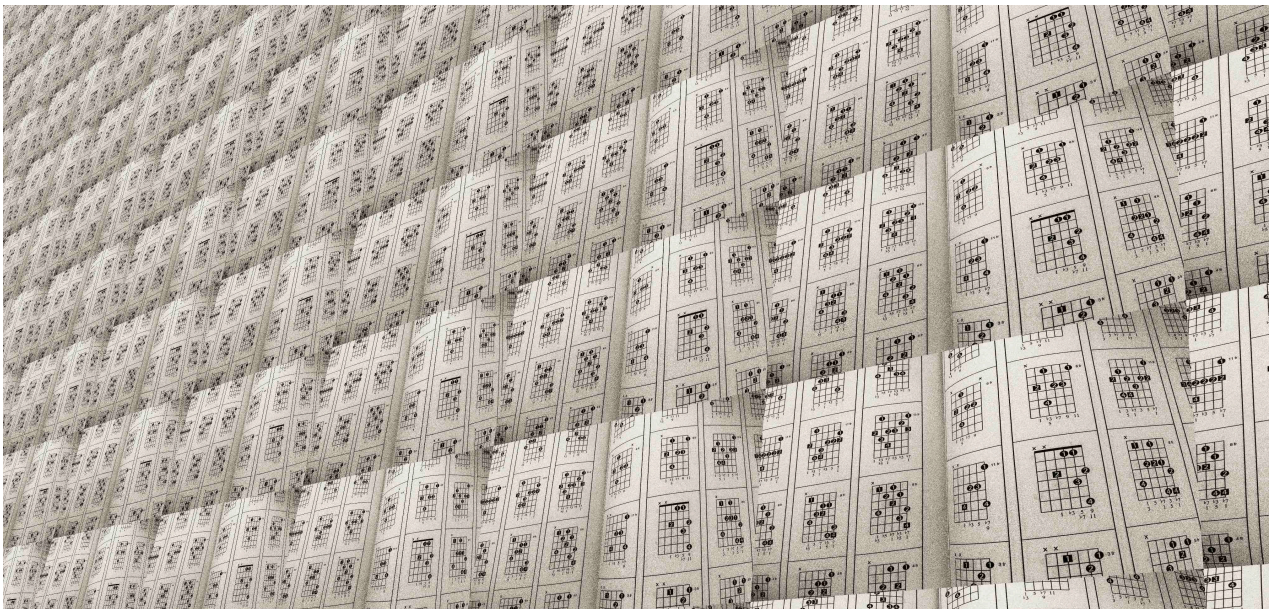


Desarrollo de una aplicación móvil para el estudio de patrones musicales en la guitarra

- Trabajo Fin de Máster -

Curso 2018/2019

Alberto Banet Masa
Máster universitario 'Desarrollo de aplicaciones para dispositivos móviles'
Desarrollo de aplicaciones para dispositivos iOS

Pau Dominkovics Coll
Carles Garrigues Olivella

05 de junio de 2019

Desarrolla una pasión por aprender. Si lo haces, nunca dejarás de crecer.
Anthony J. D'Angelo

FICHA DEL TRABAJO FINAL

Título del trabajo:	Desarrollo de una aplicación móvil para el estudio de patrones musicales en la guitarra.
Nombre del autor:	Alberto Banet Masa
Nombre del consultor:	Pau Dominkovics Coll
Nombre del PRA:	Carles Garrigues Olivella
Fecha de entrega:	05 de junio de 2019
Titulación:	Máster universitario 'Desarrollo de aplicaciones para dispositivos móviles'
Área del Trabajo Final:	Desarrollo de aplicaciones para dispositivos *iOS*
Idioma del trabajo:	Castellano
Palabras clave:	Videojuegos, Educación, Guitarra, Dispositivos Móviles
Palabras clave:	Videojuegos, Educación, Guitarra, Dispositivos Móviles

Resumen del Trabajo Fin de Máster

El presente proyecto propone una aplicación que ayuda al estudiante de guitarra en su educación musical y lo hace explorando el entorno de programación que Apple ofrece para el desarrollo de videojuegos en dispositivos móviles. Se propone una enseñanza gamificada que aproveche el auge actual tanto de la tecnología móvil como de los entornos de desarrollo de videojuegos que permiten convertir el móvil en una herramienta de enseñanza muy avanzada.

A través de una metodología ágil se desarrolla una aplicación que permite al usuario, tanto estudiar patrones musicales, como crear y compartir los suyos propios con otros usuarios.

Las pruebas finales con usuarios indican que se produce adquisición del conocimiento -lo cual significa que estamos en el buen camino-, si bien consideramos que esta aplicación es tan sólo un punto de partida que señala una dirección en la que todavía queda mucho por hacer.

En cuanto al entorno de desarrollo de Apple para videojuegos 2D -siendo muy interesante-, pensamos que todavía está lejos de la potencia y facilidad de uso de las herramientas multiplataforma para el desarrollo de videojuegos. Con menos esfuerzo se pueden obtener los mismos resultados con la ventaja de obtener un producto multiplataforma. Lo mismo ocurre con el backend y la plataforma de gamificación de Apple: todavía son muy básicas y podemos encontrar otras soluciones más abiertas y mucho más evolucionadas.

Abstract

Current project is a mobile app for helping guitar students during their musical learning process, exploring the programming environment that Apple offers for the development of video games on mobile devices. Is a proposal for a gamified teaching that takes advantage as consequence of the current boom of both mobile technology and videogames development environments that allow converting the mobile device into a very advanced teaching tool.

Through an agile methodology, the app is developed that allows to the user to study musical patterns as well as create and share their own patterns with other users.

The final tests with users indicate a knowledge acquisition occurs, which means we are on the right way, although this app is only just a starting point that points an address where there is still too much to be done.

Regarding Apple development environment for videogames 2D, in spite of being so interesting, we think that it is still far from the power and facility of use of multiplatform tools for the development of videogames. With less effort the same result can be done with the advantage of getting a multiplatform product. As well occurs with the backend and the Apple Gaming platform: still so basics and we can find better solutions and higher evolved.

ÍNDICE

INTRODUCCIÓN

1. Contexto y justificación del trabajo

El trabajo actual se enmarca en dos contextos:

1.1. Contexto tecnológico

A lo largo del máster de desarrollo de aplicaciones móviles hemos estudiado el entorno de desarrollo de *apps* para *iOS* y hemos tenido una aproximación al desarrollo de videojuegos con la asignatura de '*Introducción a videojuegos en dispositivos móviles*'.
Siendo estas las asignaturas que más he disfrutado he decidido afrontar un proyecto que me permita conocer en mayor profundidad el entorno de desarrollo *iOS* y, en concreto, el *framework SpriteKit* [1] que *Apple* ofrece para el desarrollo de videojuegos en 2D.

De forma transversal se estudiarán el *Game Center* [2], que *Apple* ofrece para facilitar la socialización del las aplicaciones a través de su *framework GameKit* [3], y el uso del almacenamiento *iCloud* (*framework CloudKit* [4]) en el que *Apple* permite almacenar tanto datos privados del usuario como datos compartidos con otros usuarios.

1.2. Contexto funcional

El proyecto está dirigido a aquellas personas que se embarcan en el arte de tocar la guitarra, entre los que me incluyo.

Después de la fase inicial, en la que el estudiante de guitarra ha adquirido un mínimo de la habilidad necesaria para empezar a descubrir el instrumento, nos damos cuenta de una realidad común a cualquier instrumento musical: la música está primero en la cabeza y después en el instrumento. Este descubrimiento obliga a duplicar el esfuerzo. Por una parte hay que seguir dedicando horas a mejorar la técnica, y, por otra, hay que empezar a

memorizar cientos (¡y hasta miles!) de patrones en el mástil: intervalos, acordes, inversiones de acordes, arpegios, escalas, modos, etcétera.[1]
La memorización de estos patrones se desarrolla actualmente por la aburrida ,pero necesaria, táctica de la repetición.

Por otro lado, la vida actual no nos permite dedicar tanto tiempo a nuestras aficiones como nos gustaría y, por si fuera poco, pasamos más tiempo realizando desplazamientos del que desearíamos. En mi caso particular, más de dos horas diarias de autobús para desplazarme entre trabajo y hogar.

Es así como surge la idea del proyecto que presentamos: ¿Existe una forma de aprovechar ese tiempo en el que no podemos tener una guitarra para mejorar nuestro conocimiento del instrumento? ¿Es posible incorporar nuevo conocimiento jugando? ¿Se pueden adquirir patrones musicales a través de una aplicación móvil que después podamos aprovechar tocando con una guitarra de verdad en nuestras manos?

Existen ya multitud de estudios que avalan el aprendizaje a través de juegos informáticos y este proyecto quiere ser una constatación más en esta dirección.[2]

Creo, en resumen, que es factible separar la técnica del instrumento de la elaboración de los patrones mentales necesarios para tocarlo. De esta forma, podremos aprovechar múltiples momentos a lo largo de nuestra jornada para mejorar como músicos sin más que disponer de un dispositivo móvil.

2. Objetivos del Trabajo

Con el presente trabajo fin de máster se persiguen los siguientes objetivos:

[1] Basta con echar un vistazo a Hal Leonard (2011) “Chord, Scale & Arpeggio finder…” [5]

[2] Por citar algunos: Richard E. Mayer (Enero 2019) Computer Games in Education [6]; Margoudi, Maria (Octubre, 2016) Game-Based Learning of Musical Instruments [7]; Baratè, A (2013) Development of Serious Games for Music Education. [8]

2.1. Objetivos académicos

1.- Profundizar en el conocimiento de desarrollo de aplicaciones en entorno *iOS* con el lenguaje *Swift* en el *IDE XCODE*.

2.- Conocer el *framework SpriteKit* para el desarrollo de juegos en 2D.

3.- Conocer las posibilidades de socialización de una *app* a través del *Game Center* de *Apple y su framework GameKit*.

4.- Estudiar y aplicar el entorno de almacenamiento en la nube de *Apple* para almacenamiento tanto público como privado.

2.2. Objetivos de la aplicación principal

1.- Asimilar los patrones musicales necesarios para el aprendizaje de los instrumentos de cuerda, centrándonos en el caso de la guitarra.

2.- Conocer la interválica musical en cada uno de los patrones aprendidos.

3.- Como objetivo secundario, la *app* facilitará el reconocimiento auditivo de intervalos musicales.

La premisa que subyace a todo el proyecto es que las estructuras mentales que se desarrollan al tocar un instrumento se pueden generar incluso en ausencia del mismo.[3]

2.3. Objetivos de la aplicación secundaria

A la hora de afrontar el desarrollo de la *app* objeto de este proyecto, es necesario realizar un trabajo de almacenamiento previo de los patrones musicales. La codificación manual de dichos patrones en estructuras de datos informáticas puede ser laboriosa y sujeta a errores. Para facilitar dicha labor, surge la idea de construir una aplicación adicional que permita definir los patrones sin más que tocar sobre el mástil virtual las notas seleccionadas.

Desarrollar esta posibilidad permitirá, además, incorporar a la *app* principal una nueva funcionalidad por la que el usuario podrá añadir sus propios patrones.

[3] Para entender mejor el objetivo que se persigue se incluye el Anexo I: Fundamentos del conocimiento teórico musical empleado. Este conocimiento será uno de los pilares en el diseño de la aplicación. Si se desea profundizar sobre el tema recomendamos acudir a Fulqueris, S (2015) [9]

3. Estudio del estado del arte

En la actualidad las apps dedicadas al aprendizaje de la guitarra se pueden clasificar en tres categorías:

3.1. Aplicaciones basadas en aprendizaje vicario

Estas aplicaciones son una recopilación de vídeos cuyo gran valor es el de utilizar profesionales reconocidos. Entre las más destacadas citar *Guitar Lessons*[4] y *Guitar Tricks*[5].

3.2. Aplicaciones que se limitan a mostrarnos esquemas de patrones y las relaciones entre los mismos.

Por su naturaleza son aplicaciones orientadas a la consulta. Son el equivalente a los libros diccionario de acordes y escalas, con la gran ventaja de no ocupar espacio y de ofrecer una navegación rápida en conceptos relacionados en la estructura del conocimiento musical.

Algunas de ellas ofrecen también una generación de banda musical para que podamos practicar el patrón que nos muestra sobre una pista musical adecuada a dicho patrón.

Entre las mejores podemos nombrar a *Chord!*[6], *Guitar Jamm tracks*[7], *ChordBank*[8] *y Guitar Scales Power*[9].

3.3. Videojuegos interactivos

[4] Fender Digital. Guitar Lessons (2.2) [Aplicación móvil] Descargada de https://itunes.apple.com/us/app/guitar-lessons-fender-play/id1226057939

[5] Guitar Tricks Inc. Guitar Lessons - Guitar Tricks (1.3.13) [Aplicación móvil] Descargada de https://itunes.apple.com/us/app/guitar-lessons-guitar-tricks/id931639254

[6] Thomas Grapperon. Chord! (2.6.2) [Aplicación móvil] Descargada de https://itunes.apple.com/us/app/chord/id606691230

[7] Ninebuzz Software LLC. Guitar Jam Tracks - Scale Trainer & Practice Buddy (3.5) [Aplicación móvil] Descargada de https://itunes.apple.com/us/app/guitar-jam-tracks-scale-trainer-practice-buddy/id436102935

[8] Better Note, LLC. ChordBank - Guitar Chord App (3.21.5) [Aplicación móvil] Descargada de https://itunes.apple.com/us/app/chordbank-guitar-chord-app/id397602509

[9] Thomas Gunter. Guitar Scales Power (2.2.2) [Aplicación móvil] Descargada de https://itunes.apple.com/us/app/guitar-scales-power/id589291541

Las anteriores categorías son, salvo raras excepciones, cubiertas por desarrolladores individuales que algunas veces consiguen resultados interesantes.
Se aprecia un salto de calidad enorme cuando estos proyectos los asume una empresa con fines comerciales. Desafortunadamente en estos casos las aplicaciones están más enfocadas al juego que al aprendizaje.
Son aplicaciones en las que vemos las notas caminar por un mástil y nos indican en qué momento tocar cada nota. Estas aplicaciones son capaces de escuchar lo que tocamos y de corregirnos (es necesario tener una guitarra conectada).
Ofrecen el aprendizaje de temas musicales concretos sin preocuparse de las estructuras musicales subyacentes a los mismos.
Guitar Hero[10] y *Yousician Guitar*[11] son los grandes exponentes en este apartado.

4. Descripción del escenario

El escenario que se propone es el diseño y construcción de una aplicación que ayude en el estudio de los patrones musicales utilizados en el estudio de la guitarra.

4.1. Requisitos del sistema

El sistema afrontará los siguientes requisitos:

a) La *app* ofrecerá un estudio interactivo de los patrones musicales.
b) El estudio de los patrones estará organizado por niveles de dificultad.
c) El estudio de un patrón se dividirá a su vez en diferentes patrones: de un nivel básico en el que se ofrecerá toda la información del patrón al usuario a sucesivos niveles en los que se irá eliminando información sobre el patrón.

[10] Activision. Guitar Hero Live. [Aplicación móvil] Descargado de https://www.guitarhero.com

[11] Yousician Ltd. Yousician Gutar (3.2.0) [Aplicación móvil] Descargado de https://itunes.apple.com/us/app/yousician-guitar-piano-bass/id959883039

d) La aplicación tiene que incorporar sonido, de forma que el oído del usuario se desarrolle también al tiempo que se adquieren los patrones musicales.
e) El estudio de un patrón se planteará como un juego en el que se propondrá al estudiante el intervalo a identificar para que lo localice en un esquema presentado en la pantalla del dispositivo móvil.
f) Se premiará la rapidez en la identificación del patrón.
g) La *app* ofrecerá una información básica de patrones a estudiar organizados diferentes niveles de dificultad.
h) La *app* permitirá añadir patrones propios.
i) Estos patrones de usuario podrán ser compartidos con otros usuarios [12]
j) Se realizará una gamificación de la *app* basada en una lista de puntuación compartida, así como en premios que el usuario puede ganar al llegar a ciertas puntuaciones.

4.2. Componentes hardware

La *app* se desarrollará para teléfonos móviles *iPhone*. Puesto que se realizará la simulación de una sección de mástil, la *app* tendrá que funcionar en formato horizontal para una óptima visualización de la representación.
La *app* también funcionará en *iPad* pero no se realizará una presentación personalizada por la limitación de los tiempos en la elaboración del proyecto.

4.3. Componentes software

En el desarrollo de la aplicación se integrarán los siguientes elementos software:

a) Se utilizará en el desarrollo el *framework SpriteKit*, *framework* orientado al desarrollo de juegos en 2D [10].
b) Para la gamificación de la aplicación se empleará el *framework GameKit* perteneciente al *Game Center* de *Apple*.

[12]Quizás no se pueda afrontar por cuestiones de tiempo.

c) Para el almacenamiento de los datos en la nube se utilizará el *backend iCloud* de *Apple* a través del *framework CloudKit*.

5. Enfoque y método seguido

Se va a desarrollar una aplicación nativa en entorno *iOS* para *iPhone*, decisión marcada por los objetivos académicos del proyecto. Si el proyecto estuviera orientado a fines comerciales, nos hubiéramos decidido por entornos multiplataforma como *Unity* o *Unreal,* ya que nos permiten publicar con un único desarrollo en una gran variedad de dispositivos.

El desarrollo se realizará siguiendo una metodología ágil. Pensamos que, dada la naturaleza del proyecto, es la que más se ajusta para tener un ciclo de desarrollo rápido y flexible.

Al realizarse el desarrollo al tiempo que aprendemos una tecnología nueva es necesario aplicar una metodología que nos enfrente a posibles problemas tan pronto como sea posible. Además, la experiencia adquirida en cada ciclo puede ayudarnos a orientar el proyecto en el siguiente ciclo.

La idea fundamental de esta metodología es la de conseguir prototipos lo antes posible para irlos mejorando a posteriori.

