



ADD: MODULO DE JUEGO

Carlos Andres Castaño Bustos

2023-11-24

Índice de contenidos

1	Requisitos	2
1.1	Requisitos funcionales	2
1.2	Requisitos no funcionales	2
2	Motivadores de la iteración	3
3	Elementos a refinar	4
4	Conceptos que satisfacen los motivadores	5
5	Elementos de la arquitectura y responsabilidades	7
6	Diseño de vistas	9
7	Validación y análisis de resultados	18

1 Requisitos

1.1 Requisitos funcionales

- El módulo debe estar en la capacidad de de mostrar diferentes canciones con el mismo código.
- El módulo debe estar en la capacidad de que las mecánicas, puntuación, combo y precisión funcionen de la misma forma independiente del instrumento musical.
- El módulo debe mostrar el puntaje del jugador en tiempo real durante el desafío musical.
- El módulo debe mantener un multiplicador de combo que aumente cuando el jugador toca notas correctamente y pase a 0 en caso de errores.
- El módulo debe proporcionar un valor que indique el porcentaje de precisión en el desafío.
- El módulo debe incluir un botón de pausa que permita a los jugadores pausar el desafío en cualquier momento.
- El módulo debe permitir desde el menú de pausa, que los jugadores opten por salir del desafío, reiniciar el desafío o retomarlo después de una pausa.
- El módulo debe permitir guardar los datos del resultado final de cada desafío.

1.2 Requisitos no funcionales

- Se debe proporcionar una interfaz de usuario que sea clara y fácil de entender.
- La implementación debe de ser modular y escalable, permitiendo que se puedan implementar a futuro nuevos servicios de almacenamiento de información adicionales, sin que esto requiera cambios en el código principal del módulo.

2 Motivadores de la iteración

MOTIVADOR	DESCRIPCIÓN
Sistema modular y escalable	<ul style="list-style-type: none">• Debe de ser independiente de los demás módulos del sistema.• Debe de permitir la integración de diferentes servicios de almacenamiento de información.• Debe de estar en la capacidad de cambiar el servicio de almacenamiento de información utilizado sin necesidad de cambiar el código principal del sistema.• Debe permitir la inclusion de diferentes tipos de instrumentos musicales.

Table 1: Componentes Motivadores. Fuente propia

3 Elementos a refinar

COMPONENTE	DESCRIPCIÓN
Módulo de juego	<p>Este componente estará encargado de varios aspectos clave como:</p> <ul style="list-style-type: none">• Mostrar en pantalla las canciones con sus respectivos elementos dinámicos como las notas musicales, puntaje, combo y precision.• Realizar la detección del momento en que el usuario toca las notas para sus respectivos cálculos.• Crear, editar y eliminar los mapas de notas musicales de los diferentes desafíos musicales.• Permitir al usuario pausar, continuar, reiniciar o salir del desafío musical.• En el modo de juego practica, se debe mostrar la pantalla de resultado final, pero no guardar estos datos.• En el modo de juego solo, se debe mostrar la pantalla de resultado final, y guardar estos resultado con el servicio de datos seleccionado.

Table 2: Elementos a refinar. Fuente propia

4 Conceptos que satisfacen los motivadores

ESTILO ARQUITECTURA	JUSTIFICACIÓN
Cliente-Servidor	El estilo de arquitectura que se ajusta a las necesidades del proyecto es el de Cliente-Servidor, ya que este implica una separación clara entre el cliente (en este caso, las vistas del juego desarrolladas en Unity) y el servidor (donde reside la lógica de almacenamiento de la información).

Table 3: Estilo de arquitectura. Fuente propia

PATRON DE DISEÑO	JUSTIFICACIÓN
Patrón Strategy	Este patrón permite definir diferentes estrategias de almacenamiento de información (por ejemplo, el servicio de Cloud Save de Unity Services o la Realtime Database de Firebase u otro adicional) como clases separadas. De esta manera, el sistema puede cambiar dinámicamente el servicio de almacenamiento de información en tiempo de ejecución según las necesidades del usuario o del administrador del sistema. Este enfoque promueve la flexibilidad y la mantenibilidad del sistema, ya que cada estrategia puede ser modificada, reemplazada o ampliada sin afectar el resto del código.

Continúa en la siguiente página.

