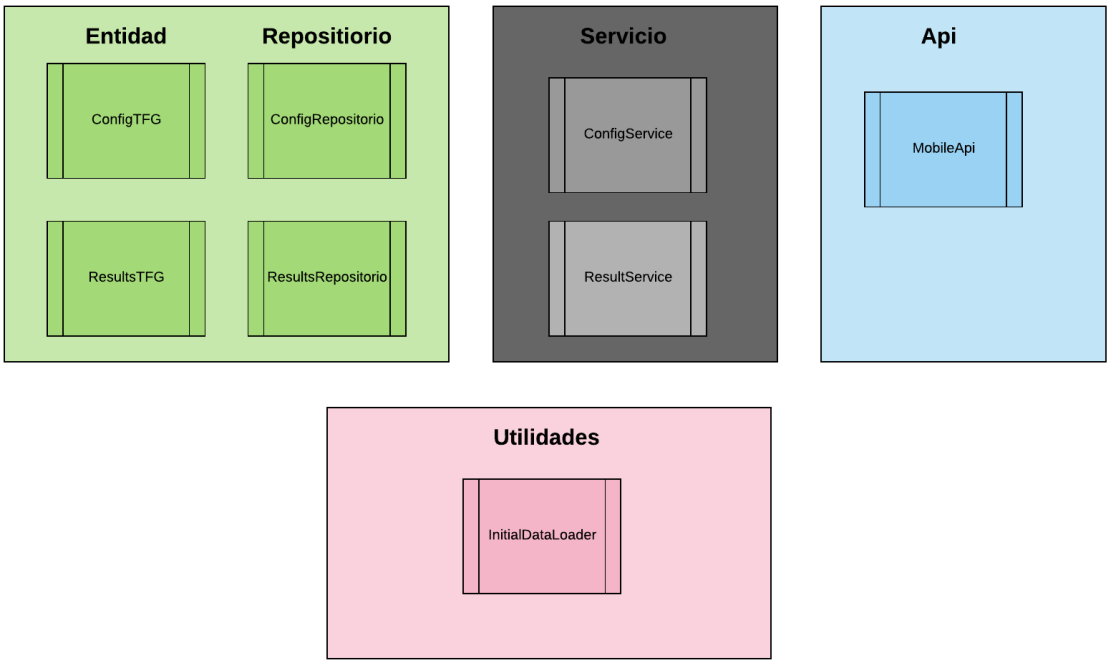

Ilustración 17. Esquema Back end

Para la configuración por usuario, se ha definido una nueva entidad que será nuestra nueva tabla de datos (ver ilustración 18) . Véase que para añadir la tabla de resultados el procedimiento seria el mismo y solo se modificarían las propiedades y el nombre de la tabla.

```java
@Data
@EqualsAndHashCode(of = "id")
@ToString(of = { "id" })
@Entity
@Table(name = "TMHE_CONFIGTFG")
public class configTFG {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "username")
    private Patient patient;
    private double gain;
    private int Yship;
    private int Xship;
    private int Ystar;
    private int speed;
    private int starSize;
    private int totalStars;
    private int levelup;
    private int vidas;
    private int nextStar;
    private int invert;
}
```

Ilustración 18. Tabla Config

Una vez se ha añadido la nueva clase que define la tabla es necesario definir la interfaz (repositorio) que implementara nuestro servicio (ver ilustración 19), en este caso ha sido suficiente definir la operación findByUsername para recuperar un registro a partir de su nombre.

```java
package es.uib.rhbc.db.repositories;

import es.uib.rhbc.db.model.ConfigRepository;

import es.uib.rhbc.db.util.CustomRepository;

public interface ConfigRepository extends CustomRepository<ConfigRepository, Long> {

    ConfigRepository findByUsername(String username);

}
```

Ilustración 19. Repositorio

Con la interfaz definida el último paso es generar el servicio que la implementará y que nos permite añadir la funcionalidad (ver ilustración 20).

```
public class GameSettingsService {

    @Autowired
    ConfigRepository ConfigRepository;


    @Transactional
   public  Config getSettings (String username) {

       return ConfigRepository.findByUsername(username);

   }
}
```

Ilustración 20. Servicio

Por último, se detallan los métodos añadidos a la clase MobileApi, dicha clase está definida como @restcontroller, esta notación sirve para indicar que las peticiones http serán gestionadas por esta clase. Las funcionalidades añadidas han sido: una operación GET que permite la obtención de los valores de configuración a partir del nombre de un paciente (ver ilustración 21) y la operación POST que permite guardar los resultados (ver ilustración 22).

La operación getConfig es la encargada de consultar la tabla de configuración y obtener los registros a partir de un nombre para devolver el JSON que recibirá la aplicación con todos estos valores y realizar el pintado en función de estos.

```
@RequestMapping(value = "/configTFG", method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<ConfigToni> configtoni(Principal principal) {
log.info("LLAMADA SERVICIO de {}", principal.getName());
return new ResponseEntity<ConfigToni>(gameSettingsService.getSettings(principal.getName()), HttpStatus.OK);

}
```

Ilustración 21. GetConfig

Mapeo de la operación para insertar los resultados en la tabla results a partir del JSON recibido, también en el controlador MobileApi.

```
@RequestMapping(value = "/postresult", method = RequestMethod.POST)
public ResponseEntity<?> createUser(@RequestBody ConfigToniResult result) {
log.info("Creating result : {}", result.getUsername());
gameSettingsService.save(result);
return new ResponseEntity<String>(HttpStatus.CREATED);
}
```

Ilustración 22. PostResult

Una vez definidas todas las clases necesarias, ha sido necesario añadir registros a estas tablas, estas inserciones se realizan desde la clase auxiliar initialLoader. Se pueden ver ejemplos de inserciones de un nuevo paciente en la ilustración 23 y una nueva configuración para el paciente en la ilustración 24.

```
Patient patientoni = new Patient();
patientoni.setName("toni");
patientoni.setUsername("toni");
patientoni.setPassword(passwordEncoder.encode("toni"));
patientoni.setWarningHour(LocalTime.of(17, 00)); // 17:00
patientoni.setSelfValuationRate((byte) 2);// 2 weeks
patientoni.setOrganization(organization);
patientRepository.saveAndRefresh(patientoni);
```

Ilustración 23. Inserción Paciente

```
configTFG settings = new configTFG();
settings.setGain(1);
settings.setXship(1);
settings.setYship(1);
settings.setLevelup(4);
settings.setYstar(2);
settings.setStarSize(30);
settings.setTotalStars(10);
settings.setVidas(3);
settings.setSpeed(4);
settings.setNextStar(3);
settings.setNextStar(1);
settings.setInvert(0);
settings.setUsername("toni");
configTFG.saveAndRefresh(settings);
```

Ilustración 24. Inserción Configuración

Una vez implementados estos servicios, se pueden testear sin tener que ejecutar la aplicación desde postman [10]. Se puede observar un ejemplo de una petición Get, en la petición se observa cómo se recibe en formato JSON la configuración para el usuario "toni", véase que estos son los valores que se han cargado en la base de datos cuando se inicializa el back end (ver ilustración 25).