En cada ciclo se realizarán las siguientes fases:

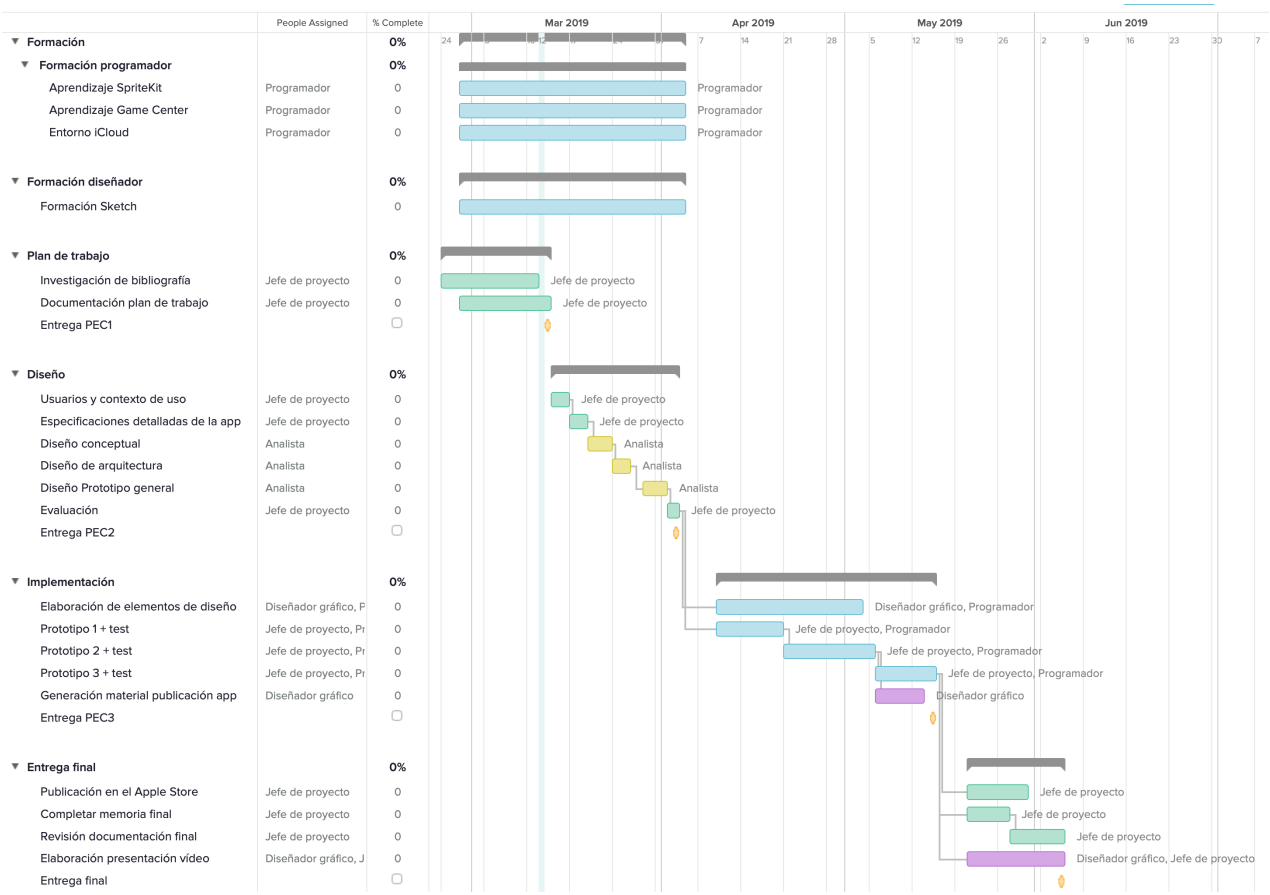
a. Planificación: identificación de recursos y tiempo.
b. Requisitos: definición de requisitos tanto funcionales como no funcionales.
c. Diseño: Definir diseño de la aplicación y patrones a utilizar.
d. Implementación: creación de una app prototipo resultado del ciclo.
e. Pruebas: Se realizarán tanto pruebas unitarias para garantizar el buen funcionamiento del código como pruebas de integración contextualizadas que garanticen una experiencia de usuario de calidad.

Al ser nosotros mismos nuestros clientes y equipo de desarrollo, y al tener una planificación marcada por los tiempos propios de un proyecto fin de máster, las fases de requisitos y diseño permanecerán bastante estables en todos los ciclos, mientras que el resto serán variables a lo largo de la construcción de los diferentes prototipos.

6. Planificación del proyecto

6.1. Planificación inicial

Se presenta a continuación la estimación de tareas que se establece a priori. Es importante destacar que es una planificación general. Según se avance en las siguientes fases del proyecto será necesario afinar y desglosar las tareas generales en tareas concretas.



Planificación de tareas con TeamGantt

A la hora de realizar la planificación se ha estimado el tiempo de trabajo disponible en tres horas diarias de lunes a domingo. Esta asignación es compatible con la jornada laboral y debería ser suficiente para conseguir los objetivos propuestos. Si se observaran desviaciones en algunas tareas se podrá dotar al proyecto de más horas en los fines de semana.

La planificación del proyecto se realiza con la herramienta online *TeamGantt* (teamgantt.com) para la gestión de proyectos. Esta herramienta se integra con Trello (trello.com), que se utiliza como herramienta de pizarra para facilitar el seguimiento y estado de cada tarea incorporada al proyecto.

Las tareas establecidas en la planificación son las siguientes:

Lista de tareas, asignación de tiempo y de roles desempeñados

Tarea	Horas	Roles asignados
Formación		
Aprendizaje SpriteKit	55h	Programador
Aprendizaje Game Center	20h	Programador
Aprendizaje iCloud (CloudKit)	15h	Programador
Diseño con Sketch	15h	Diseñador
Plan de trabajo		
Investigación de bibliografía	10	Jefe de proyecto
Documentación plan de trabajo	20	Jefe de proyecto
Diseño		
Usuarios y contexto de uso	6h	Jefe de proyecto
Especificaciones detalladas de la app	5h	Jefe de proyecto
Diseño conceptual	8h	Analista
Diseño de arquitectura	8h	Analista
Diseño prototipo general	15h	Analista
Evaluación	4h	Jefe de proyecto, Analista

Tarea	Horas	Roles asignados
Implementación		
Elaboración de elementos de diseño		Diseñador gráfico
Prototipo 1 + test	40h	Jefe de proyecto, programador
Prototipo 2 + test	80h	Jefe de proyecto, programador
Prototipo 3 + test	50h	Jefe de proyecto, programador
Generación material publicación app	5h	Diseñador gráfico
Entrega final		
Publicación en el Apple Store	3h	Jefe de proyecto
Completar memoria final	23h	Jefe de proyecto
Revisión documentación final	8h	Jefe de proyecto
Elaboración presentación vídeo	26h	Diseñador gráfico, jefe de proyecto

6.2. Planificación tras la realización del diseño

Después de haber realizado el diseño y la arquitectura de la aplicación, hemos definido y modificado las tareas implicadas en la fase de implementación. De esta nueva definición de tareas hemos sacado dos conclusiones que afectan a la nueva planificación:

1) Vamos a centrarnos en realizar un único juego en dos modalidades (secuencial y aletoria). La *app* final será más rica cuantos más variación de juegos tenga en el estudio de un patrón. La elaboración de estas variaciones será muy simple, ya que todos los elementos subyacentes a las mismas ya estarán desarrollados en el que planteamos. Renunciamos a esta variedad en al *app* para poder ajustarnos a la planificación.

2) Desplazamos la tarea de generación del material para la publicación de la app para la fase de entrega final, ganando así unas horas en la fase de implementación.

A continuación mostramos el diagrama *Gantt* de la fase de implementación y la nueva distribución de cargas de trabajo:

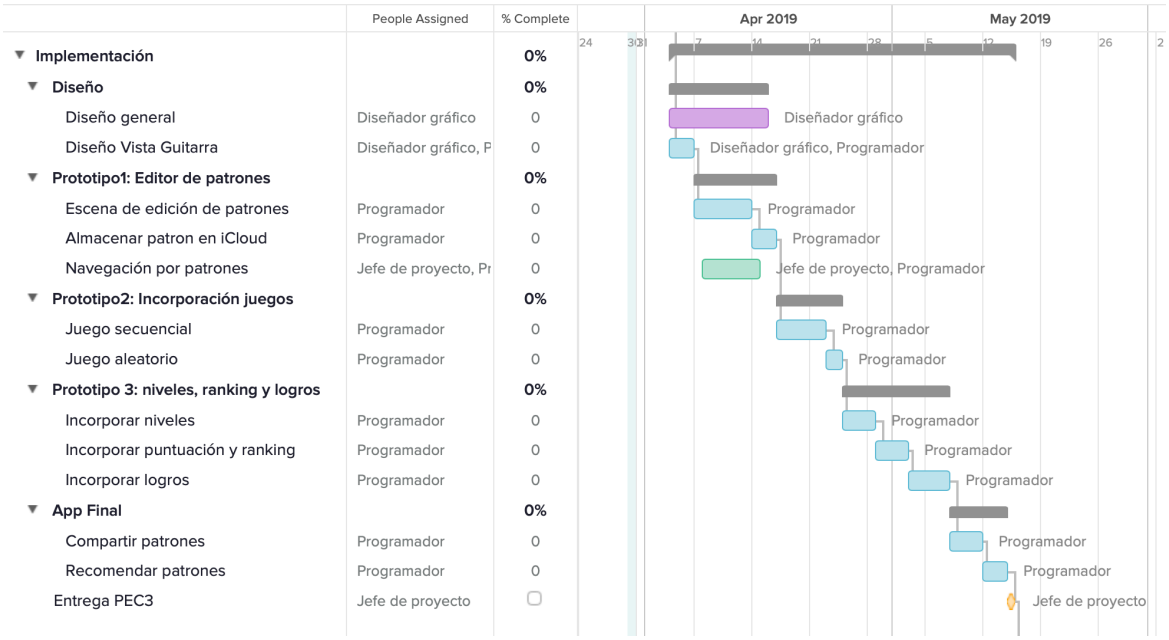


Diagrama Gantt de la fase de implementación

siendo la asignación de tiempos a las tareas identificadas:

Lista de tareas, asignación de tiempo en la fase de implementación

Tarea	Horas
Implementación	
Diseño general	8h
Diseño Vista Guitarra	15h
Editor de patrones	28h
Almacenamiento de patrones en iCloud	12h
Navegación por patrones	15h
Juego en modo secuencial	25h
Juego en modo aleatorio	6h
Incorporación de niveles	10h
Puntuación y ranking	16h
Logros	12h
Compartir patrones	10h
Recomendar patrones	10h
Revisión y entrega PEC3	3h

Se sigue manteniendo, al igual que en la planificación inicial, una asignación de tres horas diarias de lunes a domingos.

6.3. Modificaciones a la planificación durante la implementación

Durante la fase de implementación se ha modificado ligeramente el plan trazado. Las desviaciones afrontadas son las siguientes:

1. Se añade la tarea de localizar la app añadiendo el idioma inglés. La localización ha implicado modificación del código y la creación de los ficheros con las traducciones. Además los logros y los retos también se han tenido que tocar para incorporar la información en inglés. (3h)
2. Se elimina la realización del modo secuencial por carecer de interés una vez probado. No se produce un ahorro de tiempo significativo.
3. Documentar la fase de implementación: 14h.
4. Alimentación de la base de datos pública de la app. Es necesario incorporar un mínimo de acordes, arpegios y escalas para proporcionar una funcionalidad completa de la aplicación. Se ha incorporado todos los acordes, arpegios y escalas relacionados con el modo Jónico: 1h.

Se establece así un déficit de 12 horas que se ha cubierto ampliando 6 jornadas de trabajo a cinco horas, en lugar de las 3 que se han mantenido a lo largo del proyecto.

7. Productos obtenidos

Como resultado del proyecto se genera una aplicación móvil para dispositivos móviles con sistema operativo *iOS 11* o superiores. Esta aplicación, en la fecha en que ultimamos la memoria, está disponible en su versión *1.8* en el *App Store*[13]. Puesto que se concibe este proyecto como el embrión de algo más grande es muy probable que en fechas futuras se acceda a versiones superiores que incluyan una mayor funcionalidad y cambios de diseño.

[13] Puede descargarse desde un dispositivo iOS en https://itunes.apple.com/us/app/guitar-patterns/id1460697058?mt=8

Como producto adicional se ha generado una *app* para alimentar la base de datos pública que se utiliza en el proyecto. Esta aplicación no es más que un subconjunto de la anterior: el editor de patrones que el usuario utiliza para incorporar patrones a su base de datos privada se ha modificado para que dichos patrones se graben directamente sobre la pública.
Este producto no está disponible ya que no se distribuye, es de uso exclusivo para el gestor propietario de la aplicación y se utilizará para aumentar progresivamente los patrones disponibles para el usuario.

8. Estructura de la memoria

La memoria se estructura en ocho apartados principales:

1. *Diseño centrado en el usuario*, donde desde la definición de los usuarios y el contexto de uso de la *app* concretamos el diseño de la aplicación y la estrategia para llevarla a cabo. Utilizando una metodología ágil se realizan distintos ciclos que van afinando lo que será la aplicación final a través de diferentes prototipos y su evaluación.
2. *Fase de desarrollo*, en la que se abordan los aspectos propios de la programación de la aplicación. Se define la estructura del código y se hace hincapié tanto en las decisiones tomadas como en aquellas herramientas y/o procesos que nos han ayudado en la mejora del código programado.
3. *Fase de pruebas*, donde hablamos de las pruebas que se han permitido corregir y mejorar la aplicación. Desde pruebas funcionales unitarias hasta pruebas integrales realizadas con usuarios de diferente índole. Un punto importante en esta fase es el que atañe a la usabilidad de la *app*, fundamental para conseguir un buen resultado.
4. Puesta en producción, apartado en el que se documenta la puesta en producción de la aplicación. El *backend* que *Apple* ofrece a través de su servicio de *iCloud* tiene ciertas peculiaridades que hemos considerado necesario documentar.
5. *Futuro de Guitar Patterns* nos ofrece una perspectiva de lo que será el futuro de la aplicación. A lo largo del desarrollo del proyecto surgen

nuevas ideas que no se pueden afrontar en los tiempos marcados pero sí queremos dejar constancia de las mismas.

6. *Conclusiones*, donde hacemos una evaluación final del proyecto y ofrecemos las conclusiones que obtenemos comparando nuestras expectativas iniciales con el resultado final enriquecidas con todo el conocimiento adquirido por el camino.
7. *Anexos*, se ofrece dos anexos. El primero consiste en una pequeña introducción a la teoría musical que puede permitir entender mejor el proyecto a alguien sin ningún conocimiento sobre esta materia; en el segundo se describe brevemente aquellas herramientas informáticas que hemos utilizado en el desarrollo de este *TFM*.
8. Además, naturalmente, se ofrece un glosario y una bibliografía con todo el material que hemos necesitado estudiar y/o consultar en la realización del proyecto.

DISEÑO CENTRADO EN EL USUARIO

1. Usuarios y contexto de uso

A continuación se muestran los dos usuarios sobre los que se ha basado el diseño de la aplicación. Se pueden reconocer dos roles bien diferenciados que no son excluyentes: el rol de la persona que quiere incrementar sus conocimientos del instrumento sea profesional o amateur, y el rol del profesor que enseña a sus alumnos a tocar el instrumento.

Ficha de usuario - Rol: estudiante

Ficha de usuario - Rol de profesor

2. Diseño conceptual

2.1. Escenarios de uso

Para tener una idea del uso en contexto de la aplicación definimos dos escenarios de uso:

Escenario alumno

Manolo coge un autobús interurbano para ir a su trabajo. En su tiempo de autobús (dos horas diarias) aprovecha para escuchar inglés, descubrir nueva música y estudiar alguna escala musical que le permita avanzar con sus

estudios de guitarra. En estos momentos está aprendiendo a '*moverse*'[14]con la escala de blues mayor, por lo que entra en la aplicación, selecciona dicha escala y empieza a hacer los ejercicios que la *app* le propone. Tiene un compañero de clase que también la utiliza, y se ha propuesto batir su puntuación y establecer un nuevo récord. Además ve que tiene una nueva sugerencia de su profesor, quien le ha invitado a estudiar una posición de arpegio menor con la que harán ejercicios de improvisación en la siguiente clase. Aprovecha sus últimos veinte minutos de viaje para jugar con el arpegio.

Cuando Manolo llega a casa y hace su hora de guitarra, empieza a improvisar con dicho arpegio: harán falta muchas horas para desarrollar velocidad y precisión, pero parece que el esquema ¡ya lo tiene en su cabeza!

Escenario profesor

Noelia comienza su clase de guitarra. Está introduciendo a sus alumnos en la improvisación dentro del rock y les está presentando la escala pentatónica menor. Para eso abre la aplicación en su modo editor y les muestra a los alumnos cómo se forma cada una de las posiciones. Puesto que esa escala está en la base de datos de la *app,* les recomienda estudiarla. Pero, además, considera interesante que ejerciten con otros patrones importantes y que, al menos de momento, no están en la *app*. Les dice que les enviará estos patrones. Una vez en su casa, entra en el editor de patrones para crearlos y compartirlos con sus alumnos. ¡Ya tienen tarea más que de sobra para la siguiente clase!

2.2. Mapa de experiencia

A partir de lo anterior se ha conformado un mapa de experiencia en el que se identifican cuatro fases diferentes, que a su vez constan de una o varias acciones.

[14] Con el término informal 'moverse' los guitarristas se refieren al desplazamiento de los dedos por las notas de un patrón a lo largo de todo el mástil.

Las fases identificadas son las siguientes:

Fase de Selección de objetivos
Es aquella en la que el usuario navega por el conocimiento de la aplicación -ya incorporado o previamente generado por el mismo u otros usuarios- y selecciona el que será el objetivo de estudio.

Fase de Creación de contenidos
En esta fase el usuario puede generar e incorporar contenido propio a la aplicación.

Fase de Estudio
Es la fase más importante de la aplicación. Implica un trabajo activo de un patrón determinado. Dicho trabajo requiere una retroalimentación que dirija la adquisición del conocimiento del alumno. Se identificará un sistema de ranking y logros para la gamificación de la *app* para motivar al estudiante.

Fase de Compartición
Es aquella en la que un usuario decide recomendar algún patrón a otro usuario.

Mapa de experiencia							
FASES	SELECCIÓN DE OBJETIVOS		CREACIÓN DE CONTENIDOS	ESTUDIO			COMPARTIR
ACCIONES	BÚSQUEDA	RECOMENDACIONES	EDITAR PATRON	TRABAJO ACTIVO	RETROALIMENTACIÓN	LOGROS	COMPARTIR
Objetivos	Buscar el patrón a estudiar.	Ver qué patrones nos han recomendado.	Creación de nuestros propios patrones.	Jugar con el patrón seleccionado.	Conocer nuestra evolución en el aprendizaje.	Motivarnos para seguir mejorando en la adquisición de nuestro conocimiento.	Compartir o aconsejar patrones a otros usuarios.

Expectaciones	Encontrar los principales patrones musicales de la guitarra.	Conocer que patrones nos recomiendan personas que saben más que nosotros.	Poder incorporar patrones personalizados.	Memorizar patrón y los intervalos relacionados. Interiorizar la sonoridad de los diferentes intervalos musicales.	Constatar mejoras en los patrones trabajados.	Constatar nuestros progresos en el aprendizaje.	Poder enviar recomendaciones a alumnos / amigos.
Pensamientos	¿Están todos los patrones que necesito?	¿Cuá es el patrón que me vienen bien aprender en el punto en el que estoy?	¿Puedo sugerir este patrón para que se incorpore a la base de datos general?	Sería maravilloso poder incorporar nuevos patrones sin necesidad de tener la guitarra presente.	Parece que el tiempo que le dedico a los patrones está funcionando.	¡Quiero mejorar mi récord! ¡Quiere llegar al nivel de estrella de rock en este patrón!	Sería interesante que este chico estudiase bien este patrón.
Funciones	Selección de patrónes.	Ver los patrones recomendados y seleccionar aquel que queramos practicar.	1. Radar de novedades. 2. Actividad de los amigos. 3. Descubrimiento semanal.	Trabajo secuencial del patrón. Trabajo aleatorio del patrón en niveles de dificultad creciente.	Visualización de la puntuación y la situación en el ranking.	Ranking compartido y gestión de logros.	Compartir un patrón determinado. Enviar mensaje de presentación.
Interacción y contexto.							
Lugar	Transporte público	Transporte público	Casa / Escuela	Autobús	Autobus	Autobús	Transporte público
Tiempo	mañana	mañana	Tarde	Mañana	Mañana	Mañana	Mañana
Atención	Parcial (8)	parcial(8)	Parcial (9)	Total	Parcial(8)	Parcial (8)	Parcial(6)
Social	Solo	Solo	Solo o con profesor	Solo	Grupo	Solo con redes sociales	
Dispositivo	Móvil	Móvil	Tablet	Móvil	Móvil	Móvil	
Eventos	Se selecciona la figura musical a prácticar.	Selección de la recomendación a trabajar.	Definir nuevo patrón.	Juego con patrón.	Notificación del progreso. Evolución en ranking.	Información de logros obtenidos.	Notificar patrón compartido. Informar aceptación de patrón.

