Diseño e implementación de prototipo de un videojuego enfocado en el desarrollo de destrezas musicales

Carlos Andrés Castaño Bustos

Nota de Aceptación

Certificamos que el presente Trabajo de Grado Satisface, en alcances y calidad, todos los requisitos que demanda un Trabajo de Grado de Maestría.

Director

David Baldeón M.Sc

Andrés A. Navarro Newball, Ph.D

Aprobado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana Cali, para optar el título de **Magíster en Ingeniería de Software**.

HERNÁN CAMILO ROCHA NIÑO Ph. D.
Decano Facultad de Ingeniería y Ciencias

JUAN CARLOS MARTÍNEZ ARIAS
Director Posgrados de Ingeniería y Ciencias

Santiago de Cali, 15 de Abril de 2024

Acta de Correcciones al Documento de Trabajo de Grado

Santiago de Cali, 15 de Abril de 2024

Autor: Carlos Andrés Castaño Bustos

Título del Trabajo de Grado: Diseño e implementación de prototipo de un videojuego enfocado en el desarrollo de destrezas musicales

Director:

Como indica el artículo 2.13 de las Directrices para Trabajo de Grado de Maestría, he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Trabajo de Grado definieron que se efectuarán, como consta en el Acta de Evaluación correspondiente.

Firma del Director del Trabajo de Grado

Ficha Resumen
Trabajo de Grado de Maestría

Título: Diseño e implementación de prototipo de un videojuego enfocado en el desarrollo de destrezas musicales

1. Tipo de proyecto (Aplicado, Innovación, Investigación): Innovación
2. Área de trabajo: Desarrollo de videojuegos y enseñanza musical
3. Estudiante: Carlos Andrés Castaño Bustos
4. Correo electrónico: cabu0124@gmail.com
5. Dirección y teléfono: Carrera 66 #33B-35 - 3152249252
6. Director: Gerardo Mauricio Sarria
7. Vinculación del director: Director of the Undergraduate Studies in Computer Science
8. Correo electrónico del director: gsarria@javerianacali.edu.co
9. Co-Director (Si aplica): No aplica
10. Grupo o empresa que lo avala (Si aplica): No aplica
11. Otros grupos o empresas: No aplica
12. Palabras clave(al menos 5): Videojuego, Aprendizaje musical, Práctica musical, Juego de rol, Motivación, Innovación, Entretenimiento.
13. Fecha de inicio: 24 de Julio de 2023
14. Resumen: Desarrollo de un videojuego, que combina las aventuras de juegos de rol con el aprendizaje musical para fomentar la práctica de instrumentos musicales y abordar la falta de motivación en el aprendizaje ofreciendo una

alternativa innovadora. El resultado de este proyecto es el prototipo del juego el cual puede ser usado para entretenimiento, aprendizaje en escuelas, conservatorios, centros de música, etc.

Diseño e implementación de prototipo de un videojuego enfocado en el desarrollo de destrezas musicales

Carlos Andres Castaño Bustos

Proyecto de grado presentada(o) como requisito parcial para optar al título de:
Magister en Ingeniería de Software

Director(a):
Ph.D. Gerardo Mauricio Sarria

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias
Departamento de Electrónica y Ciencias de la Computación
Cali, Colombia
15 de abril de 2024

Índice

1. Introducción **11**

2. Identificación del problema **12**

2.1. Causas del problema 14

2.2. Efectos negativos del problema 15

2.3. Formulación del problema 16

3. Objetivos del proyecto **17**

3.1. Objetivo General 17

3.2. Objetivos específicos 17

3.3. Resultados obtenidos 18

4. Alcance **19**

5. Justificación del trabajo de grado **21**

6. Marco teórico de referencia y antecedentes **23**

6.1. Bases Teóricas 23

6.2. Estado del Arte 53

7. Metodología del proyecto **58**

7.1. Planificación 58

7.2. Notaciones 59

7.3. Ventajas y beneficios 61

8. Desarrollo trabajo de grado **63**

8.1. Diseño del documento del juego 63

8.2. Diseño de interfaz gráfica del juego 65

8.3. Diseño de la arquitectura de software 68

8.4. Implementación del prototipo 82

8.5. Resultados del prototipo 98

9. Conclusiones 102

10. Anexos 104

11. Referencias Bibliográficas 105

12. Glosario de Términos 107

Índice de figuras

1. Interés de las personas en diferentes entretenimientos. (Newzoo, 2023) 12
2. Entretenimientos practicados de manera activa. (Newzoo, 2023) . . . 13
3. Mundo digital vs Mundo real. (Newzoo, 2023) 15
4. Pilares fundamentales y su relación entre ellos. Fuente propia 25
5. Fonomímia de Kodály. Fuente (Fontelles Rodríguez, 2019) 29
6. Clase Player con múltiples responsabilidades. (Unity, 2021a) 35
7. Diseño refactorizado en diferentes clases con responsabilidades únicas. (Unity, 2021a) 36
8. Clase AreaCalculator con diferentes métodos para calcular el área. (Unity, 2021a) 37
9. Diseño refactorizado de la clases usando el principio abierto/cerrado. (Unity, 2021a) 38
10. Estructura de clases y subclases. (Unity, 2021a) 39
11. Comportamiento de los tipos de vehículos. (Unity, 2021a) 40
12. Violación del principio de sustitución de Liskov. (Unity, 2021a) 40
13. Diseño refactorizado de la clases usando el principio de sustitución de Liskov. (Unity, 2021a) 42
14. Estructura de clases implementando una única interfaz. Fuente propia 43
15. Diseño refactorizado de la interfaz dividida usando el principio de segregación de la interfaz. (Unity, 2021a) 44
16. Diseño de clases con alto nivel de acoplamiento. (Unity, 2021a) 45
17. Diseño refactorizado con la interfaz usando el principio de inversión de la dependencia. (Unity, 2021a) 46
18. Ejemplo practico del uso del Factory Pattern. (Unity, 2021a) 48
19. Proceso de guardar acciones mediante el patrón Command. (Unity, 2021a) 49
20. Relación entre el sujeto y los observadores. (Unity, 2021a) 51
21. Flujo de comunicación entre capas usando MVP. (Unity, 2021a) . . . 52
22. Yousician Apps. Fuente (Yousician, 2023) 53

23. Simply Apps. Fuente (Simply, 2023) . . . 54
24. Sing Sharp App. Fuente (Sing Sharp, 2023) . . . 55
25. Synthesia App. Fuente (Synthesia, 2023) . . . 56
26. Musikami App. Fuente (Sánchez Rueda, 2017) . . . 57
27. Milestones del proyecto. Fuente propia . . . 58
28. Etiquetas de historias de usuario. Fuente propia . . . 60
29. Etiquetas de tareas. Fuente propia . . . 61
30. Diseño módulo de autenticación. Fuente propia . . . 65
31. Diseño módulo de menus. Fuente propia . . . 66
32. Diseño módulo de mapas. Fuente propia . . . 67
33. Diseño módulo de juego - Desafío. Fuente propia . . . 67
34. Diagrama de contexto módulo core. Fuente propia . . . 70
35. Diagrama de contexto módulo auth. Fuente propia . . . 73
36. Diagrama de contexto módulo menus. Fuente propia . . . 76
37. Diagrama de contexto módulo mapas. Fuente propia . . . 79
38. Diagrama de contexto módulo de juego. Fuente propia . . . 81
39. Estructura del proyecto. Fuente propia . . . 82
40. Plugins del proyecto. Fuente propia . . . 83
41. Recursos del proyecto. Fuente propia . . . 83
42. Estructura módulo core. Fuente propia . . . 84
43. Contenidos módulo core en Unity Editor. Fuente propia . . . 85
44. Datos en Unity Cloud. Fuente propia . . . 86
45. Estructura módulo de autenticación. Fuente propia . . . 87
46. Implementación nuevo servicio de autenticación. Fuente propia . . . 88
47. Estructura módulo de menus. Fuente propia . . . 89
48. Implementación nuevo servicio de datos en módulo de menus. Fuente propia . . . 90
49. Estructura módulo de mapas. Fuente propia . . . 91
50. Implementación nuevo servicio de datos en módulo de mapas. Fuente propia . . . 92
51. Estructura módulo de juego. Fuente propia . . . 93