PATRON DE DISEÑO	JUSTIFICACIÓN
Patron Factory	Este patrón es el ideal para crear objetos del proveedor de almacenamiento de información basados en la estrategia seleccionada. La fábrica actúa como un intermediario entre el cliente y los servicios concretos de almacenamiento de información, permitiendo que el cliente solicite una instancia del proveedor sin conocer los detalles de implementación de esa instancia. Esto facilita la creación centralizada y dinámica de diferentes tipos de proveedor de almacenamiento de información (por ejemplo, una instancia del servicio de Cloud Save de Unity Services o una de la Realtime Database de Firebase) según las necesidades del sistema o del usuario. Además, el uso del patrón Factory asegura que la creación de las instancias sea coherente y estandarizada, lo que simplifica la lógica del cliente y mejora la mantenibilidad del código.
Patrón MVP (Model-View-Presenter)	En este enfoque, los modelos manejan la estructura de los datos, mientras que las vistas se centran exclusivamente en la presentación visual de la información. Los Presentadores facilitan la comunicación entre los Modelos, las Vistas y los servicios, permitiendo una manipulación precisa de la interfaz gráfica en respuesta a las acciones del usuario y los resultados de la autenticación. Esta separación de responsabilidades no solo simplifica el desarrollo y la depuración del código, sino que también mejora la flexibilidad y la mantenibilidad del sistema al permitir cambios en la lógica de negocio o en la interfaz de usuario de forma independiente.
ScriptableObjects	La idea principal de utilizar este patrón es separar los datos de la lógica y permitir de esta manera una mayor reutilización y flexibilidad. Esto nos viene muy bien para la estructura de este módulo ya que se tienen muchos componentes dinámicos que implicarían una complejidad y alto acoplamiento si decidiéramos no usar ScriptableObjects. Adicionalmente, hay que destacar que estos nos brindan una mayor reusabilidad, la cual necesitamos para poder plasmar los diferentes desafíos musicales y también una vez construido, su configuración y utilización es muy sencilla y rápida.

Table 4: Patrones de diseño. Fuente propia

5 Elementos de la arquitectura y responsabilidades

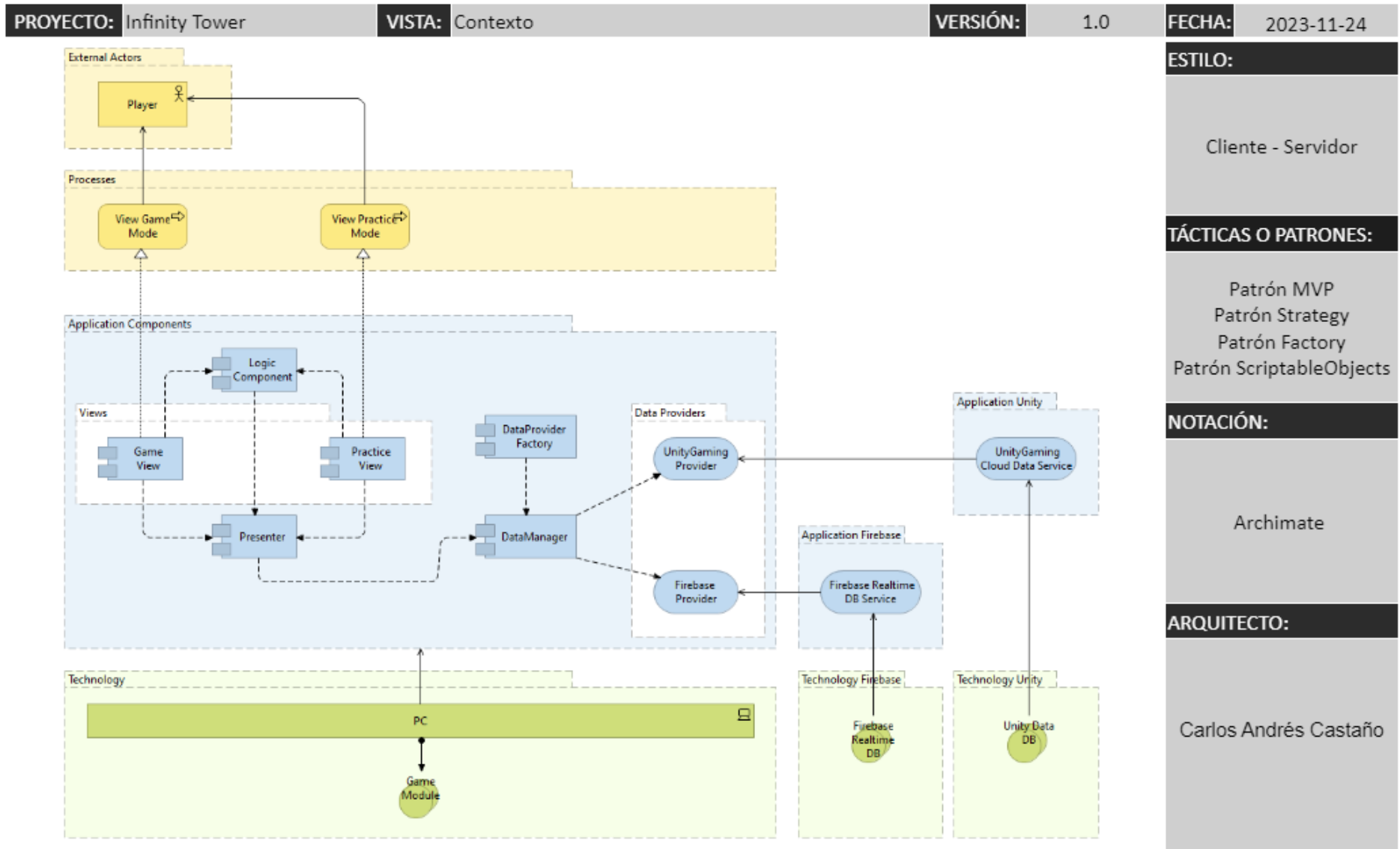
NOMBRE	RESPONSABILIDAD	RELACIONES
Vistas	Las Vistas se encargan de mostrar la interfaz gráfica al usuario final. Son responsables de representar visualmente la información y de capturar las interacciones del usuario, como los eventos de cambio de vista, actualización de datos del juego, etc.	<ul style="list-style-type: none">• Presenter
Presenter	El Presenter actúa como intermediario entre las Vistas, el Componente de Lógica y el DataManager. Se encarga de recibir los eventos del usuario desde las Vistas, procesarlos y coordinar las acciones necesarias con el Componente de Lógica y el DataManager. También es responsable de actualizar las Vistas en función de los resultados del Componente de Lógica.	<ul style="list-style-type: none">• Vistas• Componente de Lógica• DataManager
Componente de Lógica	este es el componente central del módulo de juego. Su función principal es tener toda la Lógica y ScriptableObjects necesarios para el correcto funcionamiento de los desafíos musicales, lo cual incluye datos y cálculos y eventos.	<ul style="list-style-type: none">• Vistas• Presenter
DataManager	DataManager es el componente encargado del almacenamiento de datos del usuario. Su función principal es coordinar las solicitudes para guardar los resultados de los diferentes desafíos musicales y gestionar las estrategias de información proporcionadas por el DataProviderFactory.	<ul style="list-style-type: none">• Presenter• DataProviderFactory