Decisiones	Decidir patrón concreto a practicar.	Decidir patrón concreto a practicar.	Definir datos de identificación del patrón.	Decidir notas correctas en el patrón marcado.	¿Seguimos estudiando este patrón o cambiamos a otro?	Seguir para mejorar logros o cambiar de objetivo.	Qué patrón compartir y con quién.
Problemas	Es necesario tener datos si no se ha descargado previamente la base de datos de conocimiento.	Es necesario tener datos para actualizar las últimas recomendaciones.	Pueden existir patrones duplicados.	El juego puede ser interrumpido por llamadas o recepción de mensajes.	Necesario tener datos para acceder a logros y ranking.	Comparar con gente de mucho más nivel puede desmotivar.	Con quien queremos compartir podría no estar dado de alta como usuario en la app.
Oportunidades	Descubrir nuevas digitaciones que no se conocían. Informar del tiempo dedicado a los patrones que ya se han trabajado. Informar de novedades.	Informar al usuario de si ya ha trabajado o no esas digitaciones y el nivel que tiene en las mismas.	Identificación de duplicados. Informar de patrones similares de posible interés. Asociar patrones a temas musicales.	Desarrollo del oído musical del estudiante. Realizar ejercicios basados en reconocimiento de sonido.	Posibilidad de conseguir nuevos logros y avances.	Poner en contacto a guitarristas de niveles similares.	Enviar invitación a personas que no están en la app para que se den de alta. Crear vínculos.
Estado emocional	Motivado	Motivado	Concentrado	Concentrado	Satisfecho	Ilusionado	Generosidad

2.3. Funcionalidad ampliada

De todo lo anterior surgen nuevas ideas que resultan interesantes para enriquecer el proyecto. De las oportunidades identificadas se nos ocurren las siguientes funcionalidades añadidas:

a) Información automática de novedades en la base de datos.
b) Creación de un histórico de trabajo del estudiante en el que se muestre su dedicación y se le recomiende reforzar conocimiento.
c) Posibilidad de asociar patrones con temas musicales concretos. De esta forma el estudiante obtiene una información adicional muy práctica sobre la que desarrollar las estructuras aprendidas.
d) Ampliar la tipología de ejercicios sobre patrones para incorporar ejercicios en los que el sonido no forme parte del estímulo respuesta, sino del

estímulo pregunta. Esto permitirá llegar a un desarrollo mayor del oído musical.

e) Definir cadenas de logros más complejas y personalizadas para los estudiantes.
f) Generar socialización al poner en contacto a usuarios con similar nivel.
g) Posibilidad de enviar invitaciones para descargar la *app*.

Estas nuevas funcionalidades no se abordarán en el proyecto actual, salvo que en la planificación realizada se le puedan añadir más horas de trabajo.

3. Estrategia de diseño

A continuación exponemos la estrategia de diseño con la que afrontamos el proyecto y establecemos la esencia del mismo.

3.1. Descripción de la estrategia de diseño

El producto final está pensado para ecosistemas basados en los dispositivos núcleo, aunque no sería descabellado ampliar el proyecto a un entorno web. La dinámica del proyecto explota al máximo la capacidad de interacción del usuario con la pantalla táctil, por lo que centrarnos en móviles y tabletas es más que suficiente. Otro motivo para esta decisión es que se trata de buscar tiempo en desplazamientos o en lugares donde no tengamos una guitarra, y es probable que en dichos lugares tampoco dispongamos de un ordenador.
Aunque, tal y como ya se ha comentado, el proyecto se desarrollará en entorno *iOS*, en un futuro será interesante realizar el desarrollo para *Android* , o explorar las posibilidades de un desarrollo multiplataforma con los entornos de desarrollo de videojuegos *Unity* o *Unreal* (por citar alguno).

Los conceptos centrales que mejor definen la *app* que se plantea son:

Gamificación: como punto de motivación que anime a la gente a dedicar horas a los ejercicios y fomente la colaboración en la comunidad de usuarios.

La idea es la de premiar la uso mediante puntos que permitirán conseguir logros y/o desbloquear otros niveles.

Sincronización: la información estará sincronizada en todos los dispositivos. Un patrón dado de alta en el móvil estará inmediatamente disponible a través de la tablet. Para esto será fundamental utilizar un *backend* que permita sincronizar la información.

Communificación: con la idea de crear una comunidad, se buscará utilizar el formato de red social a la hora de crear, compartir, comentar y valorar los patrones musicales. Es un concepto que nos parece fundamental para el éxito de la aplicación, aunque será el concepto a sacrificar en caso de falta del tiempo.

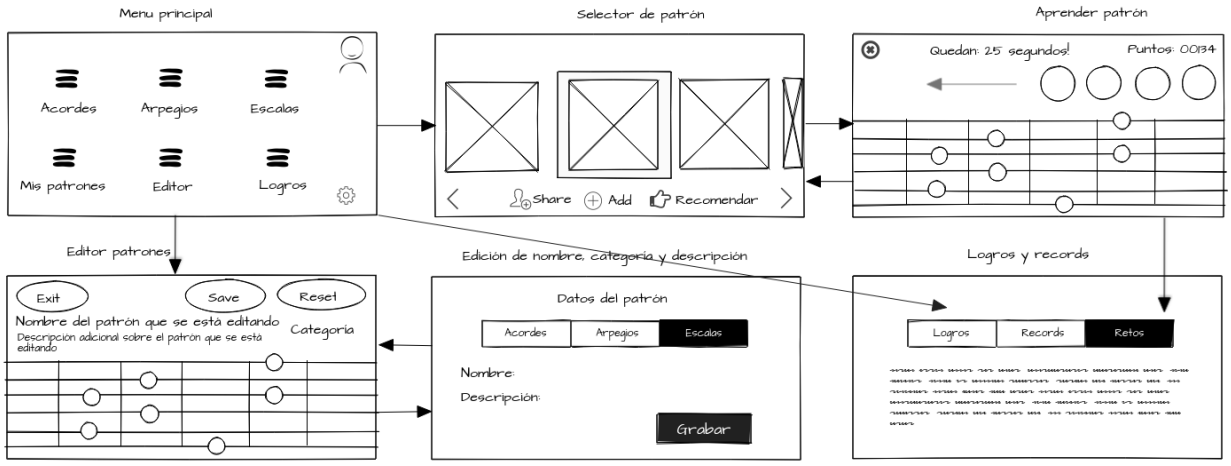
3.2. Definición de la aplicación

Después de todo lo expuesto, definimos *Guitar Patterns* como "La aplicación que jugando te ayuda a mejorar como guitarrista sin necesidad de tener una guitarra a mano."

4. Prototipado

A continuación se muestra tanto el *wireframe* o prototipo de bajo nivel como un prototipo más detallado y real de la *app* o prototipo de alta definición.

4.1. Wireframe



Wireframe de la app Guitar Pattern

Las pantallas del prototipo de baja definición representan los principales recorridos del usuario por la aplicación:

El *menú principal* permite acceder a todas las posibilidades de uso de la *app*, incluido el acceso a la información del usuario y a los elementos de la configuración. Se opta por un menú a pantalla completa en la que el usuario tiene acceso a toda la información desde que entra en la aplicación.
Las opciones que se utilizarán con mayor probabilidad se han puesto en la parte superior del menú, aunque esto dependerá del uso que se le dé a la aplicación, por lo que presentamos todas las opciones con igual peso gráfico.
Dentro de cada fila se ha escogido una presentación que sitúa a la izquierda los elementos probablemente más visitados.

La *pantalla de selección de patrón* es la pantalla a través de la cual el usuario escoge el patrón para trabajar, compartir o aconsejar. Es importante que en la selección se muestre gráficamente el patrón. Esto es así ya que muchas veces

patrones diferentes se identifican con el mismo nombre. Por ejemplo, un acorde *Maj7* puede tener más de cinco representaciones a lo largo del mástil.
Si el número de elementos en la base de datos aumenta significativamente ,será interesante incorporar filtros adicionales a la pantalla de selección.

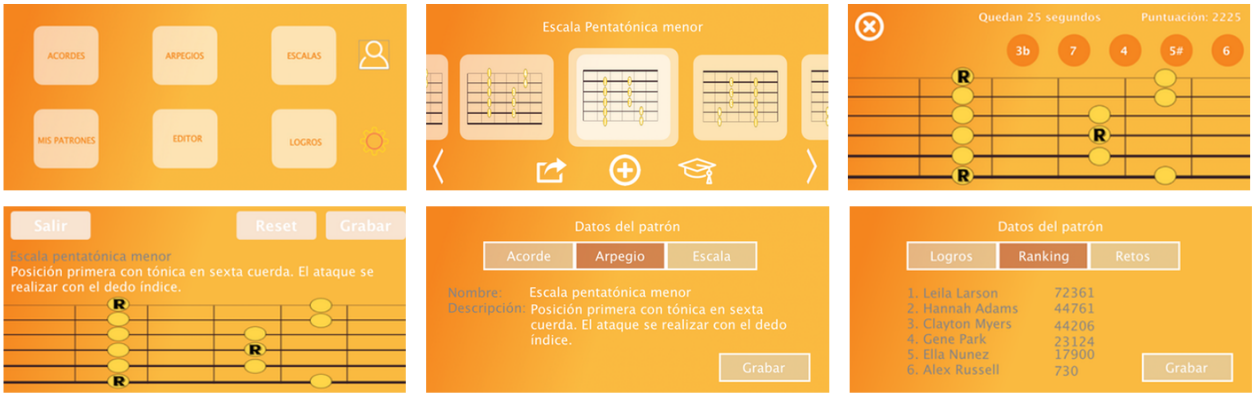
En la *pantalla de aprendizaje de patrón* nos encontramos con el estudio del patrón. Los intervalos comienzan a aparecer por la parte superior derecha de la pantalla. La velocidad de dichos intervalos se incrementará con el paso del tiempo. En la medida que el usuario reconoce y toca en la pantalla, los intervalos adecuados la fila de intervalos móviles disminuye en número.
Se trata así de forzar al estudiante a que identifique los diferentes intervalos que existen en la figura objeto de estudio. El aprendizaje se basa en la repetición de dicha identificación en condiciones cada vez más complicadas.

El *editor de patrones* es lo que nos permitirá crear una base de datos de patrones que alimentará la *app*. Además, permite al usuario crearse sus propios patrones. Es necesario que el estudiante/profesor pueda crear patrones personalizados.

La *pantalla de edición de datos del patrón* permite nombrar y describir el patrón y su tipología. Se trata aquí de definir los metadatos del patrón añadido para su posterior identificación. El usuario puede también incorporar instrucciones para su uso.

La *pantalla de logros y récords* es la pantalla que ofrece la información del *ranking* de puntuación y de los logros obtenidos, así como de los diferentes retos superados. La gamificación es una parte importante que llevará al estudiante a la repetición de los ejercicios por lo que está opción aparece también en el menú principal con la misma importancia gráfica que el resto.

4.2. Prototipo de alta definición



Prototipo de alta definición de Guitar Patterns

En la imagen sobre el párrafo se ofrece el prototipo de alta definición en el que se aplica ya una idea de diseño inicial y se ofrece una mayor definición de la información de la *app*.

Como se puede apreciar, el pilar más importante de la *app* será la creación de un objeto que represente y actúe como mástil de una guitarra en aquellos aspectos que nos interesan. Así, el componente desarrollado tiene que poder generar una imagen que se utilice como selector de patrón, tiene que permitir la interacción necesaria para jugar en los diferentes ejercicios propuestos y, además, tiene que permitir su edición a la hora de elaborar los patrones.

Es deseable desarrollar un único componente que incorpore toda la flexibilidad descrita.

5. Evaluación

A partir de la elaboración del prototipo se establece un proceso de evaluación continua que comienza con una versión estática de las pantallas del proyecto y avanza a lo largo de la implementación del mismo con la realización de los diferentes prototipos.

Este proceso de evaluación se ha realizado con la participación de dos usuarios representativos de las dos categorías a las que nos dirigimos: estudiantes y profesores de guitarra.

5.1. Evaluación por parte de los usuarios

A continuación se presentan el resultado final en el que se convirtió el prototipo presentado después de la implementación. Explicaremos, grosso modo, las razones de algunos de los cambios en el diseño que se pueden apreciar.

Diseño final de la aplicación Guitar Patterns

Hay un cambio general de diseño que se aplica a todas las pantallas: se prescinde del color naranja como fondo de pantalla. Aunque al ver las imágenes nos sigue gustando más un diseño con naranjas degradados de fondo, cuando los evaluadores probaron la aplicación todos coincidieron en que complicaba la concentración. Por este motivo, se ha optado por un color neutro de fondo sobre el que destacan los elementos realmente importantes: los intervalos sobre el mástil del instrumento.

La pantalla con el *menú principal* se mantiene bastante fiel a sus orígenes. Desaparecen los botones más pequeños (configuración y datos de usuario), ya que los datos del usuario que juega con la aplicación serán los de su cuenta en el *Game Center,* donde puede definir tanto su nombre como apodo. El acceso a la configuración estaba pensado para abordar la personalización en la visualización, pero finalmente esta tarea no se ha acometido.

En la pantalla de *selección de patrón* se aprecian varios cambios. El más evidente es que se ha optado por botones simples con texto explicativo. Puesto que había espacio para situarlos, nos pareció una solución más efectiva. Otro cambio es que el menú puede variar en esta pantalla. Si se fija uno en las dos pantallas de selección de patrón que aparecen en la imagen (segunda de la primera fila y primera de la tercera fila) se puede comprobar que los menús difieren. Esto es porque al selector de patrón se puede llegar por dos caminos diferentes: el primero es desde el banco de patrones comunes a los que se accede desde las secciones de acordes, arpegios o escalas. Aquí se accede a un patrón público sobre el que no tenemos derecho de edición; lo único que se puede hacer con él es: bien jugar, bien incorporarlo a la sección *Mis patrones*. El segundo camino ocurre cuando seleccionamos un patrón de la sección *Mis patrones.* Sobre estos patrones sí tenemos derecho de escritura, lo cual nos abre las puertas a editar el patrón, eliminarlo, jugar con él, o compartirlo con otro usuario.

En cada pantalla puede que no todas las acciones estén disponibles en un momento dado. Nuestro estudiante evaluador nos solicita que si la acción no

está disponible, el botón aparezca difuminado. El evaluador profesor, por el contrario, piensa que es mejor resaltar una opción si es necesario aplicarla ,como es el caso de grabar cuando se han hecho modificaciones. Para evitar riesgos de omisión de acciones se avisará siempre al usuario con alertas sobre la posibilidad de perder datos.

Ambos evaluadores están de acuerdo en que falta información cuando se juega con el patrón: perdemos el nombre y la descripción. Es importante que aparezcan en pantalla para que se facilite la asociación del patrón con su correspondiente nombre. Por este motivo, en la pantalla de *aprendizaje del patrón* se visualizan ahora los metadatos del patrón (nombre y descripción) de forma que el usuario los tenga siempre presentes.

El evaluador alumno nos sugiere identificar el récord personal del usuario en pantalla como un elemento de motivación añadido. Nos parece interesante y se incorpora al diseño.

La pantalla de *edición de datos del patrón* no sufre variación alguna, salvo por el color de fondo, que se mantiene gris por coherencia con el resto de la aplicación.

Sobre las pantallas de puntuaciones y logros ha habido también sugerencias de cambio, pero puesto que se utilizan las pantallas por defecto definidas por *GameCenter* hubo poco que hacer. Lo único que pudimos hacer fue acceder a la petición de nuestro profesor evaluador de agrupar las puntuaciones por categoría. Puesto que el *GameCenter* permite agrupar las puntuaciones, esto se llevo a cabo sin dificultad alguna.
Es necesario comentar que estas pantallas se integran muy bien sobre cualquier diseño, ya que son muy limpias y cuentan con fondos con transparencias que permiten heredar los colores de nuestra *app*.

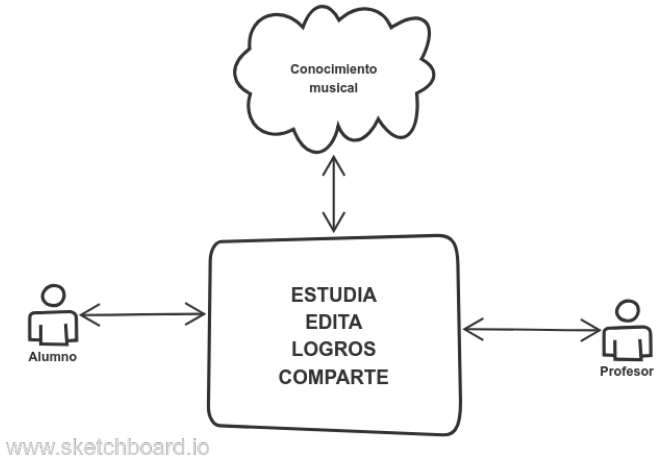
Con nuestros dos evaluadores principales centramos la estructura general de la aplicación. Como fueron también nuestros colaboradores en el diseño inicial de la misma, las variaciones fueron relativamente pequeñas.
Según se avanzó en el prototipo funcional de la aplicación se incorporaron más usuarios, para asegurarnos que construíamos una aplicación realmente útil en la que el usuario descubre de forma natural las funcionalidades principales de la aplicación.
La evaluación de estos usuarios, orientada a evaluar la aplicación como un todo, se puede ver en detalle en la narración de las pruebas por parte del usuario que documentamos en la sección de *Fase de pruebas*.

5.2. Casos de uso

Como se podrá ver, la idea de la *app* se basa sobre una estructura sencilla en la que se aprovecha al máximo las facilidades de desarrollo que nos brinda el entorno de desarrollo de *Apple*. Por ejemplo, no tenemos que pensar en las estructuras de las bases de datos de logros o *ranking,* ya que el *Game Center* nos lo da resuelto.

Hemos intentado encontrar formas de simplificar el desarrollo identificando *frameworks* que nos faciliten el trabajo. También, en algún caso, vamos a utilizar algún componente *open source* para optimizar el tiempo de desarrollo. La parte más negativa de esta opción es la dificultad de realizar una personalización completa de algunas de las pantallas de la *app*. Esta personalización puede realizarse pero a costa de invertir un mayor número de horas en el desarrollo.

5.2.1. *Esquema general de la aplicación*



Esquema general de la aplicación

Tal y como puede apreciarse en el gráfico, en la aplicación se identifican dos tipos de actores o roles: el de alumno, que utiliza la app para estudiar patrones musicales, y el de profesor, quien crea nuevos patrones y recomienda su estudio a otros alumnos. Estos roles no son estancos y dependen del uso que el usuario realizar de la *app*; es decir, un usuario que está estudiando patrones puede en un momento dado incorporar patrones que cree necesarios y recomendarlos a otro usuario de la *app*.

Los procesos básicos de la aplicación que un usuario puede realizar son:

a) Estudiar un patrón.
El usuario accede al patrón que desea estudiar y, a través del juego propuesto, adquiere el conocimiento musical del patrón seleccionado. Como resultado del proceso de juego/estudio el usuario obtiene una puntuación.

b) Editar un nuevo patrón.

A través de la edición el usuario puede incorporar nuevos patrones musicales. Los patrones pertenecerán a alguna de las categorías principales: escalas, arpegios, o acordes.

c) Conseguir nuevos logros.