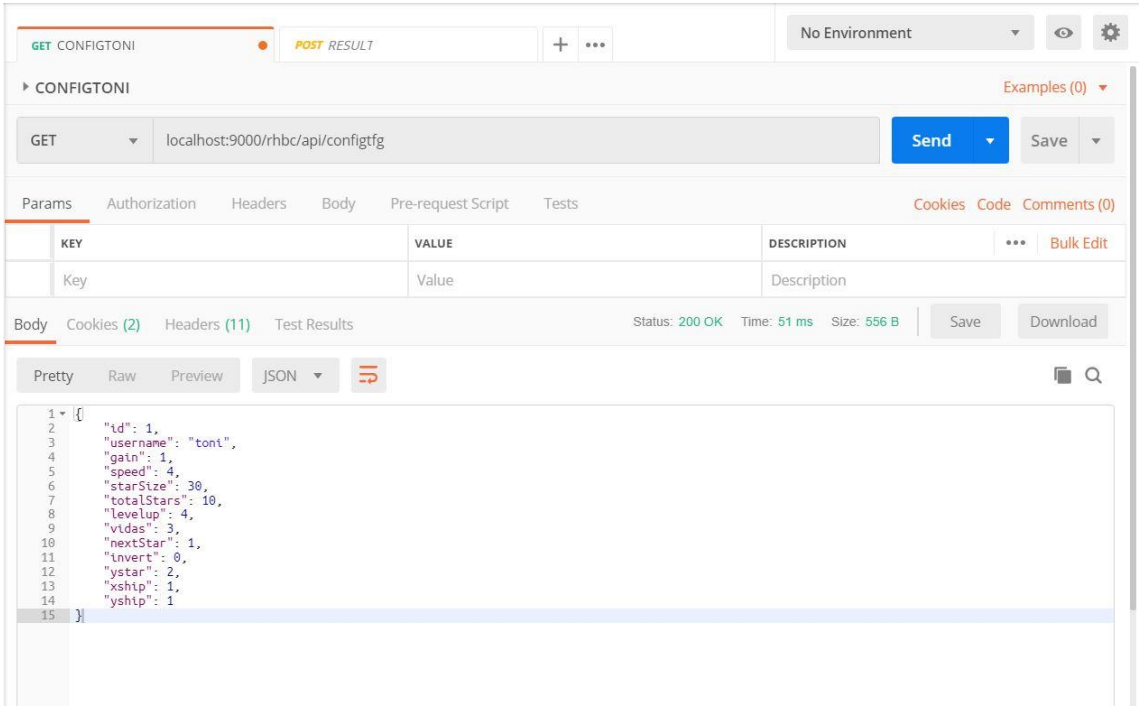

Ilustración 25. GetConfig Postman [10]

Para la petición post, se construye otro JSON con los campos de la tabla resultados, para este caso se ha rellenado este JSON con valores de prueba para realizar la llamada. En la Ilustración 26, se observa que el código de estado tras la ejecución es 201; lo que implica la creación del nuevo registro tal y como se ha definido en la implementación.

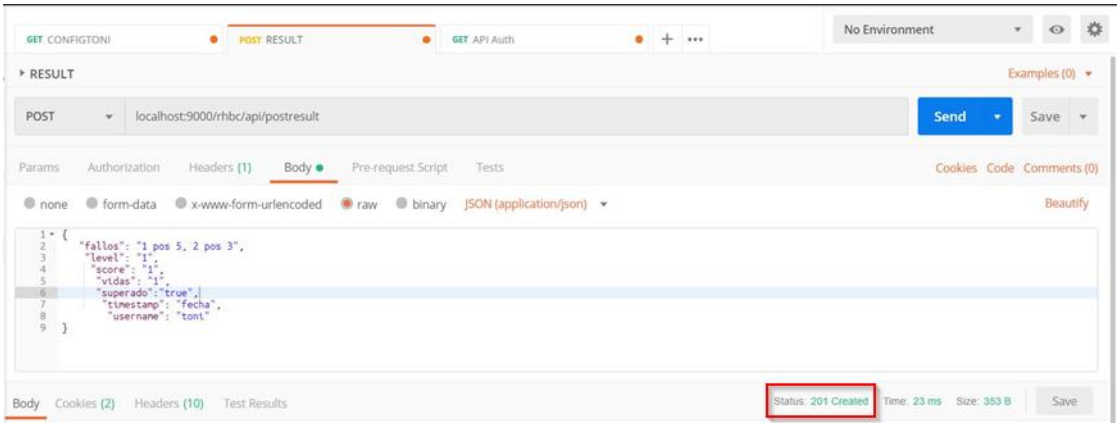

Ilustración 26. PostResult Postman[10]

Con estas peticiones disponibles, el último paso es la integración de estas en la aplicación, haciendo que esta sea capaz de recibir el JSON y aplicar la configuración definida en él y también la construcción del JSON que contiene los resultados para enviarlos al back end. Esta información se detallará en el siguiente apartado de este capítulo.

6.2. Implementación aplicación

Para realizar la implementación de la aplicación se han diseñado las siguientes clases, ver ilustración 27. Todas estas clases serán vistas en detalle en este capítulo.

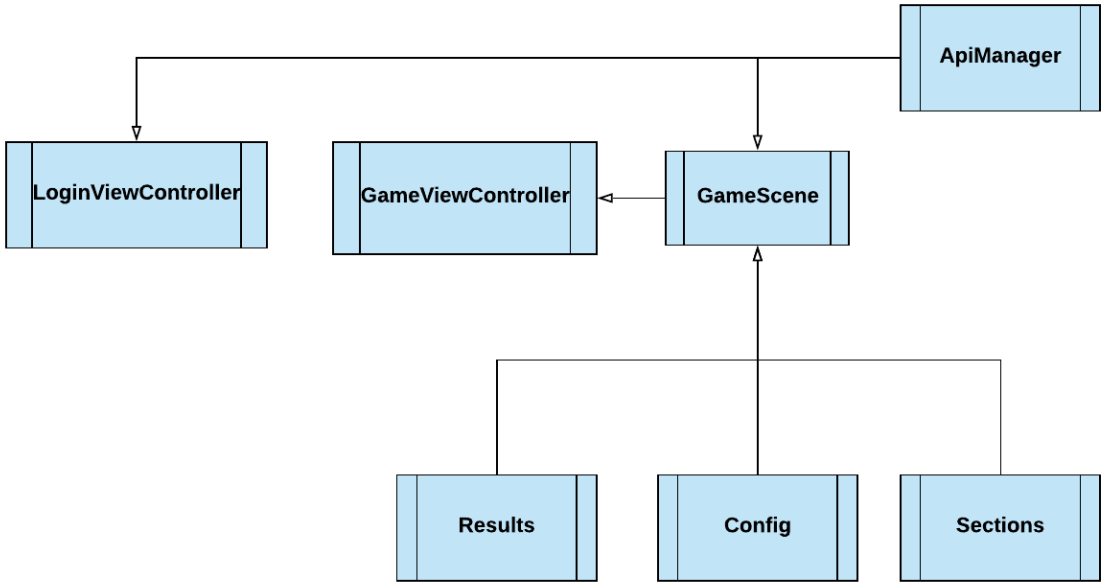

Ilustración 27. Diagrama de Clases

- **LoginViewController**: Vista para realizar el inicio de sesión.
- **GameViewController**: Vista para presentar la escena del juego y gestionar las operaciones del CTTrackingManager proporcionado por el framework "FaceMe".
- **GameScene**: Escena que contiene todos los elementos del juego, haciendo uso del framework SpriteKit.
- **ApiManager**: Clase gestora de las peticiones a la API.
- **Results**: Estructura de datos para almacenar los resultados.
- **Config**: Estructura de datos para almacenar la configuración del juego.
- **Sections**: Clase encargada de crear el sistema de coordenadas.

6.2.1. Structs

6.2.1.1. GameConfig y GameResult

Para almacenar en memoria los diferentes parámetros que recibiremos o mandaremos al back end, se han diseñado dos structs [20] (ver Ilustraciones 28 y 29). Estas estructuras

de datos heredan de la clase *codable* [21]; puesto que con esta podemos serializar y deserializar formato JSON con una sola línea de código.