52. Prefab base GameCore. Fuente propia 94
53. Variante GameCore implementación de pista de notas en el módulo de juego. Fuente propia 95
54. Propiedades físicas de objetos - Unity. Fuente propia 96
55. Implementación nuevo servicio de datos en módulo de juego. Fuente propia 97
56. Mapeo de ScriptableObject en Editor de Unity. Fuente propia 98
57. Progreso - Nivel 6. Fuente propia 99
58. Progreso - Nivel 7. Fuente propia 100
59. Progreso - Nivel 8. Fuente propia 100
60. Progreso - Nivel 9. Fuente propia 101

Índice de tablas

1. Fonomímia de Kodály. Fuente (Fontelles Rodríguez, 2019) 28

Resumen

El proyecto describe el desarrollo e implementación del prototipo de un videojuego de aventuras y música que involucra instrumentos reales como forma de interactuar con el juego y que permite al usuario mejorar sus habilidades musicales. Para lograrlo, se emplearon metodologías de enseñanza musical y se diseño una arquitectura de software modular que permite la reutilización y fácil sustitución de componentes en el software. El propósito principal es que este prototipo sirva como base para futuras etapas de desarrollo del proyecto.

Palabras Clave Videojuego, Aprendizaje musical, Arquitectura de software, Motivación, Innovación, Entretenimiento.

Abstract

The project describes the development and implementation of the prototype of an adventure and music video game that involves real instruments as a way to interact with the game and that allows the user to improve their musical skills. To achieve this, music teaching methodologies were used and a modular software architecture was designed that allows the reuse and easy replacement of components in the software. The main purpose is that this prototype serves as a basis for future stages of development of the project.

Keywords Video game, Musical learning, Software architecture, Motivation, Innovation, Entertainment.

1. Introducción

Este proyecto tiene como objetivo el diseño e implementación de un videojuego llamado Infinity Tower que combine la aventura de los juegos de rol con el aprendizaje musical. El enfoque principal del proyecto es crear el prototipo de un juego interactivo que motive y entretenga a los usuarios, al mismo tiempo que les brinde la oportunidad de adquirir habilidades musicales reales.

El proyecto se divide en varias etapas fundamentales. Inicia con la creación del documento de diseño del juego, donde se definen los elementos esenciales, como personajes, mecánicas de juego y desafíos musicales. Posteriormente, en una segunda fase, se elabora detalladamente el diseño arquitectónico de la aplicación, asegurando su modularidad, escalabilidad y seguridad. El desenlace de este proceso culmina en la creación de un prototipo funcional del juego.

El juego permite a los usuarios asumir el rol de pianista, cantante o guitarrista, y los sumergirlos en una aventura llena de desafíos musicales. A medida que los jugadores avanzan en la torre, podrán poner a prueba sus habilidades musicales, aprender nuevas técnicas y competir por la preciada colección musical del KeyMaster.

El juego esta dirigido tanto para el entretenimiento personal como para el aprendizaje musical, ofreciendo una alternativa atractiva e innovadora que pueda fomentar la práctica y el aprendizaje de instrumentos musicales reales.

2. Identificación del problema

En la actualidad, las personas dedican su tiempo libre a una amplia gama de actividades, como el deporte, ver series o películas, leer, socializar con la familia y amigos, bailar, escuchar música o jugar videojuegos. Si bien todas estas actividades tienen como objetivo entretenernos, es importante cuestionarnos si estamos invirtiendo nuestro tiempo en algo que aporte un valor real a nuestras vidas más allá del mero entretenimiento.

De acuerdo con un reporte publicado en enero de 2023 por la empresa de investigación de mercado (Newzoo, 2023), titulado "A New Era of Engagement in Media & Entertainment", se presentan interesantes hallazgos sobre las actividades de entretenimiento que las personas encuentran atractivas.

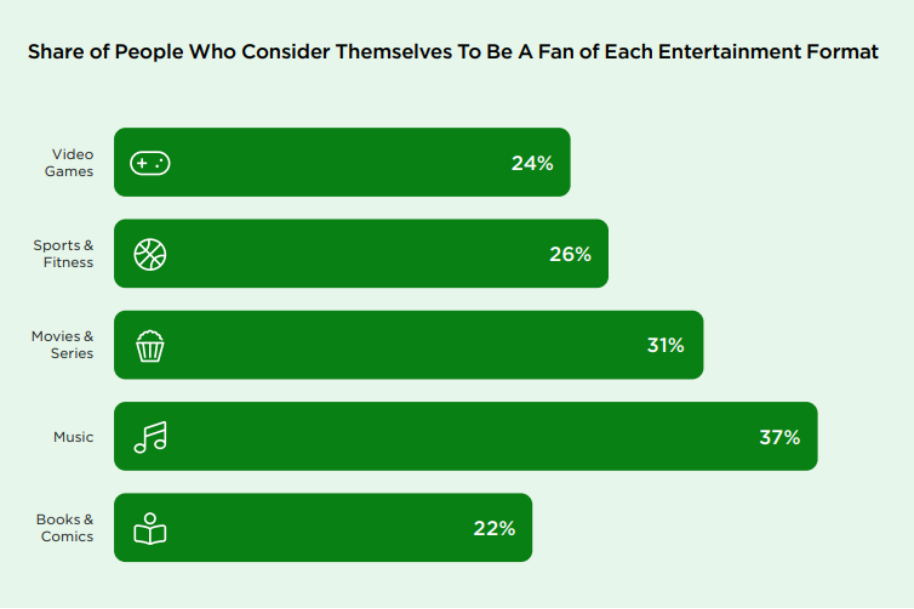


Figura 1: Interés de las personas en diferentes entretenimientos. (Newzoo, 2023)

En la Figura 1, se muestra un gráfico de barras que ilustra estos hallazgos. La música

encabeza la lista con un 37 %, seguida por las películas & series con un 31 %, los deportes con un 26 %, los videojuegos con un 24 %, y finalmente, la lectura de libros & cómics con un 22 %.

Es notable que la música es la actividad más interesante para las personas, superando incluso el interés por ver series y películas, o jugar videojuegos. Esto nos sugiere que existe un potencial considerable en cuanto al interés de las personas por aprender a tocar un instrumento musical. En este sentido, el mismo reporte de (Newzoo, 2023) también revela un cambio en la gráfica cuando se indaga acerca de las actividades que las personas practican activamente.

En la Figura 2 podemos observar que cuando se trata de actividades de entretenimiento que se practican activamente los videojuegos lideran con un 72 %, seguidos por la lectura de libros & cómics con un 62 %, los deportes con un 60 %, la música con un 37 %, y por ultimo, las películas & series con un 30 %.

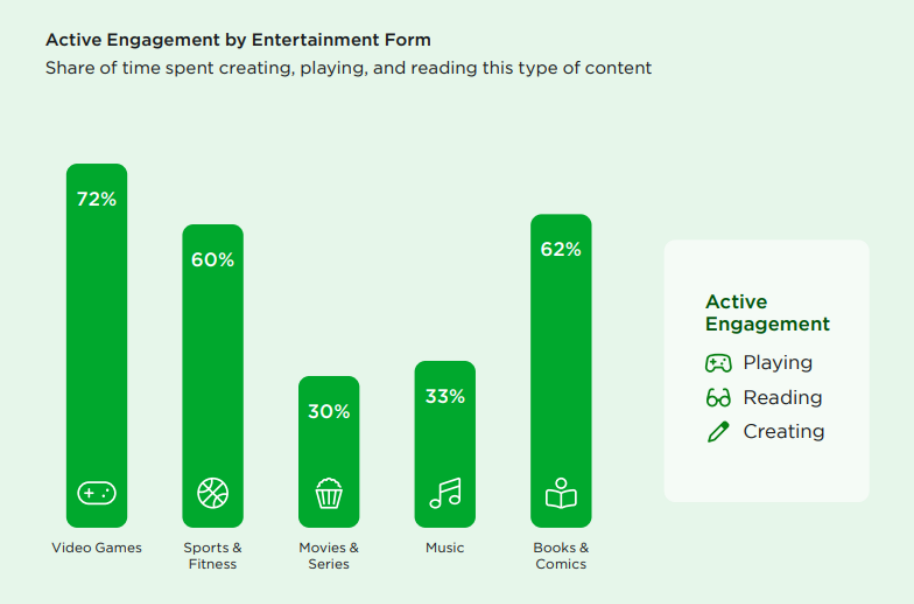


Figura 2: Entretenimientos practicados de manera activa. (Newzoo, 2023)

Si comparamos los resultados anteriores, que indicaban el interés de las personas, con estos resultados que reflejan sus actividades reales, podemos observar que la música genera interés, pero su práctica real no es tan frecuente o no es la primera opción para la mayoría de las personas.