Continúa en la siguiente página.

NOMBRE	RESPONSABILIDAD	RELACIONES
DataProviderFactory	El DataProviderFactory se encarga de crear instancias de los Proveedores de información según la solicitud del DataManager. Permite una creación centralizada y dinámica de los diferentes proveedores de información, proporcionando flexibilidad para adaptarse a diversas estrategias.	<ul style="list-style-type: none"> • DataManager
Proveedor de información	El Proveedor de información representa a un proveedor de servicios como UnityGamingServices o Firebase, que ofrecen estrategias de almacenamiento de información robustas y seguras. Su función principal es proporcionar servicios de almacenamiento de información mediante servicios.	<ul style="list-style-type: none"> • MapsManager • Base de datos proveedor de información
Base de datos proveedor de información	La Base de Datos del Proveedor de información almacena y gestiona los datos. Cada instancia de la Base de Datos está asociada con un proveedor específico y es responsable de gestionar la lectura y escritura de datos relacionados con la información para esa estrategia particular.	<ul style="list-style-type: none"> • Proveedor de información

Table 5: Componentes del sistema. Fuente propia

6 Diseño de vistas



PROYECTO: Infinity Tower

VISTA: Funcional - Nivel 1

VERSIÓN:

1.0

FECHA:

2023-10-19

ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

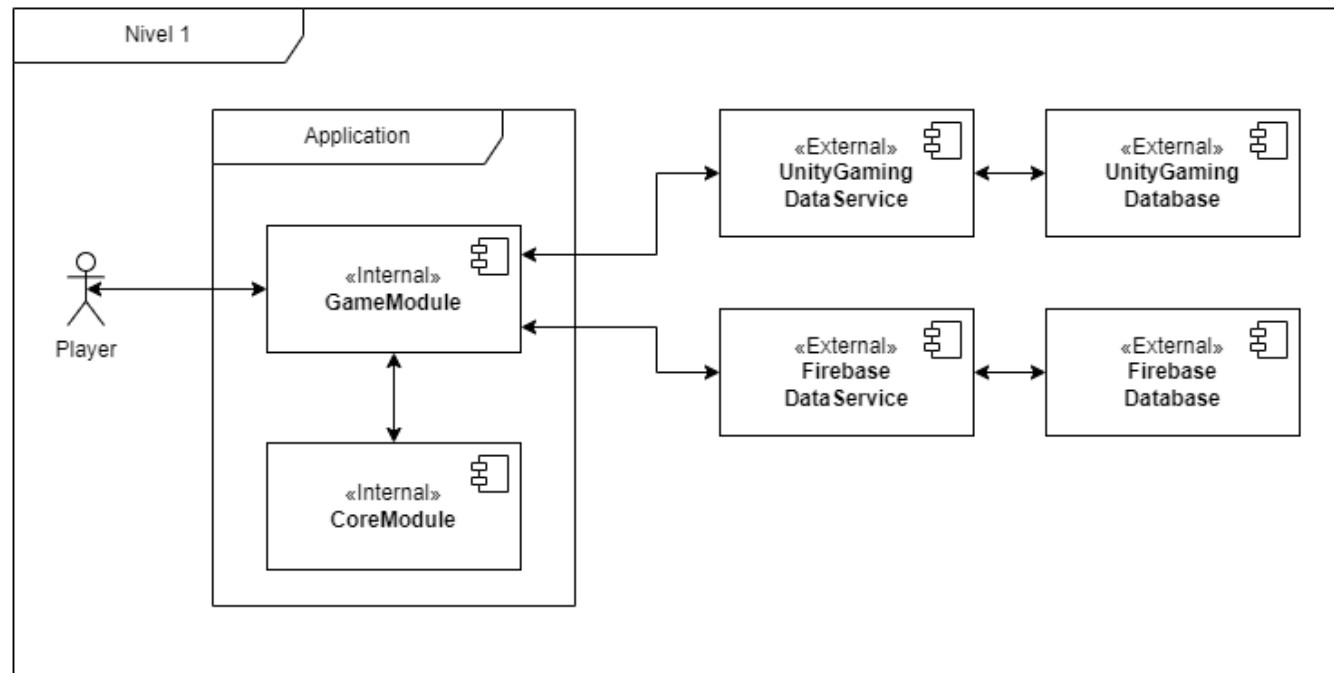
Patrón MVP
Patrón Strategy
Patrón Factory
Patrón ScriptableObjects

NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño



PROYECTO: Infinity Tower

VISTA: Funcional - Nivel 2

VERSIÓN:

1.0

FECHA:

2023-11-24

ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

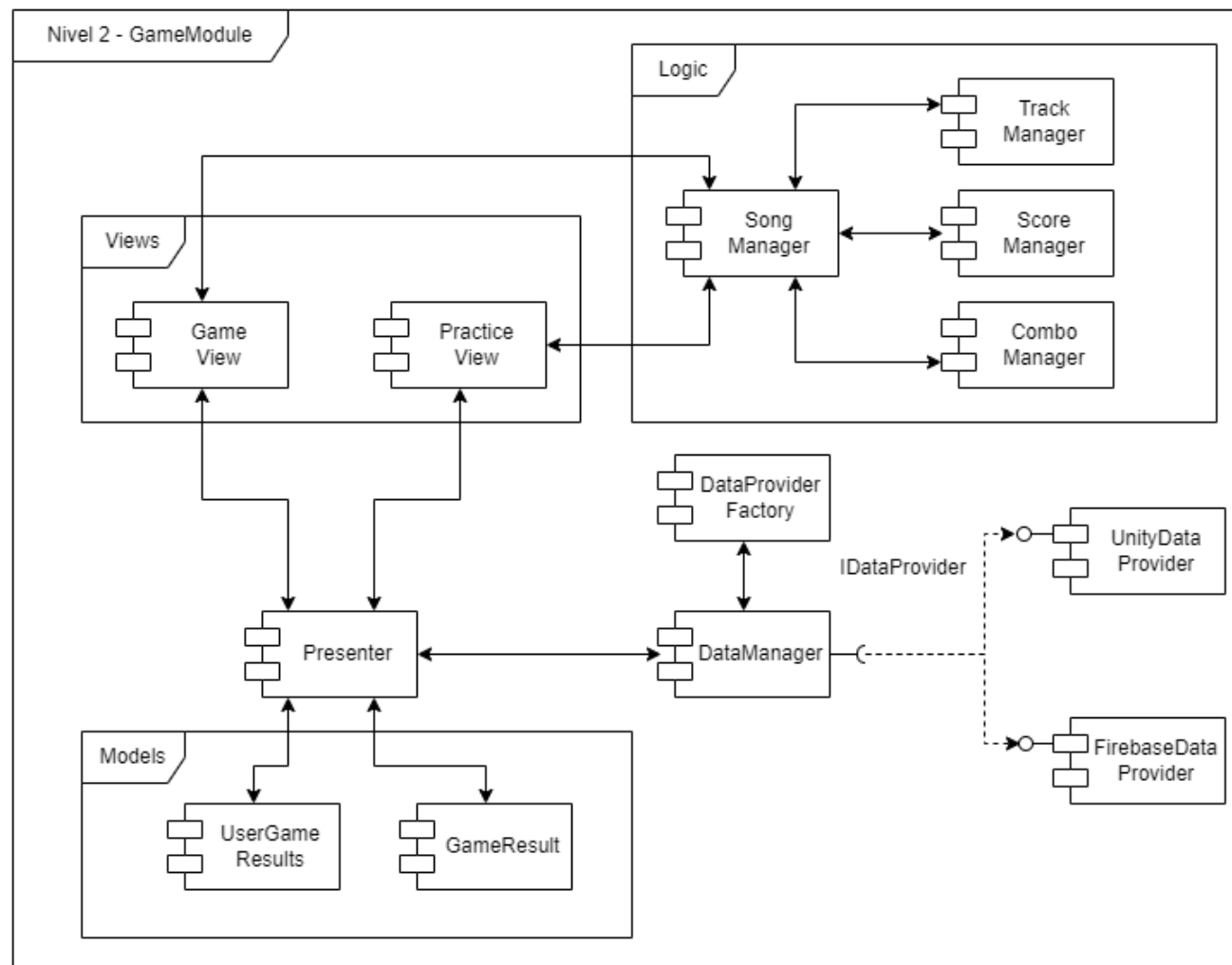
Patrón MVP
Patrón Strategy
Patrón Factory
Patrón ScriptableObjects

NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño



PROYECTO:	Infinity Tower	VISTA:	Información - Models	VERSIÓN:	1.0	FECHA:	2023-10-19
------------------	----------------	---------------	----------------------	-----------------	-----	---------------	------------

ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

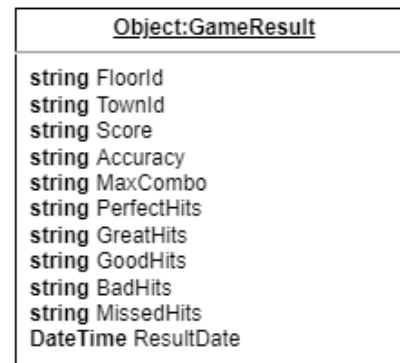
Patrón MVP
 Patrón Strategy
 Patrón Factory
 Patrón ScriptableObjects

NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño



<u>ScriptableObject: SongItem</u>
AudioClip Clip string Name string Author int Bpm MetadataList Metadata bool UseCurrentBpmMidilImport List<MidiNote> Notes bool UseGenerator bool RuntimeGenerate float OnsetSensitivity float[] OnsetData float OnsetMin float OnsetMax

<u>ScriptableObject: MidiNote</u>
NoteName NoteName int NoteOctave float Time float NoteLength float BeatIndex float BeatLengthIndex

<u>ScriptableObject: Metadata</u>
string Id int IntValue float FloatValue string StringValue Object ObjectReference

<u>ScriptableObject: MidiTrackMapping</u>
int referenceRootOctave bool ignoreOctave List<TrackMapping> mapping

<u>ScriptableObject: TrackMapping</u>
NoteName NoteTarget int NoteOctave

<u>ScriptableObject: NotePrefabMapping</u>
int referenceRootOctave bool ignoreOctave bool ignoreName List<PrefabPoolEntry> notesPrefab List<NoteMap> mapping

<u>ScriptableObject: NoteMap</u>
int NotePrefabIndex NoteName NoteName int NoteOctave