Estas estructuras cuentan con un constructor que inicializa el contenido de estos structs. Estos valores por defecto serán los que se usen para configurar el videojuego en caso de que no se hayan podido sobrescribir por los valores recibidos desde el back end.

```
import Foundation
struct gameConfig: Codable {

    var id: Int

    var username: String

    var gain: Int

    var starSize, totalStars, lives: Int

    var ystar, xship, yship,levelup,invert: Int

    var nextStar: TimeInterval
    var speed: TimeInterval

    init() {
        id = 0
        username = "test"
        gain = 1
        starSize = 100
        totalStars = 20
        lives = 1
        speed = 1
        nextStar = 1.5
        levelup = 5
        ystar = 5
        xship = 1
        yship = 5
        invert = 0
    }
```

Ilustración 28. Struct Configuración

```
import Foundation
struct gameResult: Codable {


    var username: String

    var superado: String

    var timestamp: String
    var level:Int
    var score:Int
    var vidas: Int
    var fallos:String
    var aciertos:String


    init() {

        username = "test"
        superado = "si"
        timestamp = "test"
        level = 0
        score = 0
        vidas = 0
        fallos = ""
        aciertos = ""

    }

}
```

Ilustración 29. Struct Resultados

6.2.2. Clases

6.2.2.1. Sections

Para implementar el sistema de coordenadas diseñado en la sección 4.3 se ha diseñado la siguiente clase *Sections*, como se ha visto en la ilustración 9, ha sido necesario implementar un sistema de coordenadas personalizado, para ello se ha diseñado la siguiente clase *Sections* que a partir de la altura y anchura de la vista que reciba por parámetro se encarga de dividirlo en 5 secciones. Con esto conseguimos que el sistema de coordenadas sea independiente del dispositivo que ejecute la aplicación.

Además, también gestiona la traducción de estos valores en valores más comprensibles para los fisioterapeutas.

Como ejemplo, supongamos un dispositivo cuya vista tiene 500 puntos de altura. Tras instanciarse la clase S*ections* desde la *gameScene*, se obtiene la sección Y con los diferentes atributos:

- S0:0px+Margen
- S1:100px

- S2:200px
- S3:300px
- S4:400px
- S5: 500px-Margen

Como se trabaja con estos atributos que son de tipo CGFloat [22] al identificar un objetivo perdido se utilizaba un valor numérico. Para facilitar la lectura de los resultados y no devolver valores numéricos al back end se implementó el siguiente casting a posiciones más legibles y comprensibles.

Siguiendo con el ejemplo anterior obtendríamos:

- 0px+Margen se traduce a “S0”
- 100px se traduce a “S1”
- 200px se traduce a “S2”
- 300px se traduce a “S3”
- 400px se traduce a “S4”
- 500px-Margen se traduce a “S5”

6.2.2.2. ApiManager

En este manager se ha definido un atributo que contiene la URL del servidor y una función para cada uno de los web services disponibles en el back end: iniciar sesión, obtener configuración y devolver resultados.

Para iniciar sesión necesitamos mandar dos parámetros: usuario y contraseña. Estos parámetros son enviados desde la clase *LoginController*. Una vez han llegado estos parámetros a nuestro manager se puede montar y mandar la llamada al web service.

Las llamadas a los servicios web pueden ser asíncronas o síncronas. Las llamadas asíncronas no necesitan de una respuesta para seguir con el flujo del hilo principal de ejecución, pero las síncronas si, en nuestro caso al tratarse de un inicio de sesión se ha realizado una llamada síncrona, bloqueando la aplicación hasta que nuestra petición obtenga una respuesta del servidor.

Las funciones para iniciar sesión y para obtener la configuración son funciones síncronas ya que no podremos realizar ninguna otra acción en nuestra aplicación hasta que obtengamos una respuesta por parte del servidor.

Se puede ver en detalle la implementación de la llamada al servicio web en la ilustración 30, dicha función asigna el valor en el struct *gameConfig* a partir del resultado obtenido.

```
func gettSettings(completion: @escaping (_ status: Int) -> ()) {
    let url = URL(string: mainURL+"/rhbc/api/config")!
    var request = URLRequest(url: url)
    let value = "SESSION="+TOKEN!
    request.setValue(value, forHTTPHeaderField: "Cookie")
    request.httpMethod = "GET"
    request.setValue(TOKEN, forHTTPHeaderField: "x-auth-token")
    let task = URLSession.shared.dataTask(with: request, completionHandler:
    { (data: Data?, response: URLResponse?, error: Error?) in

        if let error = error {
            completion(0)
        }
        if let response = response {
            let json = try? JSONSerialization.jsonObject(with: data!, options: [])
            if let json2 = try? JSONDecoder().decode(gameConfig.self, from: data!){
                CONFIG = json2
                completion(1)
                return
            }
        }
        completion(0)
    })
    task.resume()
}
```

Ilustración 30. Función GetSettings

Por otro lado, el contenido del struct *gameResult* se va rellenando durante el transcurso del juego para finalmente llamar a la función *sendPost*() definida en este manager, ver ilustración 31.

```
func sendPost() {
    print("calling post")
    let url = URL(string: mainURL+"/rhbc/api/postresult")!
    var request = URLRequest(url: url)
    let value = "SESSION="+TOKEN!
    request.setValue(value, forHTTPHeaderField: "Cookie")
    request.httpMethod = "POST"
    request.setValue(TOKEN, forHTTPHeaderField: "x-auth-token")
    request.setValue("application/json; charset=utf-8", forHTTPHeaderField: "Content-Type")
    let datapost: Data = try! JSONEncoder().encode(RESULT)
    request.httpBody = datapost

    let session = URLSession.shared
    let dataTask = session.dataTask(with: request as URLRequest, completionHandler: { (data, response, error) -> Void in
        if (error != nil) {
            print(error)
        } else {
            let httpResponse = response as? HTTPURLResponse
            //print(httpResponse)
            print("post done")
        }
    })
    dataTask.resume()
}
```

Ilustración 31. Función SendPost

6.2.2.3. LoginViewController

Desde esta clase, se presenta el menú de inicio de sesión y se gestionan la entrada de los valores usuario y contraseña para realizar el inicio de sesión a través del *ApiManager*.

Para ello se controla el evento click del botón situado en dicha vista. En la ilustración 32, se puede ver en detalle la implementación de esta función. Se ha añadido un modo demo que está orientado a desarrolladores o fisioterapeutas, puesto que permite la ejecución del juego sin usuario y contraseña. Para ello hace uso de los parámetros configurados en el constructor del struct *gameConfig*.

Por último, si las llamadas a los web services se han completado con éxito, se ejecuta la transición a la siguiente vista, *GameViewController*.

```
@IBAction func Click(_ sender: Any) {

    if(usertext.text == ""){
        SERVER = false
        //clear values
        usertext.text = ""
        passwordtext.text = ""
         self.performSegue(withIdentifier: "move", sender: self)

    }
    else{
        SERVER = true
        spiner.startAnimating()
        apiManager.callapi(usr: usertext!.text!, pass: passwordtext!.text!) {
            response in
            print (response)
            if(response > 0){
            self.apiManager.gettSettings() { response in
                if(response>0){
                print("completed")
                DispatchQueue.main.async {
                    self.spiner.stopAnimating()
                    self.performSegue(withIdentifier: "move", sender: self)

                    }

                }
            }
            }
            else {

                self.displayError()

            }


        }

    }
}
```

Ilustración 32. Función Click

Dicha clase también se encarga de controlar los posibles errores que presente el inicio de sesión, enseñando un pequeño mensaje de aviso si dichas credenciales no han sido válidas

o no ha podido establecer conexión con el servidor. Se puede ver su implementación en la ilustración 33.

```
func displayError(){
    DispatchQueue.main.async {
    self.spiner.stopAnimating()
    print("ERROR")
    let alert = UIAlertController(title: "Error", message: "Parece que tenemos un error,
    verifica tus datos y el estado del servidor", preferredStyle: .alert)
    alert.addAction(UIAlertAction(title: "Cerrar", style: .default, handler: nil))
    self.present(alert, animated: true)
    }
}
```

Ilustración 33. Mostrar Error

6.2.2.4 GameViewController

Esta vista gestiona los diferentes estados del juego. En la Ilustración 34, se muestran estos estados y el flujo de esta vista. Para llegar a este punto de inicio, se debe haber realizado previamente el inicio de sesión y la carga de los parámetros a través del *ApiManager* y el *LoginViewController*.