En la actualidad, existe una amplia oferta de videojuegos disponibles en el mercado y teniendo en cuenta que cerca del 72 % de las personas aceptan que los videojuegos hacen parte de los entretenimientos que practican de manera activa, como se observa en la Figura 2, es innegable que gran parte de estos capturan nuestra atención y nos brindan horas de entretenimiento. Sin embargo, a pesar de su popularidad, carecen de un valor añadido más allá de la diversión. Este tiempo que dedicamos a los videojuegos podría ser utilizado para adquirir habilidades valiosas, como aprender a tocar un instrumento musical o adquirir conocimientos en una determinada área. Desafortunadamente, muchas personas no encuentran estas actividades atractivas o entretenidas.

2.1. Causas del problema

Una de las principales causas de esta falta de motivación hacia el aprendizaje musical es la percepción que tienen las personas de que es una actividad difícil y aburrida. En contraste, los videojuegos ofrecen historias emocionantes y desafiantes que resultan más atractivas para la mayoría de las personas.

Además, es importante tener en cuenta que en la sociedad actual, las personas tienden a invertir más tiempo en actividades del mundo digital que en actividades del mundo real, como se muestra en la Figura 3, extraída del reporte publicado por (Newzoo, 2023). En este análisis comparativo del tiempo diario invertido por una persona en actividades del mundo real frente actividades del mundo digital, se observa que en promedio, las personas invierten 6.6 horas en actividades del mundo digital y 2.9

horas en actividades del mundo real.

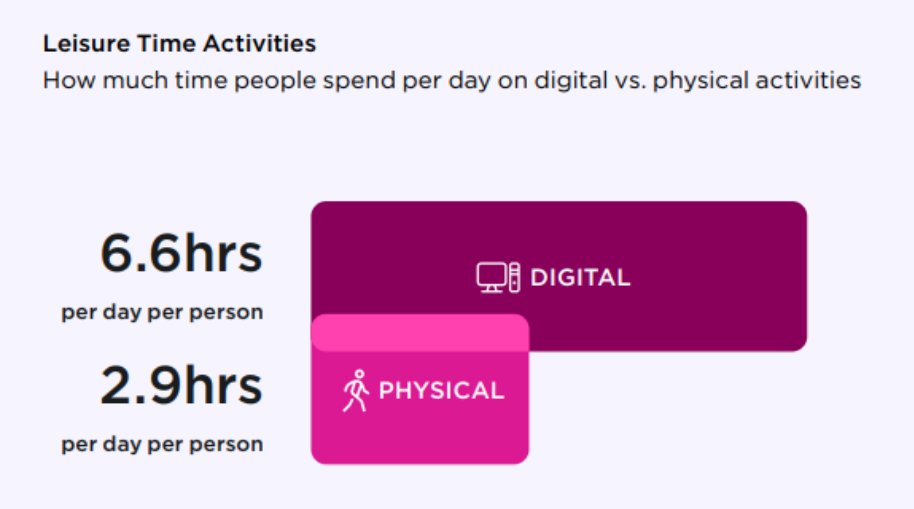


Figura 3: Mundo digital vs Mundo real. (Newzoo, 2023)

2.2. Efectos negativos del problema

En la situación actual, se observa una carencia de valor agregado en la producción y desarrollo de nuevos videojuegos. A pesar de la creación de historias cautivadoras y juegos altamente competitivos, el tiempo invertido en perfeccionar habilidades dentro de estos juegos no trasciende más allá del contexto del juego mismo.

El principal efecto negativo de esta situación es que se desaprovecha el tiempo y se limita el desarrollo personal y el enriquecimiento cultural de las personas. Aunque los videojuegos ofrecen entretenimiento y desafío, es necesario explorar nuevas oportunidades para que la experiencia de juego ofrezca beneficios adicionales, como el desarrollo de habilidades prácticas o el aprendizaje de conceptos significativos.

2.3. Formulación del problema

El creciente interés en los videojuegos representa una oportunidad significativa para el desarrollo de plataformas educativas interactivas. Según (Newzoo, 2023), la industria de los videojuegos sigue expandiéndose, capturando la atención de una audiencia global diversa. Esta tendencia destaca el potencial de los videojuegos no solo como entretenimiento, sino también como herramientas efectivas para el aprendizaje y el desarrollo de habilidades. En este contexto, surge la pregunta: ¿Como desarrollar un videojuego que incorpore elementos de aventura, pero que al mismo tiempo brinde la oportunidad al usuario de desarrollar habilidades valiosas en el mundo real, como por ejemplo la interpretación de música? Este concepto resulta sumamente interesante, ya que podría fusionar lo mejor de ambos mundos y ser algo realmente innovador y beneficioso para las personas.

3. Objetivos del proyecto

3.1. Objetivo General

Diseñar e implementar el prototipo de un videojuego de aventuras y música que involucre instrumentos reales como forma de interactuar con el juego y que permita al usuario mejorar sus habilidades musicales, centrándose en el desarrollo de la precisión rítmica y la velocidad de interpretación.

3.2. Objetivos específicos

- Crear el documento de diseño del juego lo cual incluye, modos de juego, niveles, personajes, enemigos, música y otros elementos.
- Diseñar una interfaz intuitiva y fácil de usar que permita al usuario interactuar con los instrumentos reales en el juego.
- Diseñar una arquitectura de software que sea modular, escalable y segura que permita la incorporación y actualización de nuevos módulos de manera independiente.
- Implementar el prototipo del videojuego de aventuras y música.
- Realizar pruebas con usuarios para evaluar la experiencia de juego y la efectividad del aprendizaje musical, con especial enfoque en la mejora de la precisión rítmica y la velocidad de interpretación musical.

3.3. Resultados obtenidos

Los resultados obtenidos del proyecto incluyen un prototipo de juego funcional que combina la emoción y la aventura de los juegos de rol con el aprendizaje musical. Este prototipo ha sido diseñado para captar la atención y motivación de los usuarios, desde estudiantes de música hasta aquellos sin experiencia previa en el ámbito musical. Se ha observado que el juego ha logrado incentivar el aprendizaje musical de manera entretenida, ofreciendo una alternativa atractiva incluso para aquellos con falta de motivación en el aprendizaje de instrumentos tradicionales.

Además, el prototipo del juego tiene el potencial de servir como base para el desarrollo de futuros juegos en diversos contextos, ya sea para el entretenimiento personal o para su aplicación en entornos educativos.

4. Alcance

Es importante destacar que el alcance del proyecto se limitó al diseño e implementación del prototipo del videojuego, poniendo un especial énfasis en la construcción de una arquitectura de software modular. Esto permitirá la incorporación y actualización de nuevos módulos de manera independiente, dejando espacio para futuras etapas de desarrollo y comercialización en caso de que el prototipo resulte exitoso y se decida su continuación.