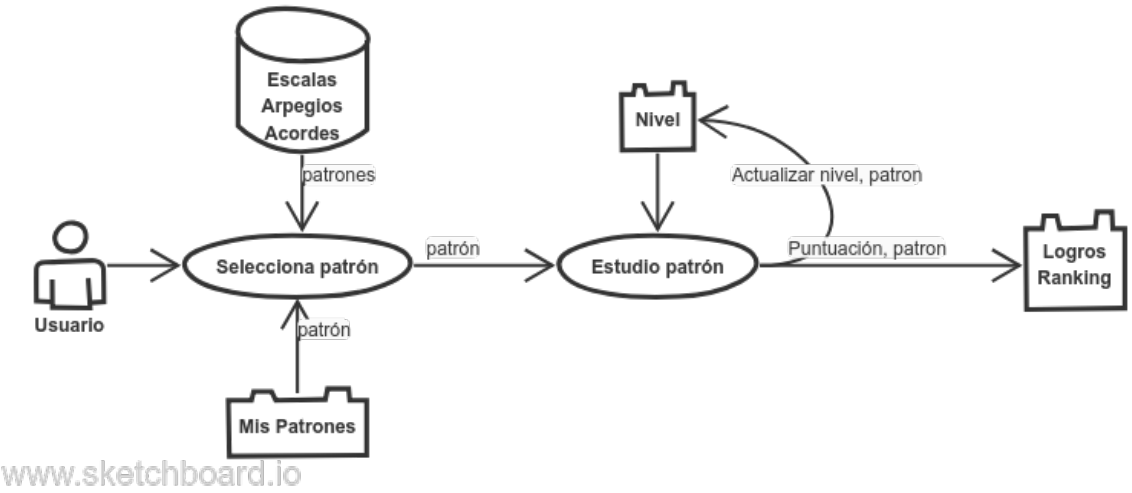
La mejora de resultados en los diferentes ejercicios permitirán un buen posicionamiento en el ranking de la aplicación y la adquisición de diferentes logros, algunos de los cuales pueden ser necesarios para acceder a otros contenidos.

d) Compartir o recomendar patrones.

Tanto los patrones que ya existen en la app como en aquellos nuevos que incorporan los usuarios, pueden ser compartidos (recomendados) con otros usuarios o añadirlos a nuestra lista personal de patrones para trabajar.

5.2.2. Procesos de la aplicación

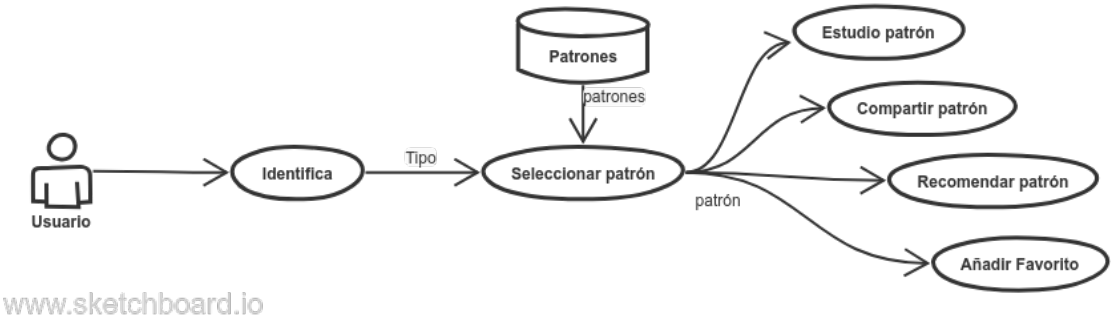
5.2.2.1. Proceso general de estudio



Proceso general de estudio

El proceso de estudio comienza con la elección de un patrón, ya sea de la base de datos de la *app* o de los patrones que previamente fueron escogidos por el usuario, o que le fueron recomendados por otro usuario.
Tras la selección del patrón objeto de estudio se pasa a la actividad interactiva que permite el estudio del patrón. Esta actividad tendrá un nivel de dificultad asociado y que depende de las puntuaciones obtenidas previamente. Como resultado final de los ejercicios se obtiene una puntuación que permitirá la actualización del nivel y/o la consecución de logros así como un posicionamiento dentro del *ranking* de la *app*.
Tanto los logros como los niveles se establecen para cada tipo de patrón.

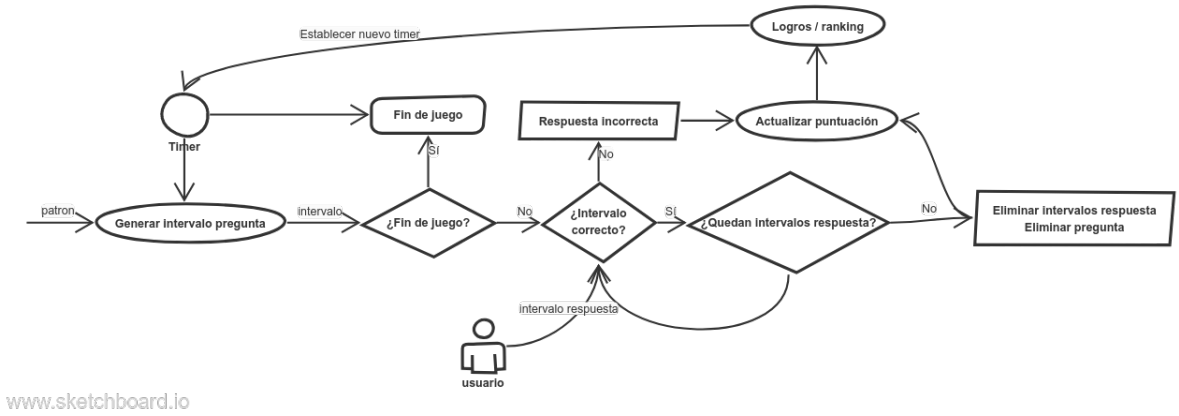
5.2.2.2. Proceso de selección de un patrón



Proceso de selección de un patrón

La selección de patrón se establece por tipo de patrón. El usuario elige el tipo de patrón y después selecciona uno específico ya sea para estudiar o compartir. En la versión final que se plantea se ha decidido trabajar sobre los tres tipos de patrones fundamentales de la música: escalas, arpegio y acordes. Trabajar sobre un conjunto de tipología definido y cerrado nos permite simplificar la navegación de la *app*. En futuras versiones sería muy interesante ampliar dicha tipología.

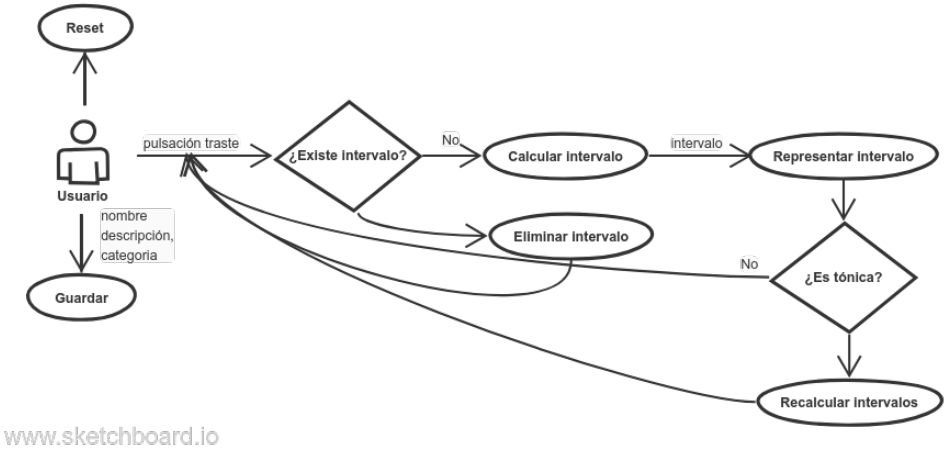
5.2.2.3. Proceso de estudio de un patrón



Proceso de estudio de un patrón

El proceso de estudio de un patrón propuesto es el de mostrar un mástil sobre el que se muestra la tónica y se le preguntan intervalos al usuario. Los intervalos pregunta aparecerán por la parte derecha de la pantalla y se irán desplazando hacia la izquierda. El usuario tiene que marcar el primer intervalo preguntado en el mástil. Hasta que la respuesta no sea la correcta el intervalo pregunta seguirá en pantalla desplazándose. Si en su desplazamiento alcanza el final de la pantalla del móvil se habrá perdido la partida. Los intervalos se generarán cada cierto tiempo, siendo este tiempo cada vez más pequeño para conseguir poner en apuros al usuario. Cuanto más resista el alumno, más puntuación conseguirá.

5.2.2.4. Proceso de creación y/o edición de un patrón



Proceso de edición de un patrón

En este proceso se le presentará al usuario un mástil vacío sobre el que puede marcar trastes tocando sobre ellos. Al marcar un traste se calculará el intervalo que representa. Para poder realizar un cálculo de intervalo tiene que existir una tónica, que es la nota de referencia sobre la que se calcularán los intervalos musicales. El criterio que se va a seguir es el siguiente: si pulsamos sobre un traste y en el mástil no existe una tónica entonces el traste seleccionado será la tónica del patrón.
Al igual que podemos añadir un intervalo sin más que pulsar sobre el traste, podemos también eliminarlo volviendo a pulsar sobre el.
Si el traste eliminado es tónica, el mástil se quedará sin tónica, por lo que el siguiente traste seleccionado será la nueva tónica del patrón. La aparición de una nueva tónica obliga a realizar un cálculo de todos los intervalos que ya existen en el patrón.

5.2.2.5. Proceso de compartir un patrón

Proceso de compartir un patrón

Para compartir un patrón basta con incorporar dicho patrón a la base de datos de conocimiento común. Se comparte con todos los usuarios de la aplicación.

5.2.2.6. Proceso de recomendar un patrón

Proceso de recomendación de un patrón

La recomendación de un patrón es una acción que se realiza de un usuario hacia otro usuario concreto. Se trata de una recomendación personalizada que se puede hacer sobre cualquier patrón que exista en la aplicación.

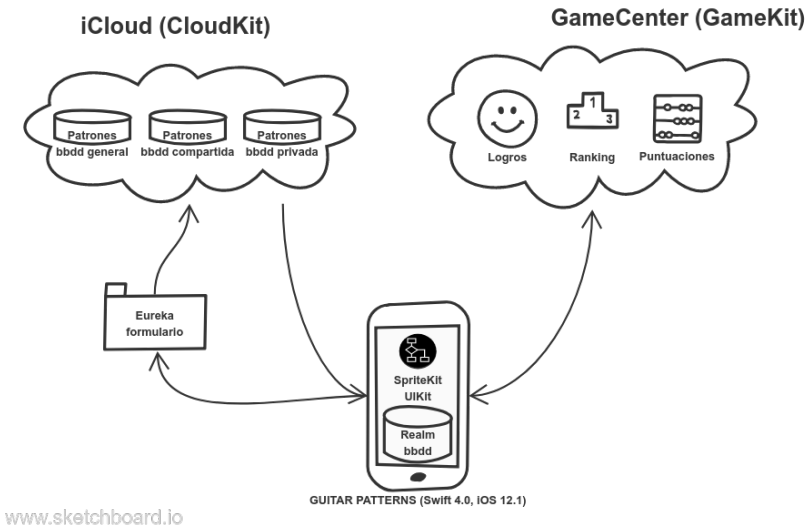
5.2.2.7. Proceso de añadir a favoritos (Mis Patrones)

Proceso de añadir patrón a favoritos

Añadir un patrón a los registros de usuario no es más que incorporar dicho patrón en una tabla en la que se almacenarán aquellos patrones que el usuario ha seleccionado para practicar y aquellos que le han sido recomendados. Es una forma rápida de que el alumno pueda acceder a los patrones que más le interesan en un momento determinado.

5.3. Arquitectura del sistema

5.3.1. Arquitectura general de la aplicación



Arquitectura general de Guitar Patterns

La aplicación *iOS* se desarrollará integrando *SpriteKit* con *UIKit*. *SpriteKit* es lo más adecuado a la hora de desarrollar un entorno dinámico de enseñanza a través del juego, sin embargo, hay elementos de *UIKit* que no se encuentran en *SpriteKit* y que pueden simplificar bastante el desarrollo. Esto nos va a dar la oportunidad tanto de conocer *SpriteKit* como de probar la integración de los dos *frameworks*.

Existen dos entornos de bases de datos: uno local y otro en la nube. La primera vez que se abra la aplicación será necesario tener conexión para

descargar de *iCloud* los patrones existentes. A partir de ese momento existirá una sincronización entre ambos esquemas.
No está claro que esta sincronización se pueda llevar a cabo en los plazos que se manejan, por lo que se comenzará con un funcionamiento que necesitará conexión continua a la nube.

El almacenamiento en la nube se realizará a través de *iCloud*, el entorno de base de datos en la nube propiedad de *Apple*. Este entorno está muy pensado para la complejidad de almacenamiento a la que se enfrentan las aplicaciones móviles y nos da resuelto el problema de tener bases de datos públicas, privadas y compartidas, así como la gestión de los permisos de lectura y/o escritura a cada una de ellas a través del identificador *Apple* de cada usuario.
El almacenamiento local se realizará a través del *framework Realm*. Aunque *Apple* tiene su propio *framework CoreData,* todas las alabanzas que hemos leído sobre *Realm* hacen que queramos aprovechar este proyecto como una oportunidad para conocerlo.

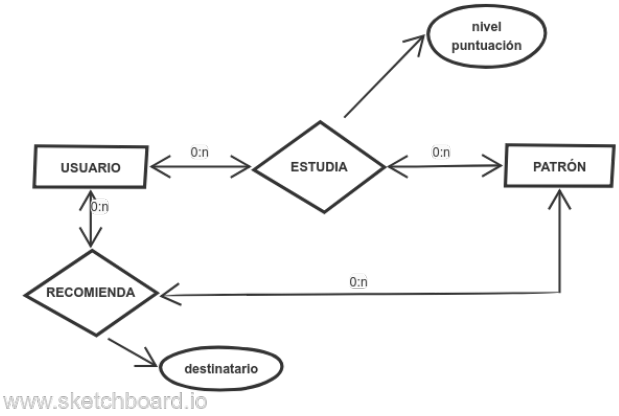
A la hora de realizar la gestión del *ranking*, las puntuaciones y los logros utilizaremos el *GameCenter* de *Apple*. Esto nos simplificará la tarea enormemente. No se realizará una personalización de las pantallas de *GameCenter* para poder cumplir con los plazos establecidos.

Por último, para no perder tiempo en pantallas de poco interés -como pueden ser las que pidan los datos descriptivos de un patrón al usuario- se ha optado por utilizar código compartido en *Git*[15] para la elaboración de formularios de toma de datos. Hemos decidido probar *Eureka*[16] para esta labor.

[15] Puede encontrarse el repositorio de la aplicación desarrollada en github.com: https://github.com/abanet/TFMGuitar

[16] Eureka es un desarrollo open source para la construcción de formularios en entornos iOS.

5.3.1.1 Diagrama UML de diseño de bases de datos



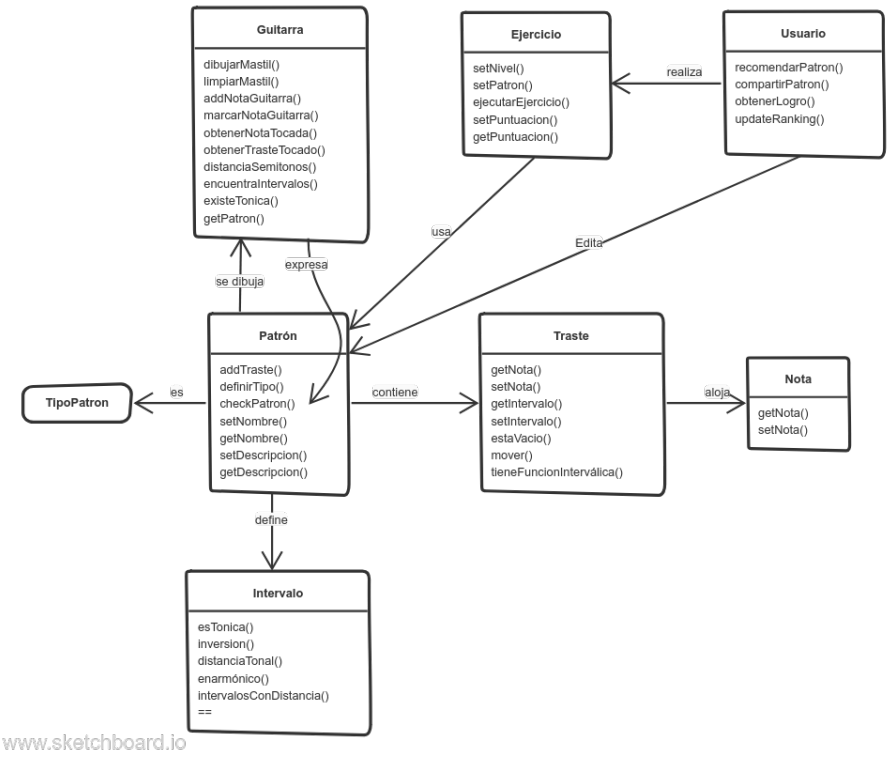
Diseño de la base de datos

La *app* se sustenta sobre una base de datos muy simple: por un lado tenemos los patrones musicales y, por otro lado, los usuarios de la aplicación. Del usuario nos interesan pocos datos, nos basta con un alias para el uso de la *app*. Del patrón, los datos necesarios para su representación, que no son más que los trastes e intervalos que lo conforma y, naturalmente, el nombre, descripción y la categoría a la que pertenece.

En un principio se pensó representar la categoría como una entidad más, pero en aras de la simplicidad del proyecto se decide que sea un campo más dentro de la entidad *Patrón*.

Del estudio de los patrones por parte del usuario obtendremos la puntuación de los diferentes niveles para cada tipo de patrón.

5.3.1.2. Diagrama UML con diseño de entidades y clases



Diseño de entidades y clases de Guitar Patterns

Ya con la vista puesta en el desarrollo de la programación de la *app* identificamos las siguientes estructuras:

Guitarra: Se encargará de la representación e interpretación del mástil de una guitarra. Contiene tantos las funciones de dibujo como de interacción con el usuario y de lógica musical del mástil.
En el mástil de la guitarra nos encontraremos los patrones.

Patrón: es la estructura musical general representada en el mástil y objeto de estudio. En primera instancia se pensó en derivar de la clase Patrón una clase por cada tipo de patrón, pero a posteriori nos parece que esta distinción no aportará grandes ventajas en el desarrollo ya que es una distinción

únicamente de clasificación y no de funcionalidad. Dicha distinción se realizará con la asignación de un tipo de patrón, probablemente un campo enumerado.
Un patrón se compone físicamente de una serie de trastes seleccionados y su lógica viene determinada por los intervalos que representa.

Traste: Es cada una de las posibles posiciones del mástil con las que el usuario puede trabajar. Un traste puede estar vacío o tener una nota.

Intervalo: cada posición dentro del patrón representa un intervalo musical. Los intervalos se rigen por ciertas reglas musicales que hay que contemplar. Aquí tendrá lugar la mayor parte de la lógica musical: distancias tonales, cálculo de enarmónicos[17], inversiones de intervalos, etcétera.

Nota: identifica a cada una de las notas musicales.