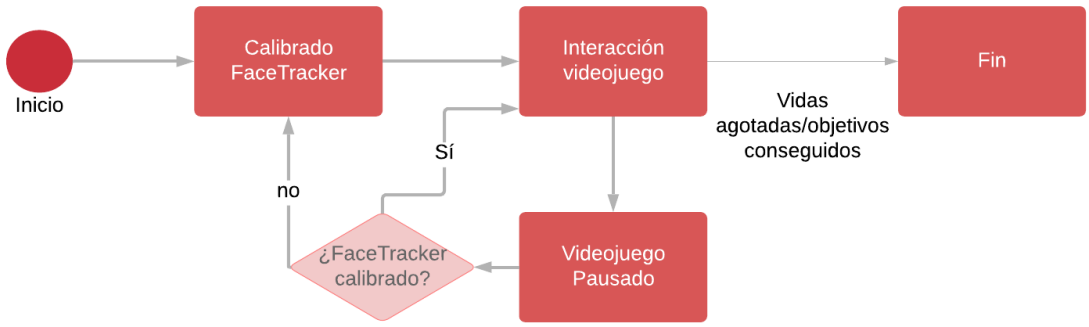

Ilustración 34. Flujo GameViewController

El uso del framework *"FaceMe"* permite instanciar el CTTrackingManager. A través de este objeto proporcionado por el framework *"FaceMe"*, se obtiene acceso a las funciones de calibrado. Una vez calibrado *"FaceMe*", es capaz de proporcionar la posición de la nariz del usuario. Gracias a esta posición la aplicación es capaz de detectar los movimientos del usuario con las operaciones definidas en la sección 4.5. A partir de estos movimientos identificados se actualizarán los valores del jugador en la escena presentada que contiene la nave y los diferentes objetivos.

A continuación, se detallan todas las operaciones definidas en la sección 4.5 de este TFG, donde a partir de la posición de la nariz del usuario somos capaces de obtener el desplazamiento y trasladarlo a nuestra aplicación. Una vez realizadas todas estas operaciones, solo queda asignar la posición de la nave a las nuevas coordenadas obtenidas (ver ilustración 35).

```
func positionTrackedChanged(to position: CTPosition) {


    let alturacast = Double(altura)

    if(trackingManager.isCalibrated){

        self.actualposition = self.trackingManager.kalmanPosition.y

        if (self.previousposition == 0) {
            previousposition = actualposition
        }

        let rec = self.actualposition - self.previousposition;

        let factor = (alturacast / 55)*self.gainY

        if(self.scene != nil && !self.scene.isPaused){

            DispatchQueue.main.async {

                self.scene.player.position.y = CGFloat(self.scene.player.position.y-CGFloat(rec*factor))

            }
            previousposition = actualposition


        }
    }

}
```

Ilustración 35. Cambio de posición

6.2.2.5. GameScene

La vista anteriormente comentada, se encargara de instanciar una nueva vista SKview [23] encargada de presentar la escena SKScene [24] proporcionada por el framework SpriteKit. Esta escena será el contenedor de lo que en SpriteKit se denominan SKAction [25] y SKNode [26], donde los SKNode son todos los elementos que podemos añadir a nuestro videojuego (un fondo, un jugador, un objetivo) y los SKAction son las diferentes acciones que estos pueden realizar (movimientos, aparición/desaparición).

Una vez instanciada la escena, se ejecutará el método presentScene() encargado de renderizar nuestra escena. Se puede ver el ciclo de vida completo de una vista SKView en la ilustración 36.

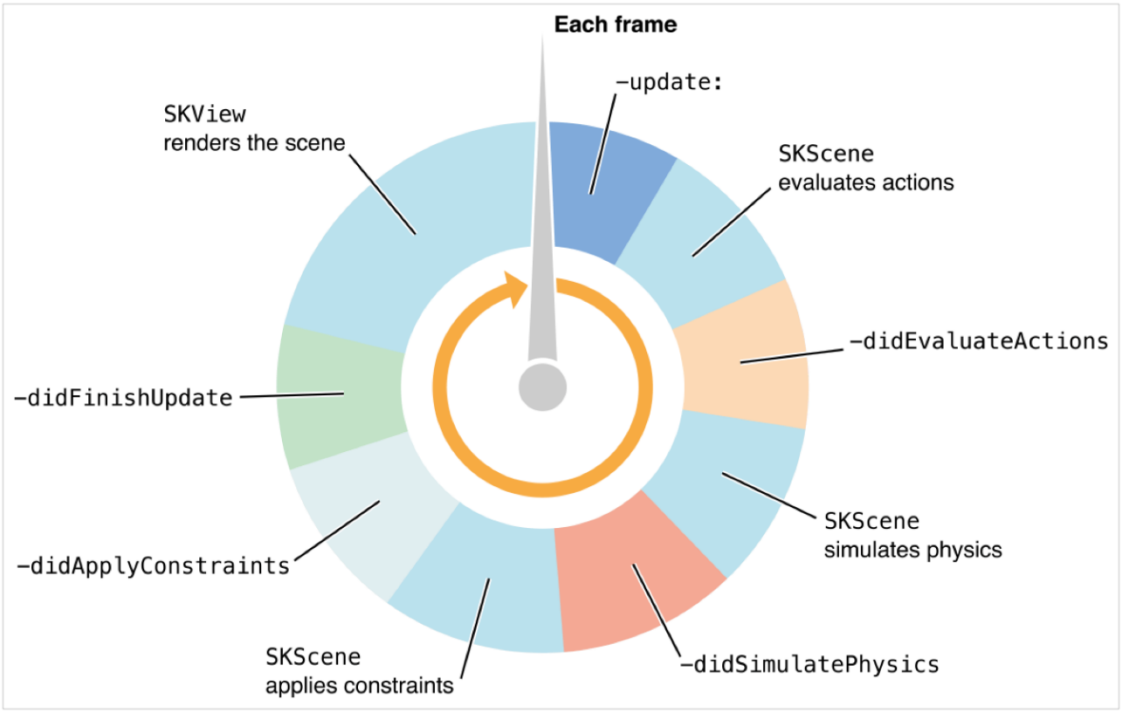

Ilustración 36. Ciclo de vida Skview

A continuación, se detalla la implementación de la función addplayer (ver ilustración 37), encargada de inicializar la posición del jugador a partir de las posiciones configuradas en el *struct GameConfig*.

```
func addplayer() {

    player = SKSpriteNode(imageNamed: "nave")
    player.size.height = 60
    player.size.width = 100

    player.position = CGPoint(x: xSections.getPos(value: CONFIG.xship), y: ySections.getPos(value: CONFIG.yship))
    player.anchorPoint = CGPoint(x: 0.5, y: 0.5)
    player.physicsBody = SKPhysicsBody(circleOfRadius: player.size.width / 3)
    player.physicsBody?.isDynamic = true
    player.physicsBody?.categoryBitMask = CategoryMask.playerCategory.rawValue
    player.physicsBody?.contactTestBitMask = CategoryMask.starCategory.rawValue
    player.physicsBody?.collisionBitMask = 0
    player.physicsBody?.usesPreciseCollisionDetection = true


    let fire = SKEmitterNode(fileNamed: "fire")!
    fire.position = CGPoint(x: -player.size.width/2+14, y: 0.5)
    player.addChild(fire)

    player.isHidden=false
    player.zPosition = 1
    if(CONFIG.invert == 1) {
      player.xScale = player.xScale * -1;
    }

    addChild(player)
}
```

Ilustración 37. Añadir Jugador

Una vez se han situado el fondo y el jugador en la escena, se preparan los objetivos. Todos los objetivos se encuentran en el límite del eje x, obteniendo así el efecto de aparición en pantalla cuando estos van apareciendo. La posición del primer objetivo viene definida por parámetro, mientras que el resto de las posiciones se obtienen a través de un algoritmo que genera una posición aleatoria en el eje. Estas posiciones disponibles son las 5 opciones explicadas en la clase *Sections*. La nueva posición obtenida puede diferir en uno respecto la posición anterior, controlando así que el usuario no pueda realizar movimientos muy bruscos. Como ejemplo, a partir de un objetivo en la posición 4 del eje, el siguiente objetivo obtendrá una posición aleatoria entre los valores 3, 4 o 5.