<u>ScriptableObject: PrefabPoolEntry</u>
GameObject Prefab int PoolSize

ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

Patrón MVP
 Patrón Strategy
 Patrón Factory
 Patrón ScriptableObjects

NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño

ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

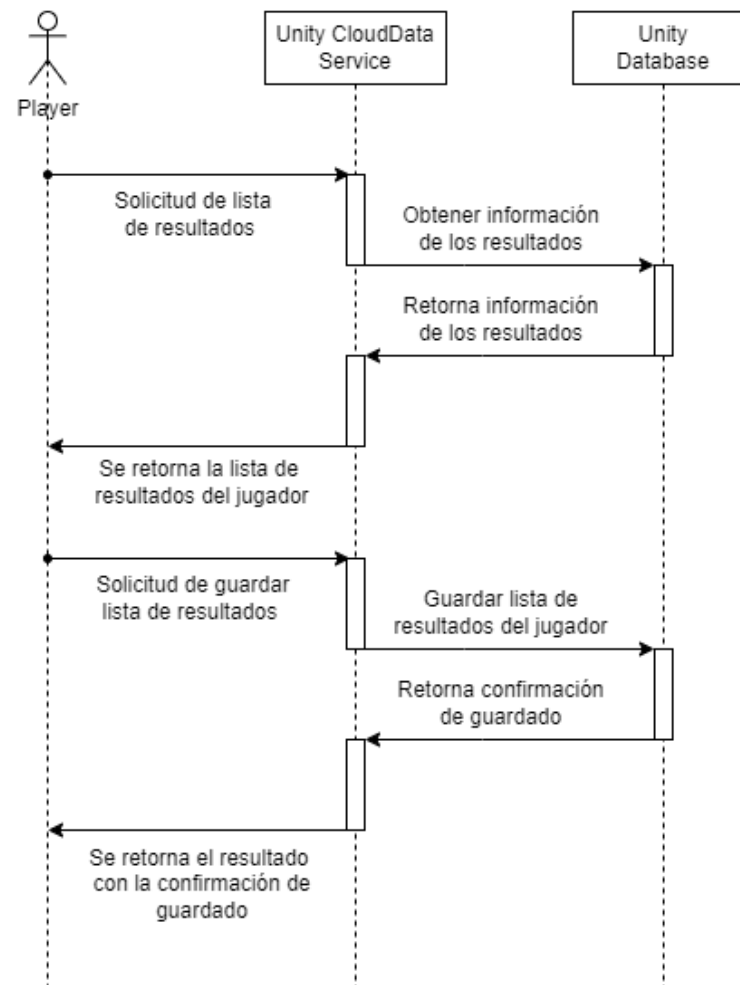
Patrón MVP
 Patrón Strategy
 Patrón Factory
 Patrón ScriptableObjects

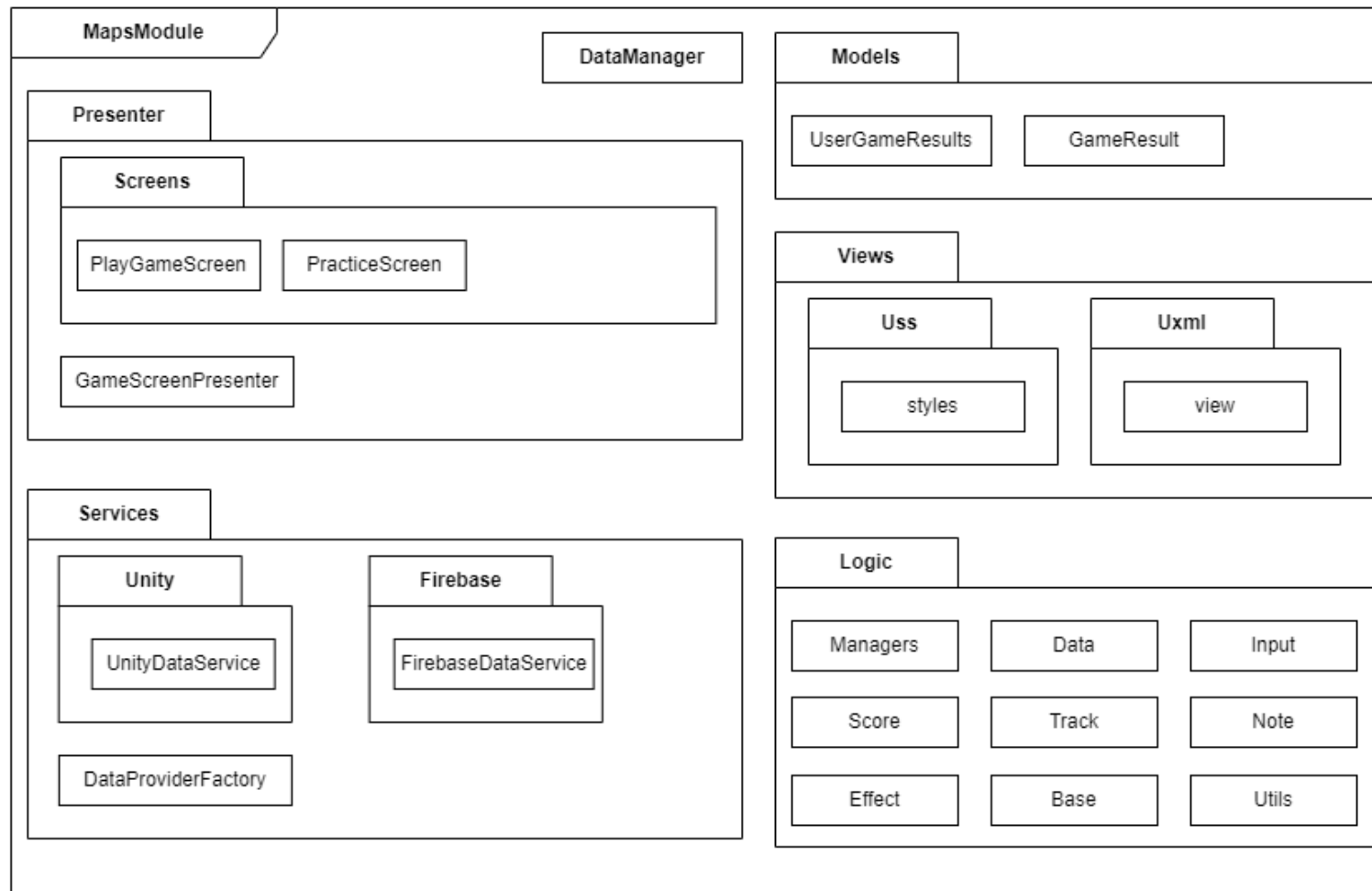
NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño





ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

Patrón MVP
Patrón Strategy
Patrón Factory
Patrón ScriptableObjects

NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño

PROYECTO: Infinity Tower

VISTA: Operación - Practice

VERSIÓN: 1.0

FECHA: 2023-11-24

ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

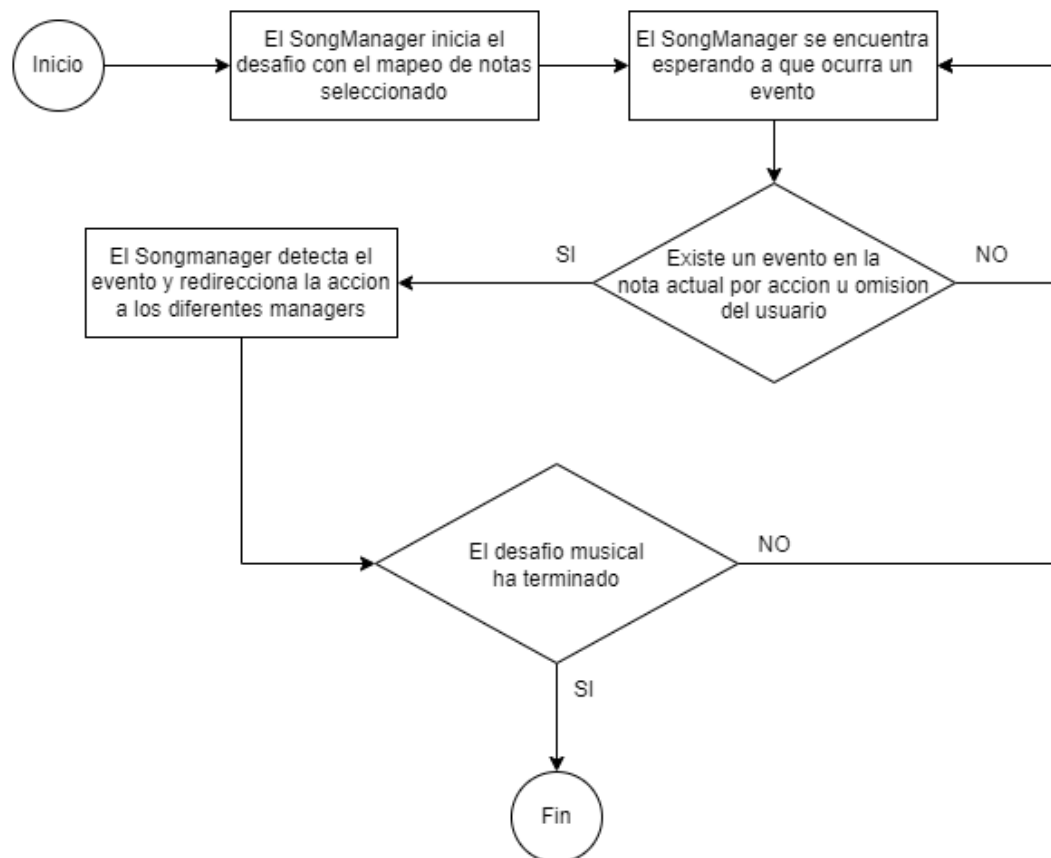
Patrón MVP
Patrón Strategy
Patrón Factory
Patrón ScriptableObjects