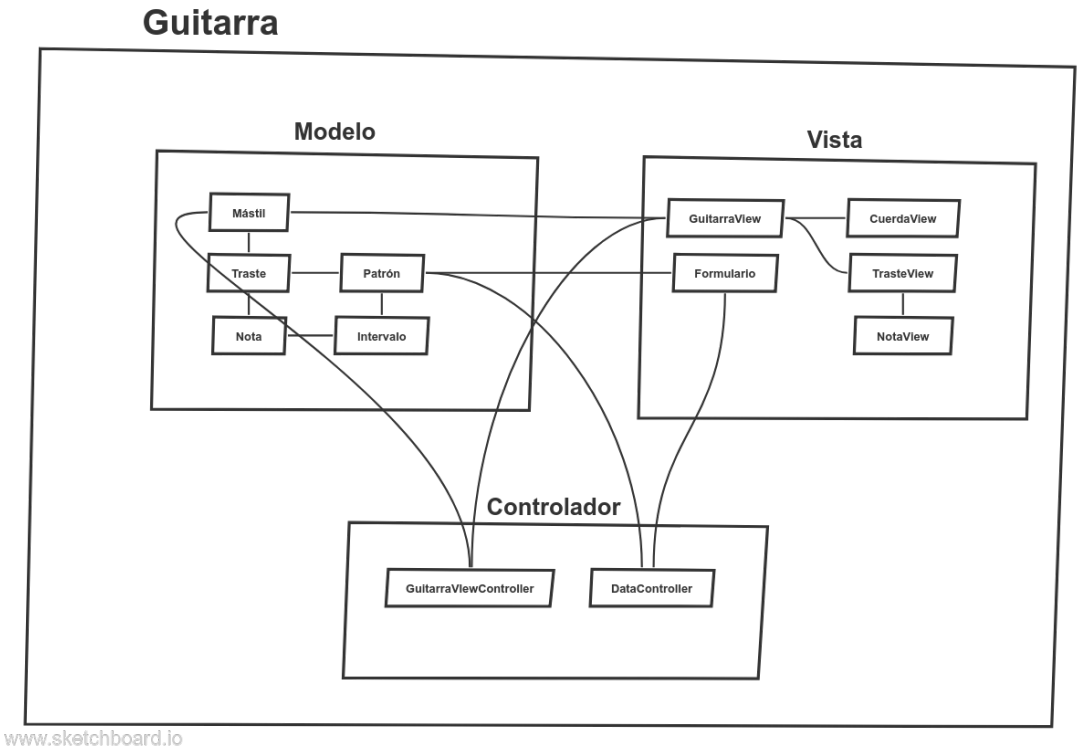
Ya en el uso que el usuario realiza de la *app* aparecen dos entidades nuevas:

Usuario: con los datos del usuario que utiliza la *app* y con todas las posibilidades de lanzar procesos sobre la *app*.

Ejercicio: que se encargará de definir y ejecutar la realización de cada ejercicio concreto que el usuario realiza con un patrón y en un nivel determinado.

[17] Ver Anexo I: Fundamentos de teoría musical

5.3.1.3. Diagrama explicativo de la arquitectura del sistema (paradigma MVC)



Arquitectura MVC

En el desarrollo de la *app* se va a utilizar el paradigma modelo-vista-controlador (MVC). El núcleo central de la *app* pasa por crear un controlador para la guitarra, que se encargará de coordinar la vista de la guitarra con el modelo lógico de la misma al que llamaremos mástil. La vista de una guitarra depende a su vez de las vistas de los diferentes componentes que aparecen en una guitarra: las cuerdas, los trastes y las notas. La parte lógica de las cuerdas se integra en el modelo de los trastes, mientras que la de las notas se apoya en los modelos de notas e intervalos (en la parte gráfica de la guitarra no diferenciamos entre nota en intervalo, ya que la representación es la misma simplemente con cambiar el contenido de la vista entre una nota o un intervalo.

El otro pilar de la aplicación es la recopilación de los datos que formarán la base de datos de la aplicación, por lo que se establecerá un controlador de dichos datos cuya vista vendrá determinada por los formularios que solicitan los metadatos del patrón y que se almacenarán en la base de datos de patrones.

Es importante decir que no se ha incluido aquí el tratamiento de la entidad *Usuarios* ni los datos asociados a *ranking* o a *logros,* ya que la arquitectura en el tratamiento vendrá determinada por la definición que *Apple* hace de usuario a través del *id de Apple* y de la que se hace del *ranking* y los *logros* en *GameCenter*. Si alguna de estas estructuras no fuera suficiente para nuestros intereses, habría que ampliar el gráfico para incluir nuevos *MVC* que incorporen nuestras necesidades a las ya ofrecidas por *Apple*.

DESARROLLO

1. Aspectos generales

A continuación se exponen los aspectos más relevantes que se han tenido en cuenta en la programación de la aplicación.

1.1. Arquitectura MVC

Tal y como ya comentamos, en el desarrollo de la *app* se ha utilizado el paradigma modelo-vista-controlador (*MVC*) intentando separar claramente lo que es el modelo de datos de su representación gráfica y teniendo un intermediario -el controlador- que coordina a los dos anteriores.
En los elementos clave de la aplicación se ha seguido esta organización de forma mayoritaria, aunque no exhaustiva. Por citar algún ejemplo: la rueda dentada que destruye las notas en el juego no representa un ente de importancia tal que requiera separar su implementación siguiendo el paradigma *MVC*. Sin duda, no estaría de más, y crearíamos un elemento reutilizable e independiente; pero, en casos de elementos simples, hemos preferido la inmediatez y simplicidad del código, favoreciendo la optimización del tiempo antes que un desarrollo más robusto y fiel al paradigma *MVC*.

En el caso del acceso a la base de dados hemos optado por utilizar el patrón de diseño *Singleton*. Aunque hemos encontrado en la literatura técnica cierta controversia sobre la pertinencia de dicho patrón, nos ha parecido una forma muy cómodo de permitir el acceso a los datos desde cualquier clase.
En este patrón de diseño es la propia clase la responsable de crear una instancia única que será accesible de forma global desde cualquier otra clase.

1.2. Aplicación adicional

Una de las partes importantes de la aplicación es la existencia de una base de datos común de la aplicación, esto es, una base de datos de patrones que existe en la parte pública dentro de *CloudKit*. Para alimentar esta base de

datos se realizó una aplicación independiente, que es un subconjunto de la presentada como proyecto: sólo tiene un menú para navegar por los patrones públicos y la capacidad de crear, modificar y borrar elementos de la base de datos pública. Esta aplicación no se ofrecerá en el *App Store* ya que su única finalidad es la de alimentar dicha base de datos sobre la que los usuarios de *Guitar Patterns* tendrán únicamente acceso de lectura.

1.3. Documentación del código

Una buena documentación del código es imprescindible cuando tenemos ya una cantidad considerable de clases, -cada una con sus métodos y propiedades- de las que es imposible recordar los detalles.
Hemos encontrado de gran ayuda realizar la documentación del código con la sintaxis *Markdown*[11] que *Apple* integra con *Xcode* y que utilizan en la documentación de sus propias librerías de desarrollo. De esta forma, contamos con ventajas muy importantes, como la de situarnos sobre el nombre de una función y poder ver al momento su objetivo, sus parámetros y forma recomendada de uso, observaciones importantes, etcétera.
Esta sintaxis permite aplicar un formato enriquecido en la descripción de nuestro código. No nos extenderemos más en este punto, baste con mostrar un ejemplo:

```
/**
     Decodifica un array de enteros en trastes.
     - Parameter trastesCodificados: Array con enteros
     ### ATENCIÓN ###
       *Cuidado* la decodificación implica la pérdida de los trastes que pudieran existir en el patrón
     */
```

1.4. Diseño de la interfaz de usuario

Al desarrollar utilizando *Xcode* como entorno de desarrollo tenemos la opción de ayudarnos en el diseño de las pantallas utilizando el *Storyboard* para

aquellas vistas que integran elementos de *UIKit*, o utilizando el editor de escenas en el caso de estar trabajando con *SpriteKit*.
En el proyecto desarrollado se ha optado por no utilizar ninguna de estas ayudas. Hemos preferido elaborar todas las pantallas directamente desde la programación. Aunque la tarea implica más tiempo de desarrollo, nos parece que la aplicación puede ser más fácil de mantener en el tiempo.
Además, la opción elegida nos permite optar entre otros *IDE* que hay en el mercado, lo cual nos parece muy interesante. En concreto comenzamos el proyecto con *AppCode*, entorno del que se leen muy buenas críticas, pero regresamos al *Xcode* para evitar el retardo de tiempo el cambio de *IDE,* ya que queríamos evitar a toda costa desviaciones posibles en la planificación del proyecto.

2. Estructura del código

Vamos a mostrar, de forma breve, la organización del código tanto a nivel de estructura organizativa como funcional.

El código está organizado en las siguientes carpetas y ficheros que citamos a continuación por orden de aparición en la estructura de *XCode*:

sonidos: carpeta con los ficheros de audio que utiliza la aplicación.

fonts: carpeta con las fuentes empleadas en la aplicación.

GuitarPatterns/GameKit: clases que acometen las tareas de comunicación con el *GameCenter* de *Apple*. La clase *GameKitHelper* es la encargada de la autenticación del usuario como del registro de los récords, mientras que *LogrosHelper* se ha creado para la gestión de logros del usuario en *GameCenter*.

GuitarPatterns/Literales: contiene los ficheros con los textos utilizados en la aplicación tanto en español como en inglés.

GuitarPatterns/Modelo: alberga las clases que conforman todas las estructuras de los datos de la aplicación.
Dentro de este directorio encontramos:

Traste, estructura de un traste y funciones para la manipulación de sus datos. Dentro de este mismo fichero se crea la estructura *PosicionTraste* que ayudará en el tratamiento intensivo que en la programación se hace de la posición de los trastes en el mástil.

Nota, estructura de una nota musical.

Intervalo, estructura y funciones de la figura pilar de la armonía con la que se trabaja: el intervalo musical.

Mastil, lógica del mástil como aglutinador de trastes y modelo de las relaciones entre las notas existentes en dichos trastes.

Patron, estructura que representa el patrón musical, la figura armónica en la cúspide de la armonía musical representada en este proyecto.

PatronesDB, contiene las funciones de grabación y recuperación de datos de CloudKit y el tratamiento de la caché de datos.

Nivel, estructura y definición de los niveles de dificultad del juego y sus diferentes parámetros.

Puntuacion, modelo de los *récords* de usuario y de sus partidas acumuladas a nivel local.

GuitarPatterns/Controladores: Aquí se encuentran los dos controladores principales de la aplicación. La pieza central de la aplicación -*GuitarraViewController*- encargada de manejar la lógica y la representación gráfica de un mástil de guitarra, y el controlador de los metadatos de los patrones de la guitarra, *EditDataVC*.

GuitarPatterns/Scenes: escenas que ponen en marcha cada una de las pantallas de la aplicación.

SceneJuego, escena que lanza el juego con el patrón escogido.

SceneEditor, permite la edición de un patrón seleccionado o la creación de uno nuevo.

SceneMenu, crea un control visual que permite la selección de un patrón dentro de un conjunto de patrones y ofrece al usuario las diferentes posibilidades de acción sobre el patrón seleccionado. Dichas acciones son configurables y pueden variar dependiendo del punto de llamada.

SceneMenuPrincipal, se encarga de la creación de la pantalla principal de la app.

GuitarPatterns/Vistas: aquí se aglutinan todas las vistas que definen la representación gráfica de nuestro juego.

StringGuitar, es la representación visual de una cuerda de la guitarra.

GuitarraView, vista que representa el mástil de la guitarra con las estructuras musicales que contenga y recibe la interacción con el usuario.

ShapeNota, es la representación gráfica de una nota en el mástil.

GuitarraStatica, esta clase deriva de *GuitarraView*. Es una representación visual de un mástil y de las estructuras visuales que contiene, pero limitando la interacción con el usuario. Es la clase que se utiliza para la creación de los menús gráficos de patrones en *SceneMenu*.

Boton, botón creado en el entorno de *UIKit*.

BotonMenuPrincipal, botón creado en el entorno de *SpriteKit*.

HUD, elemento de comunicación de información con el usuario durante el desarrollo del juego.

Panel, elabora los paneles informativos utilizados en las transiciones.

GuitarPatterns/Helpers: alberga a diferentes elementos de ayuda, en su mayoría extensiones de clases ya existentes con objeto de facilitar la programación.

Extension+SKShapeNode, extiende la capacidad de dibujo de la clase *SKShapeNode* para ayudar en el dibujo de las cuerdas, trastes y notas de la guitarra.

Alertas, facilita la creación de alertas al usuario.

Extension+String, ayuda en la localización de los textos de la aplicación.

Extension+UIView, incorpora una función para simplificar la eliminación de elementos de *UIKit* en las vistas.

EfectosEspeciales, incorpora toda la capacidad sonora de la *app,* además de la cuenta atrás que inicia el juego y de la explosión que indica el final del mismo.

Extension+CGFloat, funciones de generación de números aleatorios y de conversión de grados a radianes y viceversa.

Extension+SKAction, se amplia la funcionalidad de *SKAction* para simplificar la ejecución de tareas después de determinados intervalos de tiempo.

AppDelegate, delegado de la aplicación en el que nos limitamos a ver si hay patrones que otro usuario haya compartido con nosotros para tratarlo en caso necesario.

GameViewController, es el *ViewController* que arranca el juego. Su principal cometido es el de preguntar al dispositivo sus dimensiones y arrancar la escena principal del juego.

GuitarPatterns, define las constantes globales de la aplicación.

3. Test de desarrollo

La elaboración de test de desarrollo no ha sido tan prolífica como debiera, habiéndose utilizado sólo para aquellas funciones de capital importancia y que, de otro forma, hubiera sido muy difícil probar de forma exhaustiva.

3.1. Test de funcionalidad

Se han escrito algunos test de desarrollo orientados a probar el cálculo armónico que se utiliza en la aplicación.

Dentro del directorio *GuitarPatternsTest* encontramos el fichero *testTraste* que pone a prueba la pieza más básica dentro de la lógica interválica, y el fichero *Mastil* que pone a prueba los cálculos de intervalos entre trastes del mástil.

3.2. Test de la interfaz de usuario

No se han realizado test de la interfaz de usuario desde el *Xcode*. Actualmente este tipo de test funcionan únicamente con elementos de *UIKit* por lo que su uso carecía de interés alguno en un proyecto en el que la mayoría de componentes de la interfaz se han desarrollado con *SpriteKit*.

4. Decisión destacadas en la fase de desarrollo

Durante la fase de desarrollo o implementación del proyecto se tomaron varias decisiones importantes que afectan al producto final.
A continuación destacamos la decisiones más importantes y razonamos su motivación.

4.1. Almacenamiento de la base de datos de patrones musicales

Inicialmente se planteó una solución con la base de datos tanto en local como en remoto. Esto permitiría trabajar incluso sin estar conectados.
Albergar la base de datos en *CloudKit* era un requisito de la *app* que además era imprescindible para conseguir algunas funcionalidades como la compartición de registros o la existencia de una base de datos común a todos los usuarios.
Finalmente no se desarrolló el almacenamiento local de los patrones musicales debido en primer lugar a la economía de tiempo en la fase de implementación. Esta decisión no nos parece grave, ya que la aplicación requiere acceso a la red si queremos poder acceder a nuestras puntuaciones y logros, o si queremos compartir patrones. Por esto hemos optado por apostar por una aplicación que necesita estar conectada pero que ofrece una funcionalidad siempre completa.

4.2. Caché de datos

Debido a lo expuesto en el apartado anterior cada consulta de una categoría de patrones supondría una consulta a la base de datos remota. Esto podría suponer, especialmente en los casos con una conexión deficiente, algunos retardos en el uso de la aplicación. Por este motivo se decidió desarrollar una caché volátil de datos. De esta forma, los datos se leen al conectarse a la aplicación o si llega una petición de compartición de datos.

Esto minimiza el riesgo de que la aplicación deje de funcionar ante la falta de conexión. Si hubo conexión al arrancar la aplicación, podremos utilizarla incluso sin conexión de red salvo, por supuesto, compartir registros con otros usuarios.

La caché existen tanto para los datos públicos como los privados. Esta caché es volátil, es decir, permanece en memoria mientras la *app* está abierta aunque sea en segundo plano.

Hay que mencionar en este punto que no es necesario almacenar la información de récords y logros en local si se está jugando sin red, ya que el propio *framework GamePlay* guarda dichos datos para enviarlos en cuanto se detecte que hay señal.

4.3. Detección de conexión

En una aplicación que, como hemos explicado, no funcionará si no tiene red en el momento en el que arranca, es importante realizar un chequeo y avisar al usuario en el caso de que no haya red.

Esta tarea no tiene una complicación excesiva, sobre todo si utilizamos librerías desarrolladas por terceros, como es el caso de *Alamofire*[18].

Sin embargo esta tarea, -siendo muy importante para la salida de la *app* al *Apple Store*- no es importante a los efectos que ahora nos ocupan, por lo que no se ha tenido en cuenta con vistas a economizar tiempos.

[18] [18] Alamofire es una librería open source escrita en Swift para la gestión de la conexión a la red. https://github.com/Alamofire/Alamofire

4.4. Gestión de los récords

La gestión de *récords* se realiza también a través de *GameCenter*. Sin embargo vimos que *GameCenter* está pensado para llevar un *ranking* de puntuaciones, pero no para hacer un seguimiento estrecho a las puntuaciones de un usuario en particular.

No queríamos renunciar a algo que nos parece fundamental a la hora de aprender: a superarse uno a sí mismo. Es decir, queríamos almacenar la puntuación récord de cada usuario -aunque no entrase en el *ranking* de los mejores- para poder informarle de cuán cercano está de superarse.

Al ser muy poca la información a almacenar se optó por gestionarlos a través de la clase del sistema *UserDefaults* con lo que el tiempo añadido de desarrollo fue despreciable.

4.5. Máquina de estados finita

La aplicación actual sólo ofrece un juego sobre el patrón seleccionado[19] y dicho juego cuenta sólo con tres estados: o se está en pausa, o se está jugando o se ha perdido una partida. Esta mínima variación de estados se solventa sin problemas creando una enumeración con los tres estados y realizando el paso de uno a otro dentro de bucle principal de la escena (función *update*).

A posteriori descubrimos que *GameKit* cuenta con la clase *GKStateMachine* que facilita la gestión de los diferentes estados y la transición entre los mismos.

Pese a que nos pareció un descubrimiento muy interesante, decidimos no modificar el código ya escrito; por un lado, por economía de tiempo y, por otro, por no aportar una ventaja evidente ante una casuística de estados tan pequeña como la mencionada.

[19] Es idea de futuro que sobre un mismo patrón se pueden realizar una variedad de juegos.

4.6. Adaptación multidispositivos

El funcionamiento correcto en todos los tamaños de pantalla posibles no era uno de los objetivos propuestos. Aunque es un objetivo deseable siempre, su implementación va aparejada con más tiempo de desarrollo.

Claramente, si el tiempo no fuera un impedimento, la interfaz de usuario no sería la misma en un *iPhone* que en un *iPad*. Pero tampoco queríamos renunciar a que funcionase en cualquier dispositivo del ecosistema *Apple*.

La solución por la que se optó es por realizar un diseño basado en porcentajes de pantalla optimizando el diseño para el tamaño de la pantalla del *iPhone X* (2.436 x 1.125) pero las proporciones se adaptarán a cualquier dispositivo.

Al probar la app sobre un *iPad* se comprobó que una proporción lineal no era adecuada siempre, por lo que se introdujeron factores de corrección.

Por poner un ejemplo: el espacio reservado para el mástil en un *iPhone* (65% de la altura de la pantalla) se traducía en el *iPad* en un mástil demasiado grande e incómodo de manejar. Esto se corrigió aplicando una corrección al en los *iPad* que reducen este espacio reservado al 50% de la altura del dispositivo.

4.7. UIKit vs SpriteKit

A la hora de desarrollar ciertos elementos de la interfaz de usuario tuvimos dudas sobre que framework utilizar. Nos resulta curioso que un *framework* como *SpriteKit* no ofrezca un elemento equivalente al *UIButton* de *UIKit* que, sin duda, facilitaría la construcción de los menús de las aplicaciones.