Además de las posiciones, se definen también el tamaño de estos objetivos a partir de los parámetros configurados, ver ilustración 38.

```
var size = CONFIG.starSize
let star = SKSpriteNode(imageNamed: "star")
star.size.height = CGFloat (size)
star.size.width =  CGFloat (size)
```

Ilustración 38. Configuración objetivos

Los objetivos implementan la velocidad con la que realizaran el recorrido en el eje X, desde el valor de origen a 0 menos la altura de la estrella. Con esto se tendrá identificado también el tiempo que estará en pantalla dicho objetivo, ver ilustración 39.

```
var starmove = SKAction.moveTo(x: 0-star.size.width/2, duration: timestars)
```

Ilustración 39. Definición movmiento

Una vez se han añadido los objetivos, solo queda detectar si estos han llegado al final del recorrido, es decir, no han sido capturados por el jugador o bien han llegado al final de su recorrido. Para ello, se configura otra acción que se ejecuta si estos objetivos no son recolectados, añadiendo su posición de origen a la cadena de caracteres que contiene todos los fallos del jugador y ejecutando un pequeño sonido de error para notificar al usuario.

Para construir la cadena de caracteres que contiene todos los objetivos no capturados por el usuario junto con las características de estos se hace uso del siguiente código (ver ilustración 40).

```
let posicioneje = self.ySections.getValores(value : self.starpos[self.starindex])
RESULT.fallos += "Pos:"+posicioneje+","
```

Ilustración 40. Guardado de fallos

Para la reproducción de sonidos, en este caso la pérdida de un objetivo, es necesario implementar la siguiente sentencia (ver ilustración 41).

```
self.run(SKAction.playSoundFileNamed("fail.mp3", waitForCompletion: false))
```

Ilustración 41. Implantación sonido

Para el caso contrario, es decir, la recolección de un objetivo se hace uso del sistema de física que implementa SpriteKit, SKPhysicsContact [27].

Durante la creación de los nodos del jugador y los objetivos se les ha añadido la capacidad de interactuar, mediante los diferentes atributos que proporciona la clase *physicsBody* (ver detalles en Código 6. Gracias a SpriteKit todos los elementos que implementen estas propiedades dispararan el método definido que se puede ver ilustración 42.

```
//colision
func didBegin(_ contact: SKPhysicsContact) {

    starindex = starindex + 1
    //identify and remove s=star from scene.
    let collect = SKEmitterNode(fileNamed: "collect")!
    collect.position = player.position
    collect.particleSize = player.size
    self.addChild(collect)
    self.run(SKAction.playSoundFileNamed("collect.mp3", waitForCompletion: false))
    score = score+1
    updatevalues()

}
```

Ilustración 42. Colisiones

Una vez identificada la colisión, el objetivo se elimina de la pantalla, aparecen las partículas de recolección alrededor del jugador durante 0.5 segundos y se ejecuta el sonido de recolección.

Ya sea por una recolección o por la pérdida un posible objetivo, los valores de la interfaz que representan el nivel, las vidas y la puntuación se actualizan mediante la siguiente función, ver ilustración 43.

```
func updatevalues (){
    DispatchQueue.main.async {
        GameViewController.sharedInstance.labelNivel.text = String(self.level)
        GameViewController.sharedInstance.labelScore.text = String(self.score)
        GameViewController.sharedInstance.labelvidas.text = String(self.vidas)
    }
}
```

Ilustración 43. Actualización de valores

Por último, desde esta clase cuando se identifique el fin del juego, ya sea por haber completado el nivel o bien por haber agotado las vidas disponibles, se ejecuta el siguiente método, encargado de enviar todos los resultados de la ejecución al back end, haciendo uso del *apiManager* y el *struct RESULT* del tipo *gameResult* ver ilustración 44.

```
func postresult(){
    RESULT.vidas = vidas
    RESULT.score = score
    RESULT.username = CONFIG.username
    RESULT.level = level
    let formatter = DateFormatter()
    formatter.dateFormat = "yyyy-MM-dd HH:mm:ss"
    let fecha = formatter.string(from: Date())
    print(fecha)
    RESULT.timestamp = fecha
    let apiManager : ApiManager = ApiManager.init()
    apiManager.sendPost()
}
```

Ilustración 44. Post Result

7. Conclusiones

La meta de este TFG era implementar un nuevo videojuego para el proyecto RHBC, con la finalidad de motivar a realizar el ejercicio terapéutico que los fisioterapeutas establecen a sus pacientes y complementar el tratamiento de rehabilitación. En concreto, se buscaba desarrollar un juego para el entrenamiento de los movimientos verticales del cuello.

La implementación de este videojuego se ha realizado con éxito, pues se ha conseguido un nuevo videojuego totalmente integrado en el proyecto actual. Además, ha sido aceptado y presentado como demo en el XX Congreso internacional de interacción persona-ordenador (Ver sección 8, Publicaciones relacionadas y divulgación) Dicho videojuego ofrece un gran nivel de personalización para que pueda ser adaptado por parte de los fisioterapeutas a los diferentes usuarios que vayan a hacer uso, además se podría parametrizar el juego para convertirlo en un juego puramente lúdico si se desea.

7.1. Trabajo futuro

El trabajo futuro de este videojuego incluye la adición de nuevas funcionalidades como sistema de notificaciones para mejorar aún más la adherencia del usuario, añadir un nuevo tipo de objetivos para que sean esquivados y no recogidos como los que se presentan actualmente o poder personalizar la dificultad añadida al subir de nivel con nuevos parámetros.

A continuación, se detallan nuevas funcionalidades:

- Sistema de notificaciones: En la versión entregada la aplicación no contempla notificar al usuario, por lo que puede ser interesante añadir notificaciones para avisar al usuario que lleva varios días sin ejecutar la aplicación con el fin de incrementar aún más la adherencia y constancia del usuario.
- Nuevos objetivos: Otra posible mejora sería la adición de diferentes tipos de objetivos. Actualmente solo existe un tipo y debe ser recogido para aumentar la puntuación. Los nuevos objetivos pueden tener formas y puntuaciones diferentes, así como objetivos que deban ser esquivados para evitar la monotonía.
- Progreso del usuario: La información del progreso que va obteniendo el usuario, así como todo el historial de las acciones realizadas en el juego solo están disponibles en el back end para los fisioterapeutas. Por eso, otra funcionalidad

extra a considerar sería añadir una nueva opción en el menú donde se pueda consultar el historial de ejecuciones por parte del usuario.

- Idiomas: Añadir soporte para más idiomas, ya que ahora solo soporta inglés y castellano.
- Tutorial: Para todos aquellos usuarios nuevos, sería interesante añadir una sección que contenga recomendaciones para el uso de la aplicación y un breve tutorial.

Una vez se hayan añadido y probado todas estas funcionalidades, puede ser interesante la publicación de una versión del videojuego como juego no serio en la plataforma de distribución de aplicaciones de Apple, *App Store*.

Por último, puesto que era uno de los objetivos principales del proyecto RHBC, se deberá realizar un estudio evaluando el uso de la aplicación por parte de los usuarios. Estos usuarios harán uso de la aplicación bajo la supervisión de los fisioterapeutas para evaluar clínicamente el progreso de los usuarios.

8. Publicaciones relacionadas y divulgación

- Antonio Arenas, Maria Francesca Roig-Maimó, Iosune Salinas Bueno, Javier Varona, Katia San-Sebastián-Fernández, Cristina Manresa-Yee, 2019. Therapeutic Exercise Based on Videogames to Improve Neck Pain. In Proceedings of the XX International Conference on Human-Computer Interaction (Interacción 2019), 2 pages. Se puede consultar en el anexo (A.3).