A continuación se plantean algunas preguntas para definir el alcance del prototipo del videojuego en los principales aspectos:

- **¿Cuál es la idea del juego?** La idea es crear un videojuego que combine el aprendizaje musical y las aventuras de los juegos de rol.
- **¿Qué habilidades musicales se busca desarrollar con la ayuda del juego?** Se busca mejorar la precisión rítmica y la velocidad de interpretación musical.
- **¿De qué género será el juego?** El juego será de rol RPG competitivo.
- **¿Será en 2D o 3D?** En principio, la idea es que sea en 2D o 2.5D.
- **¿Cómo será la jugabilidad?** La jugabilidad se centrará en desafíos musicales basados en el ritmo.
- **¿Cuáles son los dispositivos objetivo?** Para este prototipo inicial, solo se considerará PC.
- **¿Qué modos tendrá el juego?** El juego contará con los modos de práctica y un modo de juego en solitario.

- **¿Cuáles son las reglas físicas que rigen el universo del juego?** Al tratarse de un juego de desafíos musicales, no se necesitarán reglas físicas.
- **¿Quién juegue podrá tener efectos que cambien la mecánica?** Los jugadores podrán experimentar efectos de disminución y aumento de velocidad.
- **Cantidad de pisos de la torre:** El juego contará con un mínimo de 1 piso y un máximo de 3 pisos.
- **Cantidad de pueblos por piso:** En cada piso habrá un mínimo de 5 pueblos y un máximo de 10 pueblos.
- **Cantidad de desafíos por pueblo:** Cada pueblo presentará un mínimo de 1 desafío y un máximo de 5 desafíos.

5. Justificación del trabajo de grado

La solución propuesta en este proyecto tiene varios impactos y beneficios positivos para el problema de falta de motivación en el aprendizaje de instrumentos musicales. A continuación, se detallan algunos de los impactos y beneficios esperados:

- Impacto en el aprendizaje musical: El videojuego tiene como objetivo brindar una experiencia de aprendizaje musical interesante e inmersiva, permitiendo a los usuarios adquirir o mejorar sus habilidades musicales de una manera divertida y atractiva. Al combinar la música con elementos de aventura y exploración, se generará un ambiente motivador que fomentará la práctica constante y el desarrollo de habilidades musicales.

- Impacto en la industria de los videojuegos: La creación de un videojuego que integre instrumentos reales como forma de interactuar con el juego tiene el potencial de generar un impacto significativo en la industria de los videojuegos. Esta propuesta puede abrir nuevas oportunidades y atraer a un público más amplio, proporcionando una experiencia de juego diferenciada y enriquecedora. También se busca impulsar en esta industria la creación de otros videojuegos enfocados en el desarrollo de habilidades que tengan un valor en el mundo real.

- Beneficios educativos: La incorporación de instrumentos reales en el juego puede tener un impacto positivo en la educación musical. El videojuego puede ser utilizado como una herramienta complementaria en escuelas, conservatorios y centros de música, brindando a los estudiantes una forma divertida de practicar y desarrollar sus habilidades musicales.

- Impacto social: El videojuego puede promover la interacción social entre los usuarios, ya sea a través de la competencia amistosa, la colaboración en la exploración del juego o la creación de comunidades en línea. Además, al proporcionar una herramienta de aprendizaje musical accesible y atractiva, se pue-

de fomentar la participación de personas de diferentes edades y antecedentes culturales en la música y la cultura.

- Beneficios económicos: Añadir publicidad dentro del videojuego puede generar ingresos económicos constantes según el número de usuarios que usen el producto. Además, la creación de empleo en el campo del desarrollo de videojuegos y la promoción de la industria tecnológica y cultural pueden contribuir al crecimiento económico a nivel local y nacional.

6. Marco teórico de referencia y antecedentes

6.1. Bases Teóricas

6.1.1. Introducción a los videojuegos

6.1.1.1. ¿Qué es un videojuego?

Definir qué es un videojuego podría parecer algo sencillo porque es algo que todos conocemos, pero para poder tener claro este concepto debemos empezar desde su definición más simple hasta su más compleja.

En el libro llamado "The Art of a game design" (Schell, 2008), nos dicen que un juego es algo que jugamos, pero esto no nos dice mucho. Por ejemplo, ese algo podría ser un juguete, pero entonces ¿Qué diferencia hay entre un juego y un juguete?

Dado que el concepto de qué es un juego puede resultar más complejo, (Schell, 2008) propone que comencemos definiendo varios conceptos que nos ayudarán a comprender qué es un juego. A continuación, se presentan las siguientes definiciones.

- La diversión es el placer con sorpresas.
- Jugar es la manipulación que satisface la curiosidad.
- Un juguete es un objeto con el que se juega.
- Un buen juguete es un objeto con el que es divertido jugar.
- Un juego es una actividad de resolución de problemas, abordada con una actitud lúdica.

Además, (Schell, 2008), recopila una lista de cualidades que un juego debería tener, basada en diversas definiciones.

- A los juegos se ingresa de manera voluntaria.
- Los juegos tienen objetivos.
- Los juegos tienen conflicto.
- Los juegos tienen reglas.
- En los juegos se puede ganar y perder.
- Los juegos son interactivos.
- Los juegos tienen desafío.
- Los juegos pueden crear su propio valor interno.
- Los juegos involucran a los jugadores.
- Los juegos son sistemas cerrados y formales.

Adicionalmente, debemos tener en cuenta otros componentes que rodean a un videojuego, como lo son la parte tecnológica, visual y auditiva. Teniendo en cuenta las definiciones y cualidades propuestas por (Schell, 2008) y los componentes adicionales que rodean a un videojuego, podemos definir a un videojuego de la siguiente manera.

"Un videojuego es una aplicación orientada al entretenimiento interactivo que combina elementos visuales, auditivos y jugables para crear una experiencia virtual. Este tiene como objetivo brindar actividades de resolución de problemas, en la cual los jugadores ingresan de manera voluntaria para satisfacer su curiosidad y buscar placer a través de la manipulación de un entorno virtual".

6.1.1.2. Componentes de un videojuego

Al construir un videojuego, es esencial comprender los componentes fundamentales necesarios para su desarrollo. Según (Schell, 2008), existen cuatro elementos clave que actúan como los pilares fundamentales en la construcción de un juego: las mecánicas, la historia, la estética y la tecnología. En la Figura 4 se pueden apreciar dichos pilares y como se relacionan entre ellos.

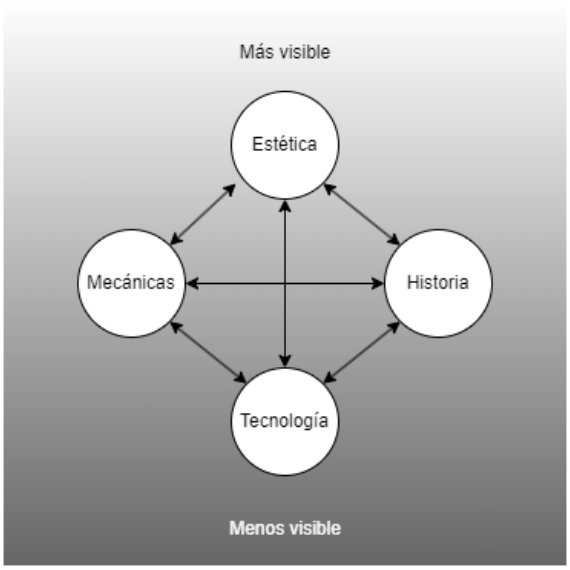


Figura 4: Pilares fundamentales y su relación entre ellos. Fuente propia

(Schell, 2008) establece los cuatro pilares fundamentales de la siguiente manera.

- **Mecánicas:** Son todos los procedimientos y reglas del juego. Describe cuál o cuáles son los objetivos del juego, cómo los jugadores pueden y no pueden lograrlos, y lo que sucede cuando lo intentan.
- **Historia:** Es la secuencia de eventos que se desarrolla en el juego. Esta puede ser lineal y preestablecida, o puede ser ramificada y emergente.

- **Estética:** Así es como se ve, suena, huele, sabe y se siente un juego. La estética es un aspecto realmente importante en el diseño de un juego, ya que tiene la relación mas directa con la experiencia del jugador.
- **Tecnología:** La tecnología no solamente se refiere a un software o aplicación exclusivamente, es cualquier material o interacción que haga posible el juego.