Decidimos explorar las dos vías que veíamos factibles: desarrollar nuestro propio botón dentro del marco de *SpriteKit*, y utilizar el elemento *UIButton* de *UIKit* conviviendo en la misma vista (*UIView*) con elementos de la vista de la escena (*SKView*). Así pues, por este afán de investigación, los botones de la aplicación se realizan con esta técnica mixta.

Nuestra conclusión es que hubiera sido mejor ceñirse a la primera opción y desarrollar todos los elementos que aparecen en una *SKView* con objetos del *framework SpriteKit*. Por un lado, una vez creado nuestro propio objeto de

botón como una subclase de *SKNode* -no más de setenta y cinco líneas de código- la tarea de añadir botones es igual de simple que trabajar con *UIButton*; por otro lado, aunque la mezcla de elementos de *UIKit* y de *SpriteKit* conviven bien en cuanto a visualización se refiere, encontramos dos problemas que tuvimos que afrontar:

1) La creación de un *layout* con referencias de unos elementos a otros se complica, ya que la gestión del posicionamiento de elementos tiene diferencias entre los dos *frameworks*. Este problema se resolvió posicionando los elementos en zonas bien diferenciadas de la pantalla y creando dos *layouts* diferentes.
2) Cuando hay una transición de escena, *SceneKit* limpia todos los elementos de su *SKView* pero, curiosamente, deja intactos los elementos de *UIKit* que hemos situado sobre esa misma vista. Para solucionar esto tuvimos que crear nuestras propias funciones para eliminar todos los elementos de las vistas. De forma consciente hemos dejado sin llamar a estas funciones cuando se termina una partida. Se puede observar que el botón *Volver* permanece inalterable pese a la transición de escena que se realiza para mostrar al jugador el panel informativo.

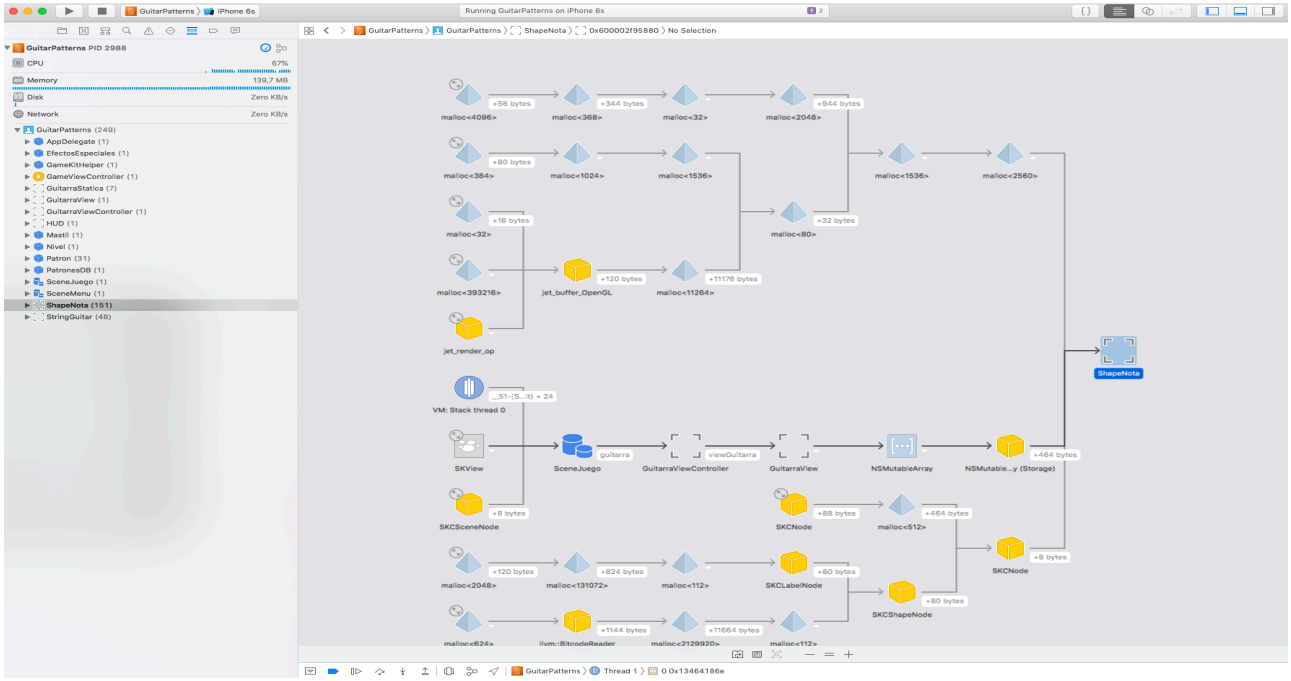
5. Herramientas de análisis del código

Antes de dar el visto bueno al desarrollo hemos utilizado las herramientas de análisis de la aplicación que ofrece *Apple* a través de la herramienta *Instruments*. *Instruments* es un analizador de rendimiento que nos permite encontrar fallos en la programación que afecten al rendimiento de la *app*.

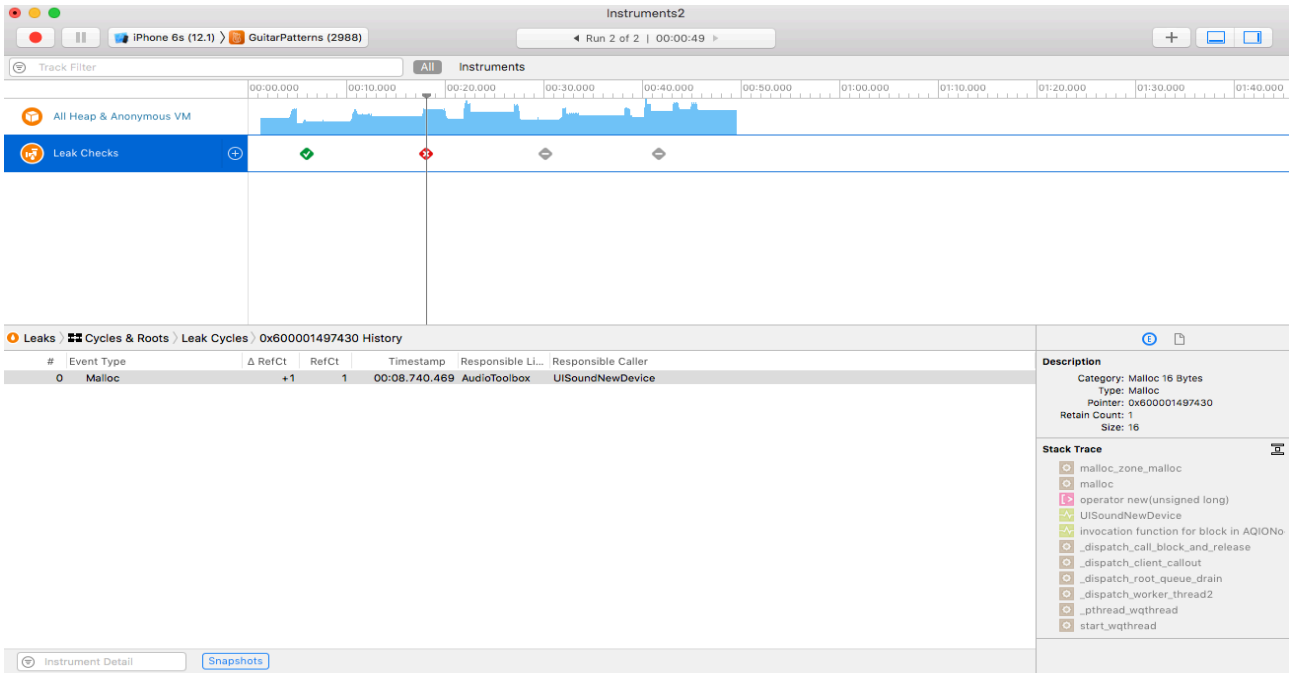
Hemos comprobado que el consumo de *CPU*, de energía, consumo y red es completamente normal para el tipo de aplicación que hemos realizado.
En los gráficos de memoria, sin embargo, hemos encontrado un ligero incremento en el tiempo que nos motivó a buscar *leaks* de memoria.
Se encontraron dos *leaks* de memoria asociados a la librería de *Apple AudioToolbox*. Por lo que hemos indagado en foros especializados, parece que

el origen se debe a un bug en dicha librería, por lo que su corrección se escapa a nuestro ámbito de responsabilidad; sólo nos queda confiar en que en futuras versiones *Apple* corrija este error.

Ofrecemos dos de los gráficos más interesantes obtenidos en el proceso de análisis y depuración de código:



Búsqueda de fugas de memoria en nuestras clases



Proceso de depuración de leaks: encontrados leaks en Audio Toolbox

FASE DE PRUEBAS

En el apartado anterior hablamos de las pruebas unitarias que aplicamos para verificar el funcionamiento de algunos de los métodos centrales a la lógica de la aplicación. Es ahora el turno de las pruebas que se han realizado para garantizar un funcionamiento correcto y deseado.

A diferencia de las pruebas anteriores que eran automatizadas, el resto de pruebas se han realizado de forma manual con la ayuda de un conjunto de usuarios. El equipo de *beta testers* ha estado formado por, -además del desarrollador que a su vez es futuro usuario de la *app*- dos profesores de guitarra, tres alumnos de guitarra y dos usuarios ajenos al mundo de la música en general

1. Pruebas funcionales

Las pruebas funcionales han sido realizadas por el jefe de proyecto para garantizar de que los diferentes módulos del proyecto funcionan, es decir, hacen lo que a priori se les pide.
Para esto se probó lo siguiente:

Test validación en GameCenter: se introduce nuestro *id de Apple* comprobando que entramos correctamente. También se comprueba que aunque decidamos cancelar el acceso se puede seguir disfrutando de la *app* igualmente.

Test de edición de patrones: el módulo de edición de patrones ha sido probado de forma exhaustiva, ya que toda la base de datos pública de patrones se ha generado con el editor construido. Se ha detectado que ,con sólo tocar la pantalla de edición en cualquier punto, el botón de grabar se activa indicando que hay cambios para grabar. Esta activación sólo debería de ocurrir si se ha tocado sobre una nota.

Test de escritura/modificación/borrado de patrón: en la elaboración de la base de datos general se han probado intensamente las operaciones sobre los patrones en la base de datos pública, dentro de la aplicación de edición de patrones. En la *app* final las operaciones se realizan sobre la base de datos privada pero el código responsable es el mismo. No obstante, también se han realizado pruebas al respecto.

Test de adquisición de logros: se configuran unos logros mínimos para poder comprobar que estos se adquieren. A la hora de publicar la *app* será necesario ajustar dichos parámetros.

Test de compartir patrón: se ha ejecutado la *app* en dos terminales. Desde uno de ellos hemos probado a compartir un registro, tanto por email como por *Whatsapp*. En todos los casos el funcionamiento ha sido positivo. El único inconveniente encontrado es que parece que el icono elegido para la compartición de registros parece no llegar al destinatario, pese a que se muestra correctamente en las pantallas de envío. No encontramos errores en la programación y, por lo que hemos podido investigar, pensamos que es algo que se arreglará cuando la app esté publicada.

Test de añadir a favoritos: se prueba sin problemas a pasar registros del catálogo público a la lista de favoritos.

Test de conectividad: se ha simulado la pérdida de red en diferentes momentos de la app. Si la pérdida ocurre después del arranque inicial en el que se cargan todas las figuras, la incidencia es mínima. Si embargo, algunas funciones dejarán de estar disponibles, y lo correcto es notificárselo al usuario. Apuntamos esto en la lista de mejoras futuras.

Test de juego: se busca reproducir toda la casuística posible: pasos de nivel, partidas perdidas, récords batidos, etcétera. Para facilitar esta tarea hubo que ajustar las configuraciones de los niveles.

2. Pruebas integrales

Se trata aquí de probar el comportamiento del usuario con la aplicación al completo. La única guía que se le ha dado a los usuarios que realizaron las pruebas es la descripción que aparecerá en el *App Store*. Se trata de probar que el flujo y organización de la *app* es el adecuado para conseguir los fines propuestos.

De estas pruebas sacamos las siguientes conclusiones:

1) La navegación principal de la app es intuitiva y no presenta ninguna dificultad. El único caso destacable es el de un usuario que escogió ir como primera opción a *Mis patrones* y se encontró con la pantalla sin ningún patrón, ya que para que aparezca alguno es necesario que el usuario los haya creado o asignado previamente. Esto supuso un momento de incertidumbre, hasta que el usuario volvió hacia atrás y al entrar en otras opciones vio el botón de añadir a *Mis patrones*. Para evitar esto hay que añadir un texto explicativo para que aparezca en pantalla cuando no existen patrones asociados.

2) Todos los usuarios, incluidos los que carecían de ningún conocimiento de guitarra, intuyeron en qué consistía el juego a base de ensayo y error y tardando desde diez segundos el más rápido hasta tres partidas el más lento. A uno de los usuarios sin conocimiento musical sí hubo que explicarle que para cumplir el objetivo había que pulsar todos los círculos que hay en pantalla con el mismo número del círculo que sale en la parte superior. Es decir, que para hacer desaparecer un intervalo es necesario marcar todas las posiciones del intervalo en el mástil. Viendo que este punto se le puede atragantar a alguno de los usuarios, se podría mostrar un panel con instrucciones que aparezca sólo en las primeras partidas (después consideramos que sería innecesario)

3) Se encontró un error que se producía cuando un usuario creaba un patrón e inmediatamente intentaba jugar con él. El error era debido a que la

interválica del patrón se calculaba únicamente cuando un patrón era descargado de la base de datos. Cuando se crean patrones nuevos, se almacenan temporalmente en una caché, por lo que no tenían interválica. Se ha modificado el código (versión 1.3) para que la interválica se calcule no sólo al leer de la base de datos, sino también cuando se crea un patrón nuevo.

4) Algunos usuarios se han mostrado dubitativos ante el proceso de grabación de un patrón. La causa se debe a que, cuando entran a editar el nombre, hay un botón de grabar, y piensan que van a grabar un patrón vacío si no se ha metido anteriormente la información de dicho patrón. Uno de los usuarios sugiere la posibilidad de que se pueda modificar toda la información en la misma pantalla. Un cambio a tener en cuenta sobre todo en las pantallas más grandes.

5) Los usuarios más avanzados nos plantean alguna discrepancia de la aplicación con la realidad. En nuestra aplicación, siempre que editamos un patrón, al pulsar una nota el intervalo que aparece es determinista; es decir, siempre que pulsamos sobre una distancia de cuatro tonos y medio desde la tónica se marca un intervalo de sexta. Esto, sin embargo, no es cierto ya que en ciertas escalas podría ser un intervalo de *7bb* (séptima con doble bemol). Este es el problema de los enarmónicos[20]: un mismo intervalo en la guitarra puede tener nombres diferentes según en que escala estemos. Este problema sólo se presentará en posiciones complejas de nivel alto, pero, si queremos que la aplicación se dirija a todos los niveles, hay que afrontar esta situación. Se plantean dos opciones que habrá que evaluar en un futuro: la más sencilla -dejar la responsabilidad al usuario- es que pulsando sobre la misma posición cambie de enarmónico si es que hay alguno disponible; otra más completa pero también más compleja sería que el nombre se pueda deducir según la

[20] Ver Anexo I: Fundamentos de teoría musical

escala en la que estemos -decidir el enarmónico adecuado de forma automática.

3. Usabilidad

Con respecto a la usabilidad todos los usuarios entendieron correctamente la navegación por el sistema, y entendieron sin gran esfuerzo las diferentes opciones de la aplicación. Hay que mejorar los dos casos ya mencionados: el proceso de grabación de un registro y el de inicio del juego.

Sí notamos que la calidad de la experiencia varía de unos dispositivos a otros. A través de simuladores se probó la visualización en diferentes tamaños de dispositivos para garantizar que toda la información se mostraba correctamente. Sin embargo, el diseño inicial y todo el desarrollo se hizo orientado al *iPhone*. Es en estos tamaños de pantalla donde el usuario encuentra el uso de la aplicación más agradable.

Sin duda el diseño de pantalla sobre el *iPad* tiene que ser otro muy diferente que habrá que trabajar en futuras versiones. Gracias a su gran pantalla se podrían reducir sobremanera la navegación de menús. Por ejemplo, se podría hacer un selector en el que se muestren acordes, arpegios y escalas en tres filas o columnas simultáneas. Esto conlleva la gran ventaja de permitir ver la relación entre una escala con los acordes y arpegios que se derivan de la misma.

Otro cambio de diseño sería sin duda el de el *editor de patrones;* en este caso , la pantalla tiene espacio más que suficiente para incluir toda la información sobre el patrón.

4. Pruebas de localización

Se cambia la configuración de idioma del dispositivo al inglés para comprobar que la localización está funcionando correctamente. Se encontró un error en la traducción del botón *Reset* que fue corregido de inmediato.

5. Pruebas de nivel

Para cada patrón se definen seis niveles de dificultad. Se trata de comenzar sin presión para que el usuario tenga tiempo de localizar los intervalos y pueda centrarse en la relación de este con el resto de notas del patrón, especialmente con la tónica. A medida que pasa de nivel la dificultad será mayor. Fue necesario encontrar un equilibrio, de forma que los niveles no resultaran ni demasiado sencillos ni demasiado complicados.

Todos nuestros evaluadores jugaron en repetidas ocasiones y fuimos ajustando los parámetros de forma oportuna a nuestros propósitos.

Con esta prueba nos dimos cuenta de que lo que para un amateur es difícil para un experto puede ser fácil. Surge así la idea de futuro de poder configurar estos parámetros según el tipo de usuario que está jugando.

6. Pruebas de adquisición de conocimiento

Sin duda la prueba más importante. Hemos podido constatar la adquisición de estructuras, al menos a corto plazo. Incluso usuarios que no sabían lo que estaban aprendiendo han llegado al nivel final (en el que ya no hay ayudas visuales) y marcaban las posiciones de los intervalos correctamente. Cierto que esto ha ocurrido con estructuras sencillas, pero aún así creo que es una prueba de que puede establecerse la asociación entre posición relativa a la tónica e intervalo correspondiente.

Usuarios con conocimiento de guitarra, entre los que me incluyo, también mostraron su ilusión al aprender estructuras nuevas en poco tiempo y, sobre todo, al desarrollar enormemente su conocimiento de la interválica en el mástil de forma casi automática.

En este sentido cabe decir que, por el momento, parece que las estructuras sencillas se pueden aprender bastante bien, al igual que las posiciones de los intervalos con respecto a la tónica, independientemente de en qué cuerda y traste puedan estar. En estructuras más complejas como son las escalas, a corto plazo se percibe que la adquisición interválica tiene lugar pese a no aprenderse la escala como un todo. En las pruebas que seguiremos haciendo a medio y largo plazo, habrá que descubrir si la comprensión de la escala

como un todo es cuestión de dedicar más tiempo a las estructuras más complejas o, por el contrario, si el foco puesto por el ejercicio en la particularidad del intervalo como unidad de conocimiento impide ver la escala como una unidad de conocimiento en si misma. Si este fuera el caso habría que diseñar ejercicios orientados en este sentido para complementar la aplicación.

PUESTA EN PRODUCCIÓN

El destino final de la aplicación es, como no podía ser de otra manera, ser publicada en la *App Store* a disposición de cualquier estudiante de guitarra.
La puesta en producción ha resultado ser un proceso más complicado de lo que habíamos previsto inicialmente.

Exponemos a continuación la casuística con la que nos encontramos.