- Demo en la XX Internationtal Conference on Human-Computer Interaction (Interacción 2019). 25 al 28 de junio en San Sebastián, España.

9. Referencias

[1] “Cheng, C. H., Su, H. T., Yen, L. W., Liu, W. Y., & Cheng, H. Y. K. (2015). ‘Long-term effects of therapeutic exercise on nonspecific chronic neck pain: a literature review’. Journal of physical therapy science, 27(4),1271-1276.” .

[2] “Zronek, M., Sanker, H., Newcomb, J., & Donaldson, M. (2016). ‘The influence of home exercise programs for patients with non-specific or specific neck pain: a systematic review of the literature’. Journal of Manual &Manipulative Therapy, 24(2), 62-73.” .

[3] “Chen, H. C., Liu, Y. P., Chen, C. L., & Chen, C. Y. (2007). ‘Design and feasibility study of an integrated pointing device apparatus for individuals with spinal cord injury’. Applied ergonomics, 38(3), 275-283.” .

[4] “Roig-Maimó, M.F., Manresa-Yee, C., Varona, J. (2016) A Robust Camera-Based Interface for Mobile Entertainment. Sensors, 16, pp. 254.” .

[5] “Manresa-Yee, C., Ponsa, P., Salinas, I., Perales, F. J., Negre, F., & Varona, J. (2014). ‘Observing the use of an input device for rehabilitation purposes’. Behaviour & Information Technology, 33(3), 271-282.” .

[6] A. Inc, “Swift.org,” *Swift.org*. [Online]. Available: https://swift.org.ard Edition 7 Documentation.” [Online]. Available: https://docs.oracle.com/javase/7/docs/. [Accessed: 11-Jul-2019].

[8] “Xcode,” *Apple Developer*. [Online]. Available: https://developer.apple.com/xcode/.

[9] D. Roy, “Eclipse Documentation | The Eclipse Foundation.” [Online]. Available: https://www.eclipse.org/documentation/. [Accessed: 11-Jul-2019].

[10] “Intro to API documentation,” *Postman Learning Center*. [Online]. Available: https://learning.getpostman.com/docs/postman/api_documentation/intro_to_api_documentation/.

[11] “The digital design toolkit,” *Sketch*. [Online]. Available: https://www.sketch.com/. [Accessed: 11-Jul-2019].

[12] “SpriteKit - Apple Developer.” [Online]. Available: https://developer.apple.com/spritekit/.

[13] “Tutorial · Building REST services with Spring.” [Online]. Available: https://spring.io/guides/tutorials/rest/.

[14] “Spring Projects.” [Online]. Available: https://spring.io/projects.

[15] “Web Services Glossary.” [Online]. Available: https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice.

[16] “Web Services Architecture.” [Online]. Available: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest.

[17] “In Introduction to HTTP Basics.” [Online]. Available: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html.

[18] “RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1.” [Online]. Available: https://tools.ietf.org/html/rfc2616#section-9.1.

[19] “H2 Database Engine.” [Online]. Available: https://www.h2database.com/html/main.html.

[20] “Structures and Classes — The Swift Programming Language (Swift 5.1).” [Online]. Available: https://docs.swift.org/swift-book/LanguageGuide/ClassesAndStructures.html.

[21] “Encoding and Decoding Custom Types | Apple Developer Documentation.” [Online]. Available: https://developer.apple.com/documentation/foundation/archives_and_serialization/encoding_and_decoding_custom_types.

[22] “CGFloat - Core Graphics | Apple Developer Documentation.” [Online]. Available: https://developer.apple.com/documentation/coregraphics/cgfloat.

[23] “SKView - SpriteKit | Apple Developer Documentation.” [Online]. Available: https://developer.apple.com/documentation/spritekit/skview.

[24] “SKScene - SpriteKit | Apple Developer Documentation.” [Online]. Available: https://developer.apple.com/documentation/spritekit/skscene.

[25] “SKAction - SpriteKit | Apple Developer Documentation.” [Online]. Available: https://developer.apple.com/documentation/spritekit/skaction.

[26] “SKNode - SpriteKit | Apple Developer Documentation.” [Online]. Available: https://developer.apple.com/documentation/spritekit/sknode.

[27] “SKPhysicsContact - SpriteKit | Apple Developer Documentation.” [Online]. Available: https://developer.apple.com/documentation/spritekit/skphysicscontact.

ANEXOS

A.1 Fuentes

- Fondos, nave jugador y objetivos
  https://opengameart.org/
- Iconos
  https://www.flaticon.com/
- Música de fondo y recolección
  https://freesound.org/

A.2 Ejemplos de configuración

Ejemplos de configuraciones, en formato JSON y su representación en la aplicación.

Configuración 1

JSON


```
{
   "id": 1,
   "username": "toni",
   "gain": 1,
   "speed": 4,
   "starSize": 30,
   "totalStars": 10,
   "levelup": 4,
   "vidas": 3,
   "nextStar": 1,
   "invert": 0,
   "yship": 1,
   "xship": 1,
   "ystar": 2
}
```