Algo importante que destaca (Schell, 2008), es que ninguno de los pilares es más importante que los demás, ya que todos son relevantes al construir o diseñar un juego. Sin embargo, es importante considerar que, desde la perspectiva de los usuarios, la tecnología suele ser lo menos visible, la estética es lo más visible, y la historia y mecánicas se sitúan en un punto intermedio. Además, es crucial tener en cuenta que todos estos pilares están estrechamente relacionados entre sí.

6.1.2. Teoría de aprendizaje musical

A lo largo de la historia, se han explorado diversos enfoques y métodos para la enseñanza musical. Dado que el objetivo de nuestra aplicación se centra en el aprendizaje musical a través de un videojuego, es esencial investigar y evaluar cuáles de estos métodos serían los más apropiados para incorporar en nuestra aplicación, teniendo en cuenta su enfoque y propósito específico. En un estudio realizado por (Fontelles Rodríguez, 2019) sobre metodologías de enseñanza musical, se presenta una amplia gama de enfoques y prácticas utilizadas en este campo.

Considerando la diversidad de métodos de enseñanza en la recopilación de (Fontelles Rodríguez, 2019), he optado por centrarme exclusivamente en aquellos que sean propicios para el aprendizaje musical basado en el ritmo, que puedan ser aplicables a cualquier edad y promover un enfoque individualizado y repetitivo en el proceso de enseñanza.

A continuación, se proporciona una breve descripción de estas metodologías; sin embargo, en el Anexo 10, se encuentran las tablas comparativas que recopilan las características clave de estos métodos, detalladas a continuación.

6.1.2.1. Método Dalcroze

Jacques Dalcroze siendo profesor de solfeo en Ginebra, identificó deficiencias en la educación musical tradicional, lo que lo llevó a desarrollar su propio método. Su enfoque se centró en el estudio del ritmo en sus aspectos visuales y sonoros, utilizando ejercicios rítmicos originales para complementar la formación de sus alumnos en solfeo y composición, al mismo tiempo que desarrollaban su habilidad auditiva.

Dalcroze fundamentó su método en la observación y la experiencia directa de sus alumnos en relación a los elementos musicales, enfocándose especialmente en el ritmo. Consideraba que la sensación de movimiento era fundamental para respaldar el aprendizaje intelectual. Para fomentar la expresión dinámica y el fraseo musical, se inspiró en las obras de su amigo Mathiz Lussy. A pesar de su propuesta de incluir su método en los programas oficiales de conservatorios, esta idea fue rechazada.

Según explica (Fontelles Rodríguez, 2019), gran parte de estos objetivos están enfocados a resolver los problemas de arritmia que pueden tener los alumnos.

He optado por incluir este método en el proyecto, ya que se enfoca en el ritmo y la expresión corporal. Este enfoque sería especialmente útil para incorporar desafíos musicales que requieran movimientos corporales en sincronía con el ritmo de la música.

6.1.2.2. Método Kodály

El Método Kodály es un enfoque de enseñanza de la música que se basa en el uso de la voz y el canto para desarrollar la habilidad musical en los estudiantes. Fue desarrollado por el músico, compositor y pedagogo musical húngaro Zoltan Kodály (1882-1967), el cual estudió música en la Academia Franz Liszt, además de obtener un título en lenguas y un Doctorado en Filosofía en lingüística (Lierse, 2010).

Según menciona (Lierse, 2010), este método es un enfoque centrado en la creencia de que la música es un lenguaje, y que como tal debe ser aprendida de la misma manera que se aprende un idioma, a través de la escucha, el canto y la imitación.

(Fontelles Rodríguez, 2019) afirma que la finalidad de este método es formar el oído antes que nada y que debemos centrarnos en desarrollar los siguientes elementos: canto, audición, lectura y escritura.

La Fonomímia de Kodaly reconoce y ubica los sonidos empleando los signos de la siguiente manera:

Nota Latín	Nota Kodaly	Descripción Fonomímia
DO	DO	Puño cerrado en horizontal.
RE	RE	Mano alzada en diagonal.
MI	MI	Mano en horizontal (mediana)
FA	FA	Índice hacia abajo (resto de dedos recogidos).
SOL	SO	Mano vertical con el pulgar hacia arriba (dorso hacia fuera).
LA	LA	Mano relajada hacia abajo.
SI	TI	Índice hacia arriba (resto de dedos recogidos)
DO	DO'	Puño cerrado y brazo hacia arriba (para diferenciarlo del Do hacia abajo).

Tabla 1: Fonomímia de Kodály. Fuente (Fontelles Rodríguez, 2019)

Este enfoque, que se centra en el desarrollo auditivo y vocal, resulta fundamental para incorporar desafíos que involucren la reproducción vocal de ritmos y melodías.

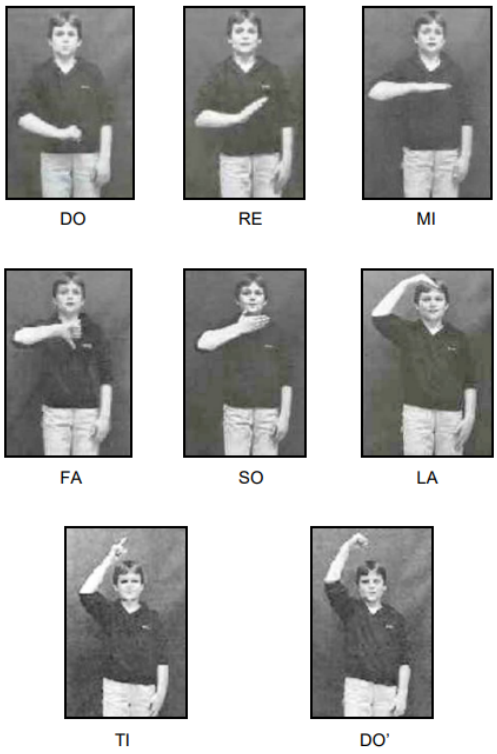


Figura 5: Fonomímia de Kodály. Fuente (Fontelles Rodríguez, 2019)

6.1.2.3. Método Orff

Carl Orff fue un destacado músico y pedagogo alemán. En el año 1924, en la ciudad de Múnich, fundó una escuela que abarcaba la danza, la música y la gimnasia, junto a la bailarina Dorothee Günther. Fue en este contexto donde Orff desarrolló su revolucionario método, el cual generó un cambio significativo en las prácticas educativas musicales de la época, rompiendo con las tradiciones arraigadas hasta entonces.

(Fontelles Rodríguez, 2019) destaca cómo Carl Orff fundamenta su metodología en la estrecha relación entre el ritmo y el lenguaje, buscando que los estudiantes experimenten la música antes de aprenderla. A través de enfoques vocales, instrumentales, verbales y corporales.

Algunos de los principios del método de Orff consisten en:

- Partir del ritmo en la palabra y combinar sus acentos.
- Aprovechar los aprendizajes musicales adquiridos hasta entonces y relacionarlos con otras áreas favoreciendo, de esta manera, la creatividad.
- El desarrollo de la improvisación y la adaptación a otras culturas utilizando el lenguaje universal que la música posee.
- Canalizar el movimiento mediante todas las actividades del método Orff-Schulwerk puesto que el aprendizaje del lenguaje musical usará de las vivencias de todos los elementos mediante el cuerpo.

Este enfoque, centrado en la exploración musical a través de la improvisación, resulta especialmente útil si se desea incorporar desafíos en los cuales no exista una pista predefinida. En su lugar, los usuarios podrán elegir una temática y, basándose en ella, ejecutar ejercicios musicales. El desafío consistirá en evaluar si dichos ejercicios se ajustan a la temática elegida, fomentando así la creatividad y la capacidad de improvisación de los jugadores.