1. Entornos de bases de datos en *CloudKit*

CloudKit ofrece dos entornos para las bases de datos: uno de desarrollo y otro de producción. De esta forma, sin cambiar nada en nuestra aplicación, cuando estamos desarrollando se *ataca* automáticamente a la base de datos de desarrollo mientras que cuando se publica nuestra *app* todas las operaciones se realizan sobre el entorno de producción.
La ventaja de esta solución es evidente: podemos hacer pruebas en desarrollo sin temor a cargarnos datos que ya están en producción.
El problema con el que nos encontramos es que cuando quisimos intentar poner nuestra base de datos en producción, descubrimos que la migración que se hace es de estructura, pero no de datos. No hay forma de mantener ninguno de los datos incorporados en desarrollo en nuestra base de datos de producción.
Encontramos lógico que esto sea así en los contenedores privados o compartidos, pero en nuestro caso particular, este hecho nos supuso un gran trastorno: la base de datos pública tiene que estar poblada en el momento en el que salga al *App Store* y, sin embargo, todos los datos de patrones que habíamos incorporado desaparecían al cambiar a la base de datos en producción.

Otro problema con el que nos encontramos es que en las primeras versiones que subimos a producción nos encontramos con casuísticas nuevas derivadas del uso de la *app* por un usuario que no tenía todos los derechos sobre la

base de datos. Por este motivo, aparecieron nuevos problemas que nos costó afrontar hasta que descubrimos dos nuevas opciones de depuración muy interesantes: la primera es el análisis de los *crash* que generan los usuarios que utilizan la aplicación -una fuente inestimable para la optimización y mejora de nuestra *app*-; y la segunda es la posibilidad de ejecutar nuestra aplicación desde el entorno de desarrollo, pero trabajando en un entorno de producción. Este segundo punto nos costó bastante descubrirlo pero finalmente fue tan sencillo como añadir un nuevo *entitlement* con la variable *com.apple.developer.icloud-container-environment* establecida a *Production*.

2. Proceso de migración de la base de datos

Encontramos dos posibles soluciones:

1) Crear una *app* para la migración de datos: se crea una aplicación que lee y almacena en local todos los datos de la base de datos pública, y que también almacena todos esos datos a la base de datos pública en la nube. Se ejecuta en entorno de desarrollo para bajar a local todos los datos que se incorporaron en dicho entorno. Entonces nos descargamos dicha aplicación a través de *TestFlight*[21] -cambio al entorno de producción- y subimos todos los datos a la pública.
2) Descargarnos la aplicación de creación de la base de datos pública a través de *TestFlight* y repetir la creación manual de patrones uno por uno, tal y como se hizo en el entorno de desarrollo.

En nuestro caso decidimos optar por la segunda opción, ya que el número de patrones no era excesivamente elevado y de esta manera se ahorraban unas horas de programación.

[21] Herramienta de Apple para distribuir las aplicaciones entre los beta testers. Una app descargada desde Testflight se ejecuta automáticamente en modo de producción.

FUTURO DE GUITAR PATTERNS

La aplicación que presentamos como *TFM* se plantea como la versión inicial de una aplicación que esperamos siga creciendo en un futuro.
A lo largo del desarrollo surgen nuevas ideas que, desgraciadamente, no se pueden abordar en los plazos establecidos. También hay procesos de mejora naturales que surgen como resultado de las pruebas finales.
En este apartado queremos dejar constancia de aquellas mejoras de las que somos conscientes y que no hemos podido acometer, y plantear ideas de desarrollo que nos parecen interesantes de cara al futuro de la aplicación.
Finalmente estudiaremos las posibilidades de monetización de la aplicación.

1. Mejoras de la App

Después de la fase de pruebas hemos encontrado algunas mejoras a la app que es necesario afrontar tan pronto como sea posible:

1) Tal y como se descubrió en la fase de pruebas hemos pecado de una programación *'optimista'* en cuando a la comunicación con el usuario. Con optimista nos referimos a que siempre que la operación realizada funciona correctamente informamos al usuario, pero hemos dejado de informar en algunos casos en los que la operación no se ha podido realizar. La solución es sencilla y debe de realizarse lo antes posible. Por ejemplo, en el caso en que el usuario no tenga conexión a la red hay que avisarle si esta fuera necesaria. Hay librerías, como puede ser *Alamofire*[22], que simplifican esta labor.
2) Ofrecer un mínimo de instrucciones que permitan afrontar con éxito los primeros encuentros con la *app*.
3) Incorporar estadísticas de uso para analiza la conveniencia o no de una base de datos pública.
4) Crear filtros por familias de acordes/arpegios/escalas. Salvo que lleguemos a la conclusión contraria el futuro de la *app* pasa por

[22] Alamofire es una librería open source escrita en Swift para la gestión de la conexión a la red. https://github.com/Alamofire/Alamofire

incrementar la base de datos pública. Al hacer eso la navegación actual puede resultar incómoda y será necesario aplicar filtros por familia. La familia es la forma natural de agrupar posiciones y no contienen demasiados elementos. Se añadiría un selector en la parte superior del mástil en la que al pulsar sobre el nombre de una familia se estableciera un filtro para mostrar los patrones pertenecientes a dicha familia. Por ejemplo, dentro de las escalas tendríamos: pentatónica, mayor, menor, dórico, blues mayor, blues menor, … Generalmente, cuando se está estudiando una familia se querrá acceder a todos los patrones de la misma, por lo que consideramos perfecto este método de selección.

5) El botón de grabar en la pantalla de edición se pone en rojo (indicando que la información del patrón ha sido modificada) incluso si tocamos sin modificar la información. Para que exista una coherencia, esto sólo debe de ocurrir si hay un cambio real en la información del patrón que se está editando.
6) En la navegación por los patrones de la base de datos pública sería necesario añadir en cada patrón una señal gráfica que indique que ese patrón ha sido copiado a nuestros patrones. De esta forma se evita que el usuario pueda añadir varias veces el mismo patrón a su base de datos privada.
7) De manera similar a lo comentado en el punto anterior, convendría informar al usuario de si un patrón ha sido superado.
8) Es conveniente contemplar la posibilidad de enarmónicos[23] en la aplicación. La solución más sencilla es la de permitir cambiar de un enarmónico a otro al tocar sobre una posición. Por ejemplo, si tocamos sobre un traste vacío que es una cuarta aumentada aparecería 4+. Si volvemos a tocar sobre él aparecería una quinta disminuida (5b), y ya tocando otra vez sobre la misma se volvería a la situación de traste vacío.

[23] Ver Anexo I: Fundamentos de teoría musical.

2. Ideas de futuro

Durante la realización del proyecto surgen ideas interesantes que pueden enriquecer enormemente la aplicación. Puesto que no estuvieron en la planificación inicial no ha habido tiempo de afrontarlas, pero serán interesante hacerlo en un futuro próximo.

Además de las ideas que ya apuntamos en la fase de diseño[24] exponemos estas complementarias:

1) Sugerir caminos en el estudio de patrones: en el estudio de los patrones hay caminos más o menos establecidos en la enseñanza de la guitarra, sobre todo en los estudiantes más recientes; en este caso es práctica común empezar por la escala pentatónica y posiciones de acordes y arpegios básicas. Basándonos en esto se podría solicitar el nivel del usuario y aconsejarle un camino a seguir para que no se pierda en una base de datos de patrones que puedan no ser de su interés.
2) Poder trabajar los patrones con notas, además de con intervalos: el primer paso y el más sencillo es aprender un patrón como un esquema abstracto formado por intervalos; sin embargo, según avanzan nuestros conocimientos musicales deberemos de ser capaces de concretar esa abstracción en las notas correspondientes a la escala en la que nos estemos moviendo. La idea es que el usuario pueda elegir que nota es la tónica y a partir de ahí sustituir los intervalos por notas. Esto es, en apariencia, sencillo de implementar pero será necesario profundizar en nuestro conocimiento de armonía para poder llevar a cabo la tarea con éxito ya que no existe una relación directa entre intervalo y nota (según la escala en la que nos encontremos una misma nota puede representar diferentes intervalos).
3) Aplicar *Machine Learning* para el reconocimiento de los patrones. De esta forma los patrones podrían nombrarse de forma automática. Además, permitiría la realización de un ejercicio en el que se pregunta un patrón al usuario y este tiene que escribirlo en el mástil en un tiempo determinado.

[24] Véase apartado 2.3 *Funcionalidad ampliada* dentro de la sección dedicada al *Diseño centrado en el usuario*.

Este tipo de ejercicio puede ayudar a la adquisición del conocimiento de patrones como un todo.

4) Posibilidad de impresión a fichero en formato *pdf*: la *app* está pensada para estudio de patrones con el móvil, sin embargo, una vez el estudiante tiene la guitarra en sus manos existen una multitud de ejercicios que tiene que practicar visualizando dicho patrón. Existiría la posibilidad de tenerlo abierto en la *app* pero algunos *beta tester* comentaron la utilidad de poder tenerlo en papel. Así, se nos ocurre que puede ser interesante ofrecer la posibilidad de imprimir un patrón a formato *pdf*. Esto abre la posibilidad de imprimirlo si se quiere en papel, o de enviárselo a otro usuario (por ejemplo un profesor a sus estudiantes). A partir de la versión 11 de *iOS Apple* incorpora el *framework PDFKit*, con lo cual la programación de este objetivo se simplifica en gran medida.
5) Incorporación de nuevos ejercicios complementarios: el objetivo de la aplicación es la asimilación de patrones musicales. Para ello creemos que es mejor realizar diferentes ejercicios sobre un mismo patrón. De esta forma, se escapa de la monotonía y será más fácil que un estudiante persevere con un patrón. Sobre los elementos clave programados es sencillo realizar variedad de ejercicios. Por ejemplo sería interesante realizar ejercicios en los que las notas no salgan al azar sino, por el contrario, en un orden lógico musical como puede ser el recorrer una escala con saltos interválicos constantes, o quizás en orden secuencial para potenciar la adquisición del patrón como un todo; también es muy interesante realizar un ejercicio similar al que se presenta pero en el que el estímulo inicial no sea una visualización de un intervalo, sino su expresión sonora. De esta forma se desarrollará también el oído del estudiante, un factor muy a tener en cuenta en el estudio de cualquier instrumento.
6) Realizar una parametrización de los niveles ajustada a la capacidad del usuario que utiliza la aplicación. En la *app* presentada hay una variación en los diferentes niveles que es única e igual para todos los jugadores. Vemos más conveniente que dichos parámetros sean capaz de adaptarse

a la habilidad del usuario y que busque sus límites de una forma personalizada.

7) Afrontar la generalización del instrumento: la *app* presentada permite trabajar sólo con un mástil de una guitarra de seis cuerdas. Sin embargo, el código está preparado para que se pueda elegir otro instrumento, como puede ser una guitarra de siete cuerdas, un bajo de cuatro o de cinco cuerdas, un ukelele, etcétera. Habrá que considerar la conveniencia de una *app* que permita utilizar diferentes instrumentos o, por el contrario, afrontar diferentes aplicaciones que únicamente se diferencien en el instrumento utilizado (generalmente un estudiante de bajo, por citar un ejemplo, no estará interesado en los patrones en una guitarra o ukelele).

3. Monetización

Una vez realizadas las mejoras identificadas llegaría el momento de pensar en la monetización de la aplicación. En este sentido destacamos tres posibles opciones:

1) Incorporar publicidad en la aplicación. Esta opción no nos gusta, ya que los resultados suelen ser mínimos salvo que nos encontremos con un uso masivo de la aplicación. En general no somos partidarios de aplicaciones con publicidad pero, -de elegir este formato- preferimos hacerlo a través de anuncios intersticiales de forma que, una vez cerrado por el usuario, la visualización de la *app* quede limpia y sin elementos distractores. De abordar esta solución tendríamos somos partidarios de hacerlo con el *SDK* de *Google Ads*[25] ya que es el más extendido y el que mejor resultados puede generar.

2) Aplicación de pago. Se trata aquí de poner un precio a la aplicación. Aunque el resultado hipotético puede ser interesante, entramos en competencia directa con otras aplicaciones de pago. Por lo general, estas están realizadas por compañías -ya sean pequeñas o grandes- y suelen

[25] Plataforma de anuncios de Google.

contar con diseños más trabajados y desarrollos más elaborados. Esta solución es siempre tentadora, ya que no es necesario tocar el desarrollo en absoluto, basta con cambiar el coste de la aplicación en el *App Store*.

3) Aplicación gratuita con compras internas. Desde *iTunes Connect*[26] podemos definir comprar internas en la aplicación. Podríamos definir paquetes de estructuras para incorporar a la aplicación. De esta forma la *app* es gratuita con una base de datos pública reducida, y dentro de la *app* se pueden ofrecer la descarga de nuevos patrones agrupados por familias. Por ejemplo: patrones del modo dórico, frigio , lidio, etc. Lo bueno de este modelo es que la oferta puede ser muy amplia, ya que los mismos patrones podrían estar en diferentes paquetes de descarga. Así, se podría ofrecer un paquete por tipo de música: rock, blues, jazz, etcétera; por escalas, por culturas musicales (los patrones de la música oriental son diferentes de la occidental, o de la música árabe); o incluso por época, ya que los patrones utilizados en la composición de la música han ido variando a lo largo de la historia de la música. Este modelo es el que más nos gusta aunque es, sin duda, el que más tiempo de desarrollo necesita. Implica tanto la labor de creación de los artículos susceptibles de ser comprados como la programación que define qué permiten dichos artículos. Además, habrá que encargarse de la gestión de las compras y de posibles restauraciones de artículos ya comprados.

[26] Herramienta online para la gestión de aplicaciones en el App Store.

CONCLUSIONES

En la presentación de este proyecto distinguíamos entre un objetivo académico en sí que era profundizar nuestro conocimiento en el desarrollo en el entorno *iOS*, y un objetivo más tangible en forma de una aplicación móvil para ayudar al estudiante de guitarra a memorizar patrones musicales.
Es necesario, por tanto, establecer conclusiones en cada uno de dichos objetivos.

Respecto al primero de ellos, hemos estudiado los framework SpriteKit, GameKit y CloudKit, que son los ofrecidos por Apple para facilitar el desarrollo de videojuegos, así como su gamificación. Nuestra conclusión en este sentido es muy clara: Apple nos sigue sorprendiendo con el gusto que pone en todos los productos que hace, y no es una excepción en aquellos que se refieren al desarrollo de software; y sin embargo, siendo entusiastas como somos con este entorno, creemos que sería más adecuado realizar un producto como el que planteamos con algún software específico de desarrollo para videojuegos como pueden ser Unreal o Unity.
Esta conclusión la basamos en la comparación del esfuerzo necesario para realizar ciertas tareas en el proyecto actual con aquellas similares que se realizaron en la asignatura de I*ntroducción a videojuegos en dispositivos móviles*. Además de poder obtener un producto multiplataforma con prácticamente el mismo esfuerzo, las tareas orientadas a la animación, movimiento e interfaz de usuario se simplifican enormemente. Existe una gran cantidad de material gráfico que permitirá enriquecer enormemente el resultado final del proyecto disponibles tanto para *Unity* como para *Unreal*.
En cuanto a la facilidad de gamificación y entorno *iCloud* que ofrece *Apple*, son fácilmente sustituibles por otros productos como pueden ser *Parse* o *GameSparks*[27], este último especializado en sociabilizar y gamificar aplicaciones.

[27] GamSparks ha sido recientemente adquirido por Amazon con lo que podemos esperar que sea mantenido y potenciado en el tiempo.

Estamos muy satisfechos con el resultado de la aplicación desarrollada y el conocimiento adquirido por el camino. Hemos constatado que el aprendizaje tiene lugar, y eso pese a ser conscientes de que el juego realizado no es el más divertido del *App Store*. La evolución de la tecnología móvil y del desarrollo de videojuegos tiene que suponer una evolución en el aprendizaje de cualquier materia, y si esto no es ya una realidad, es por motivos de índole económica. Sin duda es un aspecto en el que las universidades públicas deberían de hacer énfasis y ser pioneras en este frente que, a mi juicio, está muy descuidado en la actualidad.

En cuanto al proceso de trabajo empleado, la metodología ágil se ha mostrado muy útil y la planificación ha sido ajustada y no ha sufrido grandes variaciones. Sí hemos tenido que incorporar más horas de las previstas al descubrir que el entorno *iCloud* de *Apple* no permitía una migración directa de la base de datos pública de desarrollo a producción. Este hecho, y algún problema que nos ha costado resolver, hizo que tuviéramos que aumentar las horas de desarrollo planificadas. También tuvimos que renunciar a introducir algunas mejoras interesantes que han surgido durante la realización del proyecto, pero que finalmente hemos tenido que limitarnos a mencionar en el apartado en el que hablamos del futuro de la aplicación.
La metodología ágil ha favorecido la aparición de nuevas ideas continuamente y sólo se han visto frenadas por la limitación del tiempo de trabajo.

Con la aplicación final también estamos satisfechos en cuanto a que realmente hace lo que nos habíamos propuesto. Sin embargo, como crítica constructiva, sí tenemos que decir que dista mucho de la idea que tenemos en mente. Por un lado, es necesario contar con diseñadores gráficos que conviertan un entorno gráfico parco -el diseño no es nuestro fuerte- en algo realmente atractivo y sugerente; por otro lado, habría que contar con expertos en armonía musical para poder afrontar patrones musicales más complejos y dotar a la aplicación de un orden en la adquisición del conocimiento. Ahora mismo, dicho orden se deja a la elección del usuario, con

lo cual es necesario que el estudiante esté orientado en este sentido por algún profesor en la materia.
También creemos que será necesario enriquecer las actividades para trabajar un patrón musical. Tener siempre una misma tarea que simplemente cambia de nivel puede llegar a cansar a los estudiantes menos motivados. Con las clases desarrolladas es sencillo definir nuevos juegos que permitan variar la dinámica del aprendizaje.

De forma global consideramos la realización de este Trabajo Fin de Máster muy satisfactoria, ya que nos ha permitido incrementar en gran medida el nivel de nuestra programación en entornos iOS y, además, hemos obtenido una herramienta que nos permite avanzar en nuestros conocimientos de guitarra cuando viajamos cada día en el autobús camino del trabajo. Han sido meses en los que hemos tenido una excusa para dedicar tiempo a dos de nuestras pasiones: la programación con Swift y el estudio de la armonía en la guitarra; y si, como aseguraba Stendhal[28], el *alma es el conjunto de las pasiones*, este proyecto lleva parte de la mía.

[28] Henry Beyle (1783-1842), escritor francés más conocido por su seudónimo Stendhal.

GLOSARIO

Acorde	En teoría musical es un conjunto de tres o más notas diferentes que suenan simultáneamente.
App	Aplicación software que se instala en dispositivos móviles o tablets. En el texto nos referimos indistintamente a una *App* como a una *Aplicación*.
Apple	Nos referimos de esta forma abreviada a la compañía informática americana *Apple Inc.*
App Store	Tienda online a través de la cual *Apple* distribuye sus aplicaciones móviles.
Armonía	La armonía (musical) estudia cómo combinar las notas musicales emitidas simultáneamente y la sucesión de las mismas.
Arpegio	Es la sucesión de las notas que componen un acorde.
Backend	Capa que comprende todos los componentes de la aplicación que están alojados en la nube.
Beta tester	Usuario que prueba la aplicación en la fase de pruebas con objeto de detectar fallos en la misma.
Caché	Área de almacenamiento dedicada a la recuperación a gran velocidad de los datos utilizados con mayor frecuencia.
Cloud	Utilizado como abreviatura de *cloud computing*. Se refiere a aquellos servicios que se alojan en servidores a los que se accede a través de la red.
Core Data	Framework de Apple para la persistencia de datos. Incluye tanto el modelado de los datos como su almacenamiento y recuperación.
Crash	Condición en la que la aplicación deja de funcionar en su totalidad que no ha sido contemplada en la programación.
Cuerda	Nos referimos por cuerda a los elementos vibratorios de la guitarra que originan el sonido producido por el instrumento.
Enarmónico	Son aquellas notas que teniendo diferente representación gráfica tienen el mismo sonido o tono.
Entitlement	Es la forma en la que asignamos permisos a una aplicación para utilizar un servicio o tecnología dentro del ecosistema de *Apple*.
Escala	Escala musical entendida como una sucesión ordenada de notas musicales.
Escena	Hablamos de escena para referirnos a una pantalla de la aplicación. Técnicamente es el nodo raíz en la representación gráfica del contenido en *SpriteKit*.
Eureka	Librería Open Source que facilita la creación de formularios en *Swift*.
Framework	Estructura conceptual formada por diferentes módulos de software que facilita el desarrollo en una temática determinada.
Game Center	Red social online de *Apple* orientada a los juegos y su gamificación.