Representación

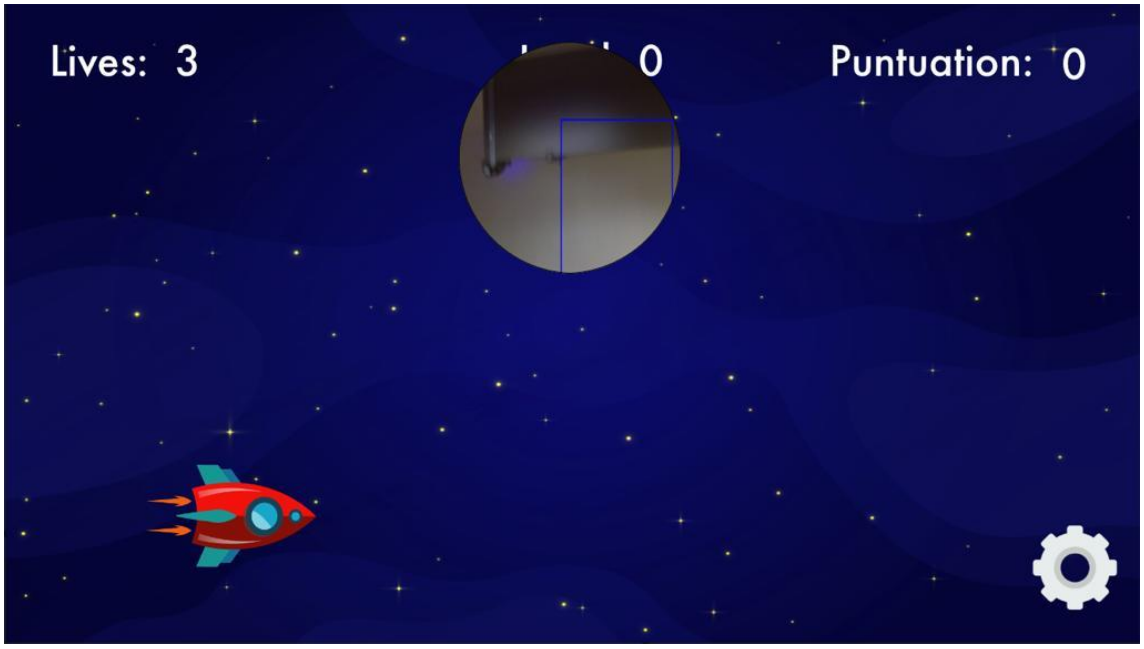

Ilustración 45. Ejemplo 1 Representación

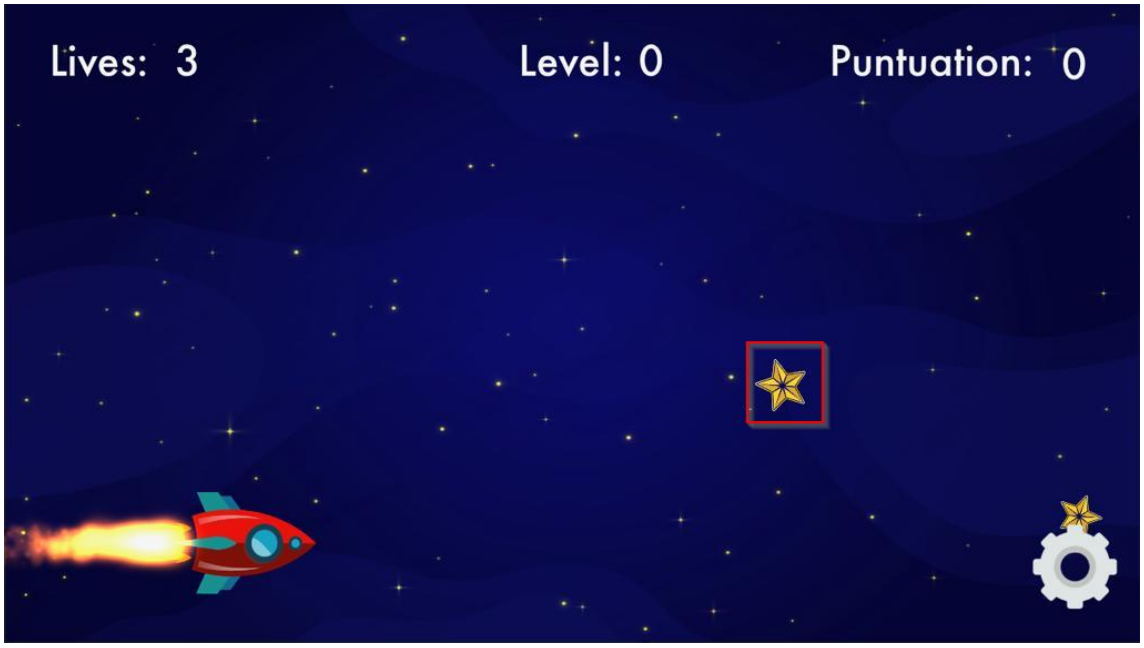

Ilustración 46. Ejemplo 1 Representación

Configuración 2

JSON


```
{
  "id": 2,
   "username": "pepe",
   "gain": 1,
   "speed": 4,
   "starSize": 10,
   "totalStars": 10,
   "levelup": 4,
   "vidas": 30,
   "nextStar": 1,
   "invert": 0,
   "yship": 2,
   "xship": 2,
   "ystar": 2
}
```

Representación

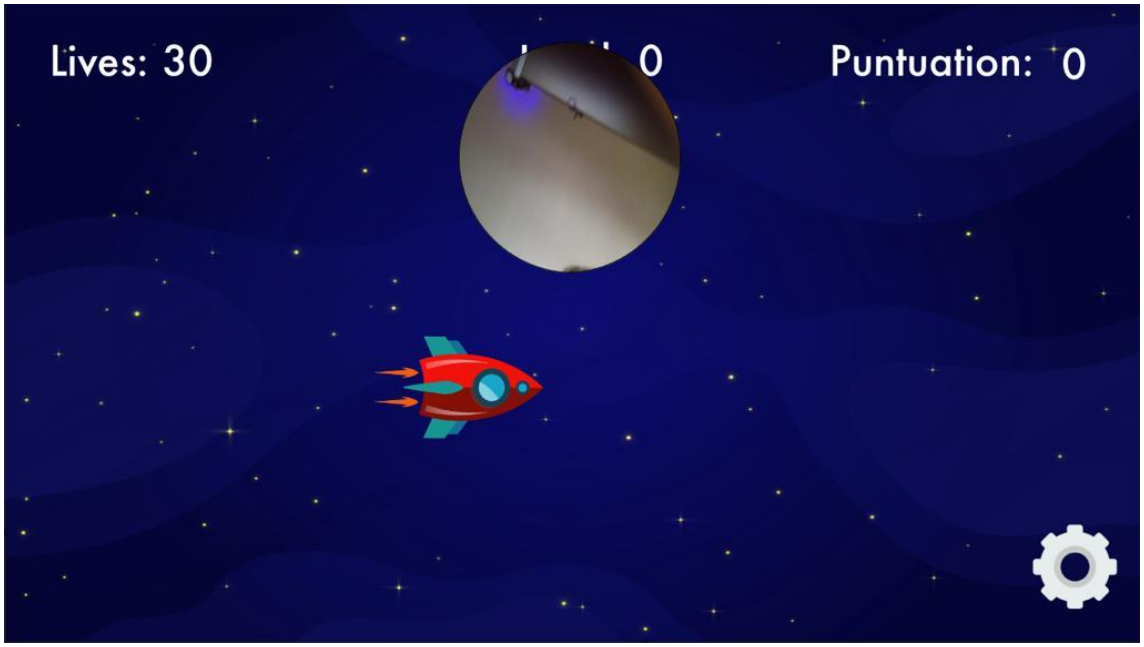

Ilustración 47. Ejemplo 2 Representación

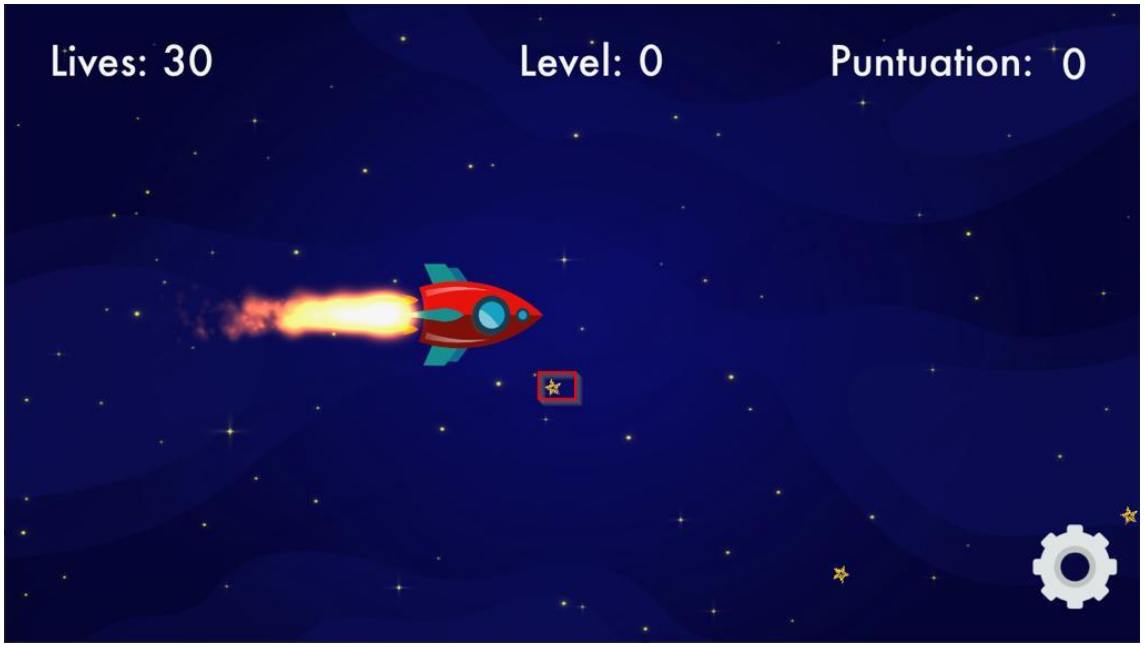

Ilustración 48. Ejemplo 2 Representación

A.3 Paper Interacción 2019

Therapeutic Exercise Based on Videogames to Improve Neck Pain

Antonio Arenas
Department of Mathematics and Computer Science
University of Balearic Islands
Palma, Spain
antonio.arenas1@estudiant.uib.cat

Javier Varona
Department of Mathematics and Computer Science
University of Balearic Islands
Palma, Spain
xavi.varona@uib.es

Maria Francesca Roig-Maimó
Department of Mathematics and Computer Science
University of Balearic Islands
Palma, Spain
xisca.roig@uib.es

Katia San-Sebastián-Fernández
Department of Nursing and Physiotherapy
University of Balearic Islands
Palma, Spain
katia.sansebastian@uib.es

Iosune Salinas-Bueno
Department of Nursing and Physiotherapy
University of Balearic Islands
Palma, Spain
iosune.salinas@uib.es

Cristina Manresa-Yee
Department of Mathematics and Computer Science
University of Balearic Islands
Palma, Spain
cristina.manresa@uib.es

ABSTRACT

The demo presents a videogame for mobile devices designed by a multidisciplinary team comprised by physiotherapists and computer scientists. The videogame aims at motivating the patient to exercise the neck zone to relieve the neck pain, reduce the neck disability and increase the adherence to the therapeutic treatment. The patient interacts with head movements captured by the frontal camera of the device and the physiotherapists can adapt the system to each patient by configuring the settings.