6.1.2.4. Método Suzuki

Shinichi Suzuki fue un violinista, educador y filosofo japones. Enfoco su teoría sobre el hecho de que la habilidad musical no es un talento innato, si no una destreza que puede ser desarrollada, de la misma manera que todos desarrollamos la capacidad de hablar nuestra propia lengua materna.

La siguiente es una analogía del método Suzuki con el aprendizaje del habla expuesta por (Fontelles Rodríguez, 2019):

- Método de lengua materna. El recién nacido solo escucha los sonidos.
- Comienza a producir los primeros sonidos. Comienza a imitar.
- Perfecciona su imitación.
- Su familia lo corrige repitiendo las palabras.
- Primero forma palabras, después oraciones.
- Cuando posee un amplio vocabulario se inicia en la lecto-escritura.

Este método, basado en el aprendizaje auditivo y la repetición, es fundamental en la concepción principal de Infinity Tower. Dado que los desafíos se centran en el ritmo, es probable que los jugadores no puedan superarlos de inmediato, sino que necesiten practicarlos una y otra vez hasta lograr superarlos. La constante repetición de estos desafíos permitirá a los jugadores mejorar su precisión rítmica y desarrollar un oído musical afinado.

6.1.2.5. Método Gordon

Edwin E. Gordon fue un contrabajista y pedagogo, el cual baso su principio en que la música se puede aprender mediante procesos análogos al aprendizaje del habla. La base teórica de los estudios de Gordon esta construida sobre las aptitudes musicales y la capacidad auditiva según nos muestra (Fontelles Rodríguez, 2019) al explicar los principales aspectos de esta metodología.

Esta metodología, enfocada en el desarrollo auditivo y los patrones rítmicos, resulta invaluable para incorporar patrones en las prácticas previas a los desafíos. Al incluir estos patrones, se brindará a los jugadores la oportunidad de desarrollar una

comprensión más profunda del ritmo y mejorar su habilidad de lectura rítmica. Estas prácticas les permitirán familiarizarse con los diferentes patrones y adquirir la destreza necesaria para enfrentar los desafíos rítmicos con mayor confianza.

6.1.2.6. Enfoque

Al considerar los diferentes métodos de enseñanza musical y su aplicación potencial en el juego, es importante evaluar cuáles podrían ser más beneficiosos para nuestro enfoque específico y por qué.

El Método Dalcroze, con su énfasis en el ritmo y la expresión corporal, es altamente relevante para nuestro juego, ya que la precisión del ritmo es fundamental en la mecánica del juego. La integración de desafíos que requieran movimientos corporales en sincronía con la música podría mejorar la experiencia de aprendizaje y la conexión entre el jugador y la música.

Por otro lado, el Método Kodály, centrado en el desarrollo auditivo y vocal, también es valioso, ya que fortalecería la capacidad de los jugadores para reconocer y reproducir patrones auditivos y ritmos. Esta habilidad es esencial para mejorar la precisión en la interpretación musical, que es uno de los objetivos principales de nuestro juego.

El Método Orff, con su enfoque en la improvisación y la exploración musical, es menos relevante para nuestro juego, ya que nuestros desafíos están más estructurados y requieren una precisión específica en la ejecución. Aunque la creatividad es importante, la prioridad es la precisión y la repetición en la práctica.

El Método Suzuki, que se centra en la repetición y el aprendizaje auditivo, es altamente beneficioso para nuestro juego, ya que la repetición constante de desafíos musicales es una parte integral de la experiencia de juego. La capacidad de los jugadores para mejorar su precisión musical a través de la práctica repetida se alinea

perfectamente con nuestro enfoque de aprendizaje.

Finalmente, el Método Gordon, que se enfoca en la comprensión de patrones rítmicos, también es relevante para nuestro juego. La integración de patrones rítmicos en las prácticas previas a los desafíos ayudaría a los jugadores a familiarizarse con diferentes ritmos y mejorar su habilidad para interpretarlos con precisión durante el juego.

6.1.3. Arquitectura de software en videojuegos

6.1.3.1. Características de un buen diseño

Es ampliamente conocido que en muchas profesiones no es necesario reinventar la rueda, y contar con una buena receta que alguien ha desarrollado en el pasado es la base para realizar un trabajo exitoso. Esto aplica tanto en la cocina, las finanzas, la arquitectura, y por supuesto, en el desarrollo de software.

En el ámbito del desarrollo de software, estas recetas se conocen como patrones de diseño. Estos patrones fueron creados en el pasado para resolver problemas comunes que los desarrolladores enfrentaban. Y esto nos lleva a la primer Característica fundamental en el diseño y desarrollo de software: la **reutilización de código**.

La **reutilización de código** tiene como objetivo evitar desarrollar funcionalidades desde cero, y utilizar patrones de diseño nos permite tener componentes más flexibles que, en muchas situaciones, son la solución ideal para resolver problemas de manera rápida y sin incurrir en costos innecesarios.

La segunda Característica fundamental es la **extensibilidad**. Esta Característica está estrechamente relacionada con un hecho inevitable en la vida de un programador: el cambio. Las tecnologías evolucionan constantemente, surgen nuevas versiones,

algunas se vuelven obsoletas, o simplemente se desea cambiar un framework o tecnología por diversas razones. Por esta razón, es de vital importancia diseñar una arquitectura que facilite la **extensibilidad** desde el principio en nuestras aplicaciones. Aquí es donde los patrones de diseño vuelven a ser de gran utilidad.

6.1.3.2. Principios de SOLID

Antes de centrarnos en los patrones de diseño, es importante que tengamos en cuenta unos principios que influyen en el desarrollo y funcionamiento de dichos patrones. En un libro publicado por Unity llamado "Level up your code with game programming patterns" (Unity, 2021a), nos hablan de los principios de SOLID. Este es un acrónimo mnemotécnico de cinco fundamentos básicos que se tienen en el diseño de software.

- **S**ingle responsibility principle (Principio de responsabilidad única) .
- **O**pen-closed principle (Principio de abierto/cerrado).
- **L**iskov substitution principle (Principio de sustitución de Liskov).
- **I**nterface segregation principle (Principio de segregación de la interfaz).
- **D**ependency inversion principle (Principio de inversión de la dependencia).

6.1.3.2.1. Principio de responsabilidad única

En resumen, el principio de responsabilidad única establece que una clase debe tener una única responsabilidad. Este principio se refuerza en (Unity, 2021a), donde se indica que cada módulo, clase o función debe ser responsable de una sola cosa y encapsular únicamente esa parte de la lógica.

La importancia de seguir este principio radica en construir pequeñas funcionalidades en lugar de grandes funcionalidades con múltiples tareas combinadas. Esto permite que las clases y métodos sean más concisos, fáciles de entender, explicar e implementar.

Un ejemplo ilustrativo de lo que se debe evitar se muestra en la Figura 6. En dicha figura, se presenta una clase llamada **Player** que asume múltiples responsabilidades, como el manejo de audio, el movimiento y los controles del juego. Esta situación viola el principio de responsabilidad única y puede llevar a un código complejo y difícil de mantener.

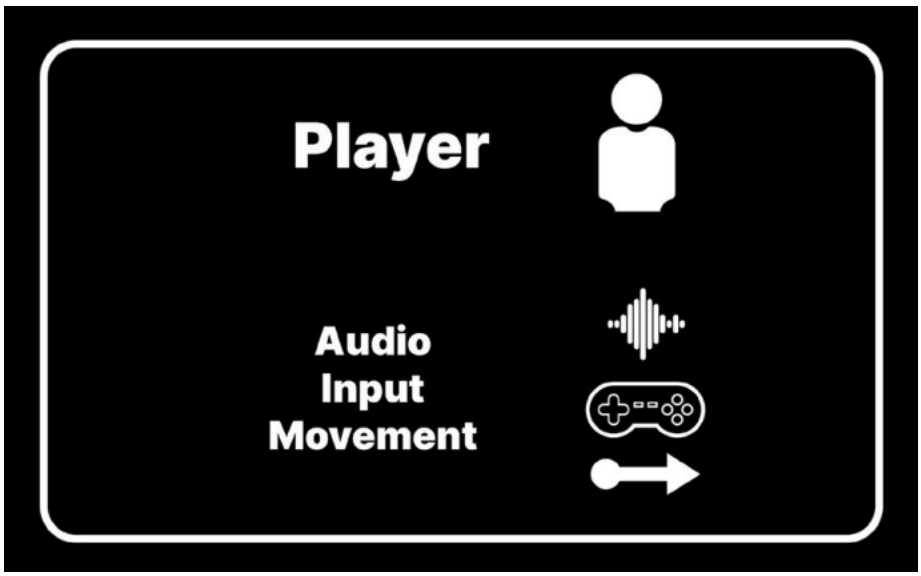


Figura 6: Clase Player con múltiples responsabilidades. (Unity, 2021a)

Por otro lado, en la Figura 7 se puede apreciar cómo las responsabilidades se han dividido en diferentes componentes, cada uno cumpliendo una única función. Esto se alinea con el principio de responsabilidad única y da como resultado una estructura más limpia y un entendimiento más claro de las responsabilidades de cada componente.