GameKit	Framework proporcionado por *Apple* para la construcción de juegos sociales a través de *Game Center*.
Gamificación	Técnica de aprendizaje que traslada la mecánica de los juegos al ámbito de la educación.
Git	Software de control de versiones.
HUD	Acrónimo del término inglés Heads-Up Display que hace referencia a la presentación de la información al usuario.
iCloud	Sistema de almacenamiento en la nube o *cloud computing* de *Apple*.
IDE	Acrónimo del término inglés *Integrated Development Environment*. Es un entorno de desarrollo integrado para la construcción de software.
Intersticial:	Formato de un anuncio que ocupa toda la pantalla en la que se visualiza.
Intervalo	Referido a intervalo musical. Es la diferencia de frecuencia entre dos notas musicales medidas en tonos y semitonos.
iOS	Sistema operativo móvil de *Apple*.
Layout	Referido al diseño representa una plantilla de distribución de los elementos gráficos en pantalla.
Logro	Reconocimiento virtual que se puede definir en Game Center para premiar a los usuarios por sus habilidades con nuestra aplicación.
Markdown	Lenguaje de marcado que propone el *IDE* de *Apple* (Xcode) para comentar el código fuente. Tiene la ventaja de crear de forma automática una ayuda integrada con el IDE.
Mástil	Parte de la guitarra sobre la que se encuentran las cuerdas y los trastes.
ML	Machine Learning. Disciplina dentro del ámbito de la inteligencia artificial que crea sistemas que aprenden de forma autónoma.
MVC	Modelo Vista Controllador. Estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos.
Nota	Entendida como nota musical es el elemento a partir del cual se componen las melodías y armonías en la música.
Nube	Traducción del término inglés Cloud y que se refiere a aquellos servicios que se alojan en servidores a los que se accede a través de la red.
Open Source	Calificativo aplicado al software cuya programación está disponible para otros programadores ajenos a los creadores originales de forma gratuita y abierta a colaboración.
Patrón	Entendido como patrón musical es una abstracción sobre una figura musical en la que las notas concretas que la componen se sustituyen por la relación existente entre las mismas.
PDFKit	Librería *Open Source* que facilita la creación de ficheros con extensión *.pdf*.
Realm	Librería orientada a la gestión de bases de datos en aplicaciones móviles. Actualmente es la alternativa más extendida como sustitución al *Core Data* de *Apple*.
SDK	Acrónimo del término inglés *Software Development Kit*. Es un conjunto de herramientas para el desarrollo de software que permite a un programador desarrollar software sobre un sistema concreto.

Singleton	Patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.
SpriteKit	*Framework* de *Apple* orientado a la creación de videojuegos en dos dimensiones .
Storyboard	Herramienta gráfica que Xcode ofrece para la creación y diseño de las vistas de la aplicación móvil.
Swift	Lenguaje de programación creado por Apple enfocado al desarrollo de aplicaciones para iOS y macOS. Nace en 2014 con la idea de sustituir al obsoleto Objective-C.
TestFlight	Aplicación de *Apple* que permite distribuir las versiones *beta* entre los *beta tester* definidos para probar la aplicación.
TFM	Acrónimo de Trabajo Fin de Máster.
Tónica	Nota en un sistema tonal que hace referencia al primer grado de la escala musical. Es la nota que define la tonalidad.
Traste	División horizontal del mástil de la guitarra que delimita las zonas sobre las que puede sonar una nota.
UIKit	Framework de *Apple* para la construcción de la interfaz de usuario en aplicaciones sobre iOS o tvOS.
Usabilidad	Facilidad de uso que tiene una aplicación determinada.
Wireframe	Esquema que representa el esquema de la aplicación a un nivel de definición bajo.
Xcode	Entorno de desarrollo integrado para *macOS* formado por un conjunto de herramientas creadas por *Apple* destinadas al desarrollo de software para *macOS*, *iOS*, *watchOS* y *tvOS*.

BIBLIOGRAFÍA

[1] Appl Inc., "Apple Developer. SpriteKit, Documentation and Resources," [Online] Available: https://developer.apple.com/spritekit/. [Accesed: Feb 21, 2019]

[2] Appl Inc., "Apple Developer. GameKit. Framework Documentation," [Online] Available: https://developer.apple.com/documentation/gamekit. [Accesed: Feb 23, 2019]

Apple Inc., "Apple Developer. Game Center. Documentation and Resources." [Online] Available: https://developer.apple.com/game-center/. [Accesed: Feb 21, 2019]

[3] Apple Inc., "Apple Developer. Game Center. Programming Guide," [Online] Available: https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/

[4] Appl Inc., "Apple Developer. Using the CloudKit Framework," [Online] Available: https://developer.apple.com/documentation/cloudkit. [Accesed: March 4, 2019]

[5] Hal Leonard Corporation. Chord, scale and arpeggio finder : easy-to-use guide to over 1,100 [guitar] chords, 1,300 scales and 1,300 arpeggios. Milwaukee, WI : Hal Leonard, 2011

[6] R. E. Mayer. "Computer games in education," *Annual Review of Psychology,* Vol. 70 (January), pp. 531-549, 2019. [Online] Available: https://www.annualreviews.org/doi/10.1146/annurev-psych-010418-102744 [Accesed:: March 10, 2019]

[7] M. Margoudi [*et. al.*]. "Game-based learning of musical instruments: A review and recommendations," in *Proceedings of the European Conference on Games-based Learning*, 2016, Vol. January, pp. 426-433

[8] A. Baratè [*et. al.*], "Development of Serious Games for Music Education," *Je-LKS,* Vol. 9(2), pp. 89-104, 2013. [Online] Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0004364902340237 [Accesed: Feb 24, 2019]

[9] S. Fulqueris. *Armonía aplicada a la guitarra*. [s.l.] : Sergio Fulqueris, 2015

[10] raywenderlich.com Tutorial Team. 2D Apple Games by Tutorials: Beginning 2D iOS, tvOS, macOS & watchOS Game Development with Swift 3, [McGaheysville, Virginia] : Razeware LLC, 2016

[11] G. Theodoropoulos. "Documenting Your Swift Code in Xcode Using Markdown," *APPCODA,* 12 de mayo, 2016. [Online] Available: https://www.appcoda.com/swift-markdown/ [Accesed: Apr 4, 2019]

[12] Razeware LLC, raywenderlich.com, *Beginning CloudKit.* [Online] Available: https://www.raywenderlich.com/4247-beginning-cloudkit/lessons/1 . [Accesed: Apr 5, 2019]

ANEXOS

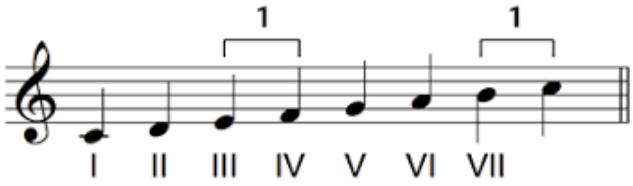
Anexo I: Fundamentos de teoría musical

En el estudio de cualquier instrumento se encuentran dos áreas bien diferenciadas: la técnica entendida como la habilidad necesaria para tocar el instrumento, y el conocimiento musical que es el que guía al músico para saber qué notas puede tocar en cada momento dentro de un contexto musical.

Este proyecto pretende ayudar en la segunda área ayudando al guitarrista a incorporar aquellos conocimientos abstractos que la técnica convertirá en sonidos.

Exponemos a continuación un breve resumen del conocimiento teórico musical que sirve como base a este proyecto fin de máster. La exposición no pretende -ni mucho menos- ser exhaustiva, y se añade aquí con el único propósito de ofrecer un mayor entendimiento al contexto del proyecto.

El pilar de la música occidental es la escala musical de siete notas. Es conocida por todos la escala mayor de *C*[29](*Do*)



Escala musical de Do (C)

Lo realmente importante de una escala es que realiza una selección sobre los doce tonos que empleamos en la música occidental. Cualquier escala identifica un patrón de intervalos[30] que navegan por estos doce sonidos.

[29] Se utilizará la notación anglosajona a la hora de nombrar las notas musicales: C-D-E-F-G-A-B en lugar de Do-Re-Mi-Fa-Sol-La-Si.

[30] Un intervalo identifica la distancia que hay entre dos notas en función de los semitonos que las separan en la escala.

La distancia mínima entre un sonido y otro es de un semitono. De esta forma lo que caracteriza a la escala mayor es una distribución de *tono-tono-semitono-tono-tono-tono-semitono*[31] que es lo que dota a la música creada sobre ella de una identidad y características propias.
Así pues, sobre esa selección de intervalos que forman una escala tenemos un conjunto de notas que suenan bien a un oído educado en la música occidental. Este conjunto de notas se pueden combinar para formar armonías (notas tocadas simultáneamente) sobre las que se añaden melodías (notas tocadas de forma secuencial).
Las agrupaciones armónicas pueden ser de dos ó más notas siendo las más conocidas los acordes de tres notas o triadas, pero existen muchas otras agrupaciones: quintas, tetradas, acordes compuestos, etcétera; mientras que en las melodías destacan el uso de los arpegios[32] así como de escalas subconjunto de la escala tocada.

Al tocar la guitarra es fundamental interiorizar todos los patrones que se forman en la guitarra a raíz de lo anteriormente expuesto. Por ejemplo, si queremos tocar con la escala de *C* tendremos que conocer -como mínimo- las posiciones de los acordes que se forman a partir de esta escala, las posiciones de los arpegios resultantes y, por supuesto, las posiciones de la escala en la que estamos.

La guitarra como instrumento tiene una particularidad que hace que el número de patrones se multiplique: las notas se encuentran distribuidas por el mástil entre las cuerdas del instrumento, lo que hace que no sea evidente las posiciones de las notas tal y como se puede apreciar en el siguiente gráfico:

[31] La escala mayor se identifica con el modo jónico identificado por los griegos, existen seis modos más que se corresponden con las interválicas resultado de recorrer la escala mayor desde cada uno de sus grados (posición de nota).

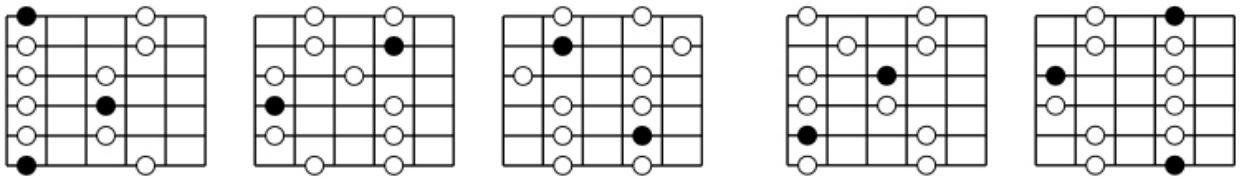
[32] Un arpegio se compone de las notas de un acorde tocadas de forma secuencial.

	1	2	3	4	5	6	7	8	9	10	11	12
E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E
B	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
G	G#	A	A#	B	C	C#	D	D#	E	F	F#	G
D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D
A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A
E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**

Mástil de una guitarra con afinación estándar

Por si esto fuera poco, el guitarrista tiene que estar preparado para tocar en cualquier posición del mástil y preparado para tomar el camino más rápido desde donde tenga situada su mano. De esta forma aparecen nuevos patrones según nos desplazamos horizontalmente a lo largo del mástil y, además, aparecen nuevos patrones según ataquemos con un dedo u otro[33].

Para entender mejor lo expuesto vamos a mostrar el patrón de una de las escalas más utilizadas en el rock: la escala pentatónica menor. Esta escala es un subconjunto derivado de la escala menor en la que se han eliminado las distancias de semitonos. Al movernos por el mástil aparecen cinco posiciones básicas según dónde se encuentra la tónica[34] (puntos rellenos).
En cada gráfico se identifican las cuerdas (líneas horizontales) y los trastes[35] (espacios delimitados por las líneas verticales).



Posiciones de la escala pentatónica menor

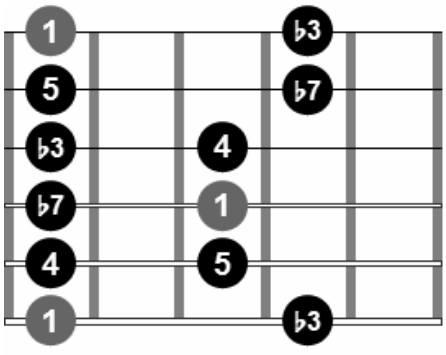
[33] Los patrones se pueden atacar con todos los dedos salvo con el pulgar que permanece en la parte posterior del mástil.

[34] La tónica es la primera nota de la escala siendo esta la que da nombre a la escala. Por ejemplo en la escala de E mayor se parte de E.

[35] Al movernos en trastes consecutivos nos estamos desplazando un semitono.

Dentro de todo el mar de patrones que se generan hay una buena noticia: lo importante de los patrones no es tanto aprenderse las notas que los conforman sino los intervalos que aparecen. Las notas de la escala pentatónica de C y de D son totalmente diferentes, pero los intervalos siempre los mismos. De esta forma basta con saber situar la tónica del patrón y a partir de ahí tocar sobre el patrón correspondiente que tendremos grabado a fuego en nuestra mente de guitarrista.

Es decir, para cada patrón que tengamos que aprender 'sólo' será necesario memorizar el esquema identificando la posición de la tónica y la posición de los intervalos del resto de notas en referencia a dicha tónica.
En el siguiente ejemplo se pueden ver los intervalos que conforman una de las posiciones de la escala pentatónica menor:



Interválica de una posición de la escala pentatónica menor

Así, en este patrón es imprescindible aprender que teniendo la tónica (1) en la sexta cuerda (la inferior del gráfico), tres trastes -tres semitonos por tanto- a la derecha tenemos una tercera menor (3b), justo debajo de la tónica en la quinta cuerda tenemos una cuarta justa (2 tonos y medio), etc.

Nuestra capacidad de creación musical aumenta en la medida en que conocemos estos patrones y nos movemos de forma fluida y dinámica por los mismos. En el caso del gráfico anterior podemos haber tocado un intervalo de quinta (5) y ahora realizar un desplazamiento tonal en el que convertimos esa

quinta en la tónica, surgirá otro gráfico muy diferente resultado de convertir mentalmente la quinta en tónica y empezar a movernos por uno o varios de los nuevos patrones que surgen a raíz de dicho movimiento.

Estos patrones de intervalos se concretarán en un futuro en notas. Por ejemplo, si el patrón de un acorde mayor séptima es 1-3-5-7, será importante saber que si la tónica es C (Do) tendremos C-E-G-Bb, mientras que si es E las notas que encontraremos son E-G-B-D.
Al trabajar con notas concretas en lugar de intervalos nos aparece el problema teórico de los enarmónicos. Una nota musical es enarmónica de otra cuando ambas representan el mismo sonido (o tono) pero se representan de forma diferente. Este problema se arrastra desde la Edad Media cuando se creó la escritura musical pensando únicamente en la escala diatónica (teclas blancas del piano). Cuando se fueron añadiendo nuevos sonidos no había sitio para escribirlos en el pentagrama por lo que se los nombró como alteraciones de notas existentes (sostenidos o bemoles). Un sostenido eleva la nota un semitono, mientras que un bemol la baja un semitono. De esta forma, para nombrar el tono que se encuentra entre las notas C (Do) y D(Re) podemos elegir entre: C# (Do sostenido), Db (Re bemol), B## (Si doble sostenido) y Ebb (Mi doble bemol). Aunque el sonido es el mismo es necesario precisar el nombre de acuerdo con la teoría musical. Dicho nombre dependerá exclusivamente de la tonalidad en la que estemos y el grado que ocupe la nota dentro de dicha tonalidad.
En cualquier caso, estas peculiaridades nos llevan a adentrarnos en la complejidad de la armonía musical [7] lo cual queda fuera del ámbito de esta breve introducción a la armonía musical. Baste con entender la complejidad que conlleva la concreción de los patrones musicales dentro de una tonalidad.

Anexo II: Herramientas empleadas

Se citan a continuación, por orden alfabético, las herramientas que se han utilizado en la elaboración de este proyecto tanto en su fase de diseño como de implementación.

CocoaPods [cocoapods.org]
Utilizamos *CocoaPods 1.6.1* para la gestión de las librerías utilizadas en el desarrollo.

Git [github.com]
Utilizamos *GitHub* como repositorio. A la hora de realizar el control de versiones utilizamos *Git* en modo comando; pese a estar integrado con *Xcode* encontramos que esta integración es todavía muy limitada. Sí utilizamos el entorno de *Xcode* a la hora de comparar ficheros de versiones ya que la ayuda gráfica en este caso es muy buena.
Todo el código de nuestro proyecto en sus diferentes versiones puede encontrarse en *https://github.com/abanet/TFMGuitar*.

Mockflow [mockflow.com]
Mockflow es una herramienta online para la visualización de conceptos que incluye entre sus posibilidades la creación de *wireframes*. Se ha utilizado en la construcción del *wireframe* de la *app*.

Pages [https://www.apple.com/es/pages/]
Editor de textos de Apple utilizado en la escritura y maquetación de la memoria final.

ScreenFlow [https://www.telestream.net/screenflow/]
Herramienta de edición de vídeo que facilita la grabación de la pantalla así como la de dispositivos móviles conectados al ordenador. Se ha empleado a la hora de generar el vídeo de presentación final del proyecto.

Sketch [sketch.com]
Herramienta de diseño vectorial, utilizada tanto para la elaboración del *wireframe* de alta definición como para los gráficos finales de la aplicación.

Sketchboard [sketchboard.me]
Herramienta de creación de diagramas online que hemos utilizado para los diagramas de los casos de uso y de definición de la arquitectura del sistema.

TeamGantt [teamgantt.com]
Herramienta online para la planificación de proyectos y realización de diagramas de *Gantt*.

Trello [trello.com]
Herramienta online que permite el seguimiento del estado de las tareas de un proyecto de una forma sencilla y muy gráfica basada en la organización de las tareas en pizarras.
Tiene la gran ventaja de integrarse con *TeamGantt* con lo que cualquier cambio en un entorno queda reflejado automáticamente en el otro.

Xcode e Instruments [https://developer.apple.com/xcode/]
Se utilizó el entorno de desarrollo ofrecido por *Apple* pese a que se evaluó utilizar *AppCode* de *Jet Brains*. Finalmente decidimos trabajar en el entorno que ya conocíamos para evitar posibles retrasos debidos al desarrollo en un entorno en que no tenemos experiencia alguna.
La versión utilizada ha sido la *10.2.1* sobre un *iMac* con *macOS Mojave* versión *10.14.4*.

Xtensio [xtensio.com]
Se utiliza la herramienta colaborativa online *Xtensio* en la realización de las fichas de usuario.