CCS CONCEPTS

• Applied computing~Consumer health. • Human-centered computing~Gestural input

KEYWORDS

Neck pain; Therapeutic neck exercise; Vision-based interface

ACM Reference format:
Antonio Arenas, Maria Francesca Roig-Maimó, Iosune Salinas-Bueno, Javier Varona, Katia San-Sebastián-Fernández, Cristina Manresa-Yee, 2019. Therapeutic Exercise Based on Videogames to Improve Neck Pain. In *Proceedings of the XX Internaciontal Conference on Human-Computer Interaction (Interacción 2019), 2 pages.* https://doi.org/10.1145/3335595.3335599

Interacción 2019, June 25–28, 2019, Donostia, Gipuzkoa, Spain

ACM ISBN 978-1-4503-7176-6/19/06.
https://doi.org/10.1145/3335595.3335599

1 Introduction

The demo we present, is based on a previous work where we designed a vision-based interface which replaced the mouse in desktop environments addressed to people with motor impairments [5]. A study carried out with people with cerebral palsy using the interface for five months showed benefits regarding the improvement of head control, coordination or the increase of neck range movement and endurance [4]. Based on the results of this experience, in this work, we address therapeutic exercise to improve the non-specific neck pain using mobile devices.

Non-specific neck pain is a very common musculoskeletal disorder [8] which often becomes chronic, that is, the neck pain persists for more than three months. This condition restricts the functionality of the person, affecting their quality of life [2] and it also has an economic impact for the families, the community and the health system [3].

The rehabilitation of non-specific neck pain encompasses multiple therapeutic treatments such as mobilizations, manual therapy or therapeutic exercise. In addition, the therapeutic exercise has shown to be effective in relieving pain and improving the quality of life of people with neck pain. However, therapeutic exercises require perseverance, since their effect is not maintained over time, so adherence to treatment is essential [1] .

One way to increase adherence is to entertain the patient while performing therapeutic exercises through an exergame, that is, a video game that aims at physical exercise while amusing the player, and in our particular case the ultimate goal of the exergame is the relieving the neck pain.

We adapted the aforementioned design of the vision-based interface to mobile devices, and used the frontal camera to sense

the patient and use the visual information in an HCI context [6]. Taking into account the characteristics of the interface, the screen's size and the neck movements to perform in a safe manner, the multidisciplinary team composed by physiotherapists and computer scientists, designed a set of video games to exercise different neck movements and positions.

These videogames are included in a platform where for each patient, the physiotherapists have planned a set of exercises (performed within a videogame) with personal configurations depending on the patients' needs. When a patient logs into the system, there is an initial stage to warm up the neck using videos, just after, the patient will have access to the videogames defined for his or her profile. Finally, a cool down video finishes the session. Further, the results of the performance are stored in the platform to evaluate clinically the progress of the patient and count with clinical evidence of the safety, performance and effectiveness of the system [7].

2 Demo

The demo consists in one of the games included in the platform (see Fig. 1). The aim of the game is to collect the stars without missing any.

2.1. Vision-based Interface

The interface uses the information of the frontal camera of the mobile device to detect features in the nose region. We chose the nose because is almost always visible in all positions of the head facing the screen, it is in the center of the face and it is not occluded by beards, moustaches or glasses. We select a set of points around the nose that are easily tracked (e.g. nostrils). Then, these points are tracked, recovered when lost and finally we calculate the average as the final nose position for each frame. The nose position is then mapped to the mobile screen using a pre-defined transfer function in order to use the nose as a pointer on the mobile's screen.
A more detailed and technical description of the interface can be found in [6].

2.2. Videogame settings

The clinical aim of the videogame is to train the vertical neck movement. Physiotherapists recommend using a tablet instead of a mobile phone (due to the screen size) and place it comfortably at the eyes level (see Fig. 2). The physiotherapists will configure for each patient and level the next parameters:

- Vertical initial position of the spaceship and direction (left to right or right to left): as an example, when the spaceship starts in the lower positions, a greater effort has to be done to reach the higher positions on the screen.
- Maximum vertical distance between consecutive stars: therapists do not want patients to perform abrupt movements, therefore the distance between consecutive stars is part of the configuration.
- Speed and size of the stars
- Distance between consecutive stars

Figure 1. GUI interface of the videogame

Figure 2. Recommended placement. Elbows can lean on the table for comfort, or the tablet can be placed on a surface.

ACKNOWLEDGMENTS

This work has been partially supported by the project TIN2016-81143-R (AEI/FEDER, UE). We also thank the Balearic Islands University and its Department of Mathematics and Computer Science for their support.

REFERENCES

1. Hsieh-Ching Chen, Yung-Ping Liu, Chia-Ling Chen, and Chih-Yong Chen. 2007. Design and feasibility study of an integrated pointing device apparatus for individuals with spinal cord injury. *Applied Ergonomics* 38, 3: 275–283. https://doi.org/DOI: 10.1016/j.apergo.2006.06.003

2. Chih-Hsiu Cheng, Hao-Tsung Su, Ling-Wei Yen, Wen-Yu Liu, and Hsin-Yi Kathy Cheng. 2015. Long-term effects of therapeutic exercise on nonspecific chronic neck pain: a literature review. *Journal of Physical Therapy Science* 27, 4: 1271–1276. https://doi.org/10.1589/jpts.27.1271

3. D G Hoy, M Protani, R De, and R Buchbinder. 2018. The epidemiology of neck pain. *Best Practice & Research Clinical Rheumatology* 24, 6: 783–792. https://doi.org/10.1016/j.berh.2011.01.019

4. C. Manresa-Yee, P. Ponsa, I. Salinas, F.J. Perales, F. Negre, and J. Varona. 2014. Observing the use of an input device for rehabilitation purposes. *Behaviour and Information Technology* 33, 3. https://doi.org/10.1080/0144929X.2013.795607

5. C. Manresa-Yee, P. Ponsa, J. Varona, and F.J. Perales. 2010. User experience to improve the usability of a vision-based interface. *Interacting with Computers* 22, 6. https://doi.org/10.1016/j.intcom.2010.06.004

6. M.F. Roig-Maimó, C. Manresa-Yee, and J. Varona. 2016. A robust camera-based interface for mobile entertainment. *Sensors (Switzerland)* 16, 2. https://doi.org/10.3390/s16020254

7. K. San-Sebastián-Fernández, J. Varona, M.F. Roig-Maimó, and I. Salinas-Bueno. 2018. Clinical evaluation of a mobile app for therapeutic exercise for neck pain. In *Proc. of the Second International Conference on Accessibility, Inclusion and Rehabilitation using Information Technologies*, 11–13.

8. Margaret Zronek, Holly Sanker, Jennifer Newcomb, and Megan Donaldson. 2016. The influence of home exercise programs for patients with non-specific or specific neck pain: a systematic review of the literature. *Journal of Manual & Manipulative Therapy (Maney Publishing)* 24, 2: 62–73. https://doi.org/10.1179/2042618613Y.0000000047