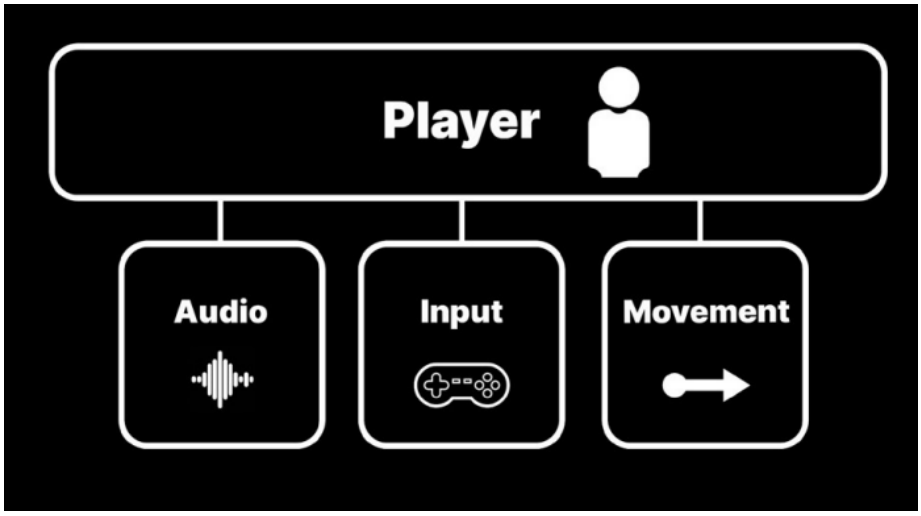


Figura 7: Diseño refactorizado en diferentes clases con responsabilidades únicas. (Unity, 2021a)

6.1.3.2.2. Principio de abierto/cerrado

Este principio nos indica que las clases deben ser abiertas para su extensión pero cerradas para su modificación. Es decir, debemos diseñar nuestras clases de tal manera que puedan incorporar nuevos comportamientos sin necesidad de modificar la clase original.

Un ejemplo sencillo de este principio se puede observar en una clase que calcula el área de diferentes formas. En la Figura 8, tenemos una clase llamada **AreaCalculator** que tiene un método para cada forma en la cual se desea calcular el área.

A primera vista, el código puede parecer correcto, pero si en el futuro se desea agregar más formas para calcular el área, esta clase podría volverse demasiado compleja y difícil de entender.

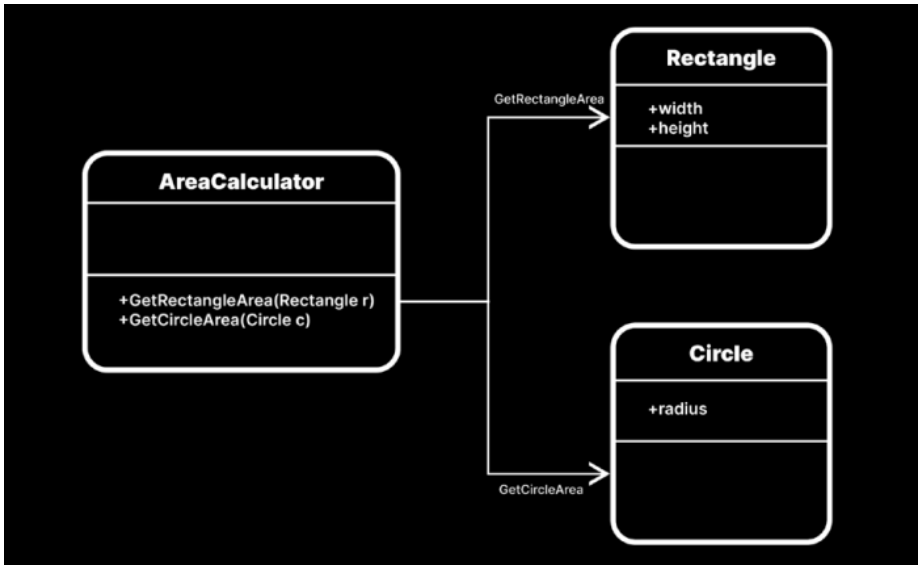


Figura 8: Clase AreaCalculator con diferentes métodos para calcular el área. (Unity, 2021a)

Ahora, en lugar de tener la lógica de cálculo de cada forma dentro de la clase **AreaCalculator**, podemos mejorar el diseño creando una clase abstracta llamada **Shape**. En esta clase abstracta, definimos un método **CalculateArea()** y luego hacemos que cada clase de formas, como **Circle** o **Rectangle**, hereden de esta clase abstracta.

De esta manera, logramos que la clase **AreaCalculator** pueda obtener el área de cualquier forma que implemente la clase abstracta **Shape** de forma adecuada. Este enfoque se puede visualizar en la Figura 9.

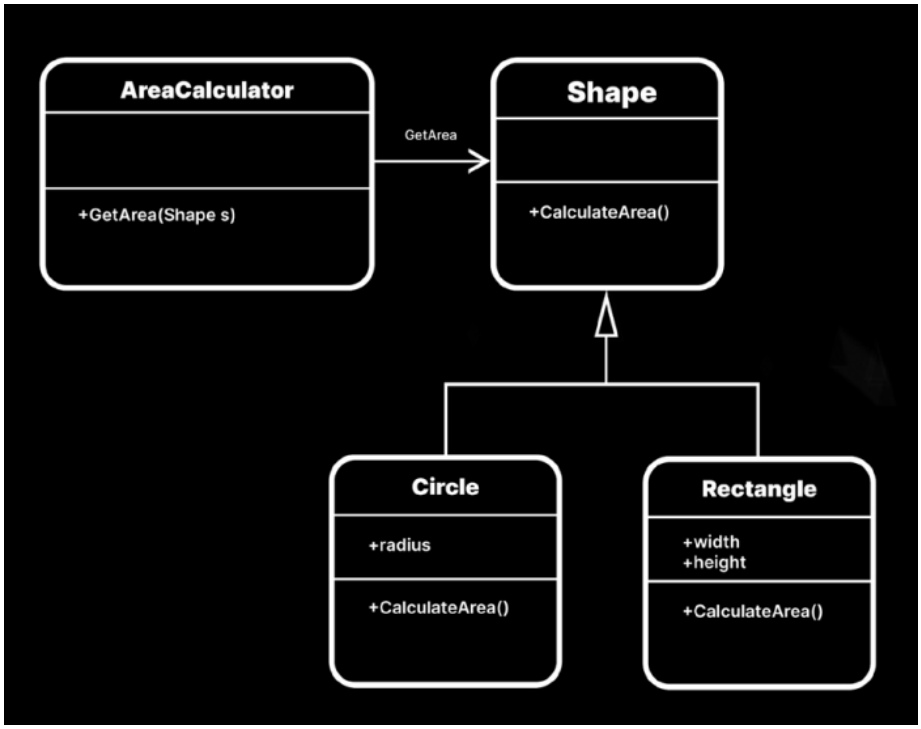


Figura 9: Diseño refactorizado de la clases usando el principio abierto/cerrado. (Unity, 2021a)

6.1.3.2.3. Principio de sustitución de Liskov

Este principio establece que las clases derivadas deben ser sustituibles por sus respectivas clases base. En otras palabras, debemos asegurarnos de que las subclases puedan ser utilizadas de manera transparente en lugar de la clase base, sin generar comportamientos inesperados.