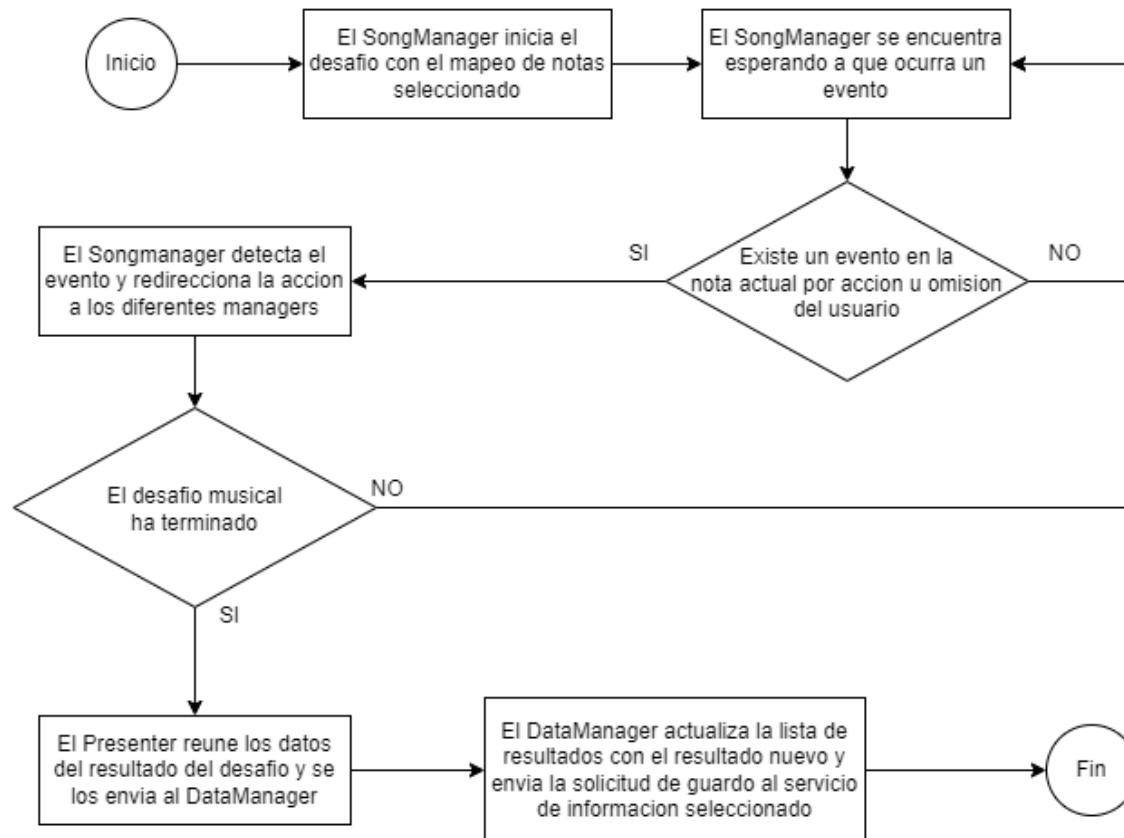
NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño




ESTILO:

Cliente - Servidor

TÁCTICAS O PATRONES:

Patrón MVP
 Patrón Strategy
 Patrón Factory
 Patrón ScriptableObjects

NOTACIÓN:

UML

ARQUITECTO:

Carlos Andrés Castaño

7 Validación y análisis de resultados

	DESCRIPCIÓN
RESULTADOS	<p>Tomando como base el diseño de la arquitectura realizado para los modulos anteriores, este de igual forma ha producido un buen resultado transformando el código en algo comprensible, fácilmente mantenible y adaptable para la inclusión de nuevos servicios de información.</p> <p>Estos patrones han proporcionado una estructura coherente que ha simplificado la complejidad del sistema. Esta flexibilidad es crucial para garantizar que el código pueda evolucionar con el tiempo, adaptándose a nuevos requisitos y desafíos sin comprometer la estabilidad del sistema existente.</p> <p>La inclusion de ScriptableObjects para gestionar los objetos dinámicos fue una muy buena elección. Ya que esta decisión, ha permitido que el módulo sea versátil al enfrentarse a diversos desafíos musicales sin necesidad de realizar modificaciones en el código principal. La única variación reside en los ScriptableObjects, los cuales se adaptan según las necesidades específicas de cada caso, proporcionando así una solución flexible y eficiente.</p>
ACCIONES A SEGUIR	<p>Dado que los resultados del diseño de la arquitectura han sido satisfactorios y han cumplido con los objetivos establecidos, la acción a seguir es llevar a cabo la implementación completa del modulo. Este paso implica traducir el diseño abstracto en código funcional, asegurando que todas las decisiones arquitectónicas se reflejen fielmente en la implementación.</p> <p>Las pruebas rigurosas también deben llevarse a cabo para validar que la implementación se ajusta a las expectativas y que todas las funcionalidades operan como se espera.</p>

Table 6: Resultados y acciones a seguir. Fuente propia