Tomemos como ejemplo una clase principal llamada **Vehicle**, que servirá como clase base para diferentes subclases representando distintos tipos de vehículos. En este caso, consideremos las subclases **Car** y **Train**, como se muestra en la Figura 10.

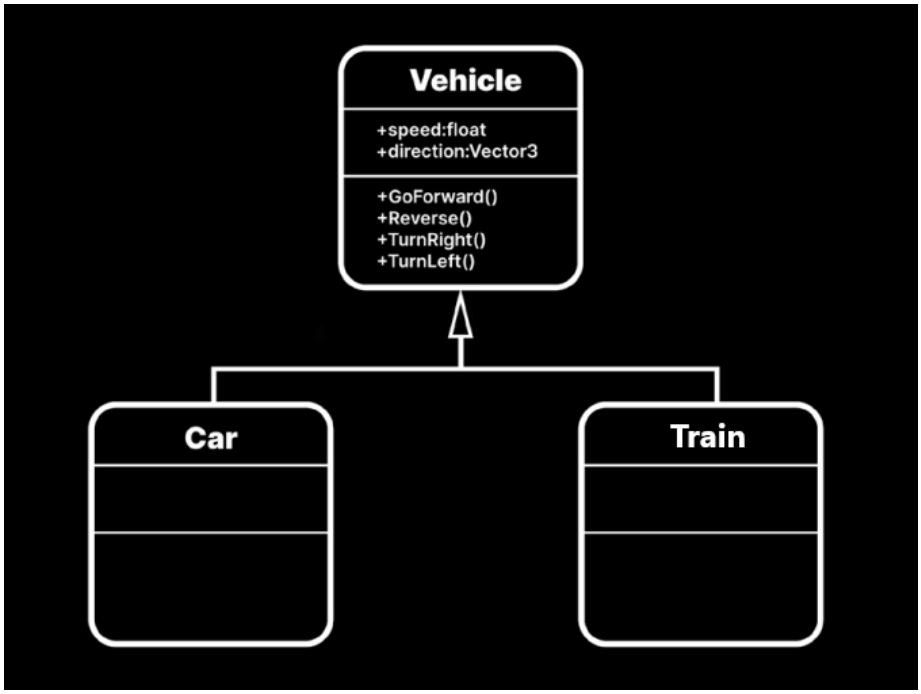


Figura 10: Estructura de clases y subclases. (Unity, 2021a)

Supongamos que los tipos de vehículos que deseamos implementar tienen comportamientos distintos. Por ejemplo, los carros pueden moverse hacia adelante, atrás, derecha e izquierda, mientras que los trenes solo pueden moverse hacia adelante o atrás, como se muestra en la Figura 11.

En este escenario, podríamos caer en una mala implementación de métodos en la clase base **Vehicle**, como se ilustra en la Figura 12.

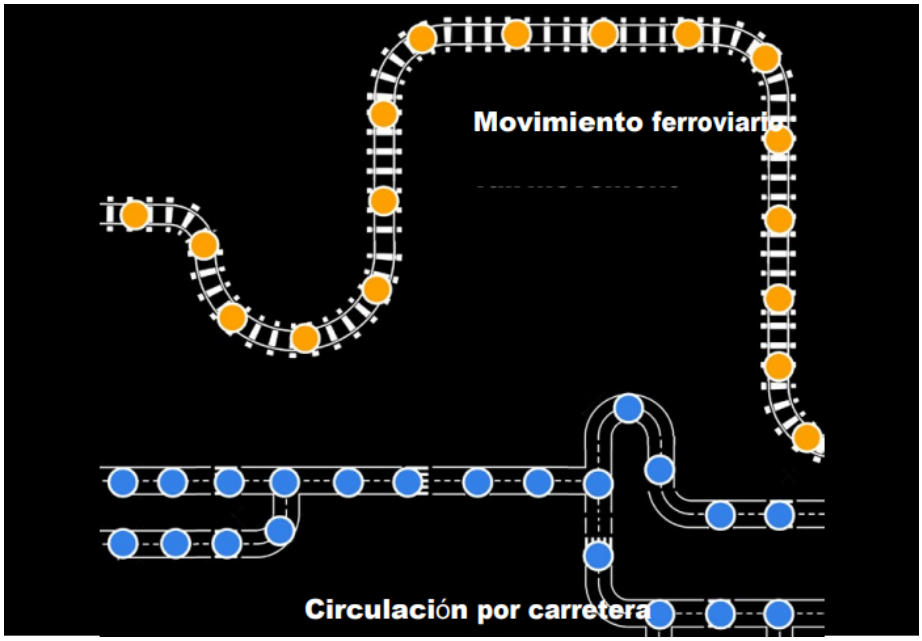


Figura 11: Comportamiento de los tipos de vehículos. (Unity, 2021a)

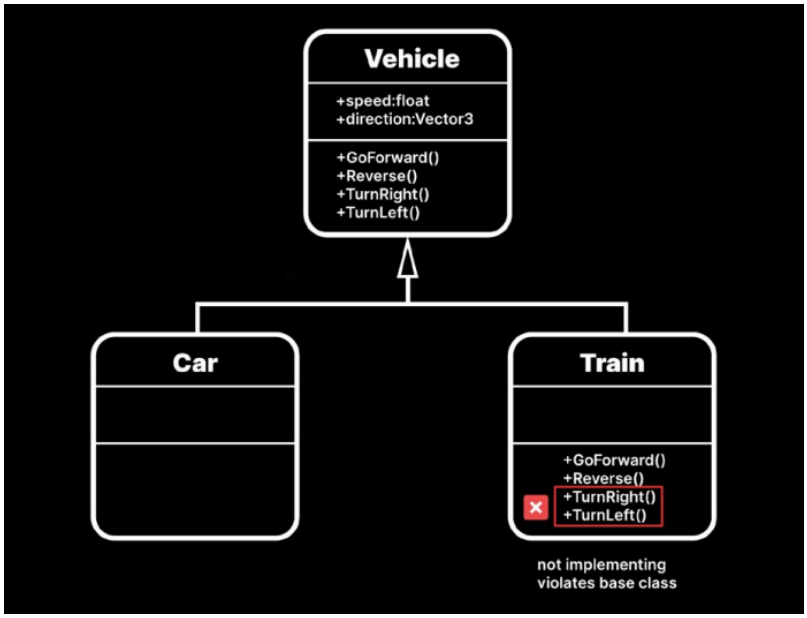


Figura 12: Violación del principio de sustitución de Liskov. (Unity, 2021a)

En esta implementación, se definen los métodos de movimiento (GoForward, Reverse, TurnLeft, TurnRight) en la clase base **Vehicle**, y se asume que todas las subclases heredarán estos métodos. Sin embargo, esta implementación viola el principio de sustitución de Liskov (LSP) porque los trenes no pueden realizar movimientos laterales (izquierda y derecha), generando un comportamiento inconsistente.

Para solucionar esta mala implementación, debemos eliminar los comportamientos inapropiados de la clase base y trasladarlos a interfaces. En este caso, podríamos tener una interfaz que defina el comportamiento de moverse hacia adelante y hacia atrás, y otra interfaz que defina el comportamiento de moverse hacia la derecha o hacia la izquierda.

De este modo ahora podriamos tener dos clases separadas **RoadVehicle** y **RailVehicle** que pueden implementar de dichas interfaces los comportamientos que les correspondan, como se puede apreciar en la Figura 13

De esta manera, podemos tener dos clases separadas: **RoadVehicle** y **RailVehicle**, las cuales implementarán las interfaces correspondientes y proporcionarán los comportamientos apropiados para cada tipo de vehículo, como se muestra en la Figura 13.

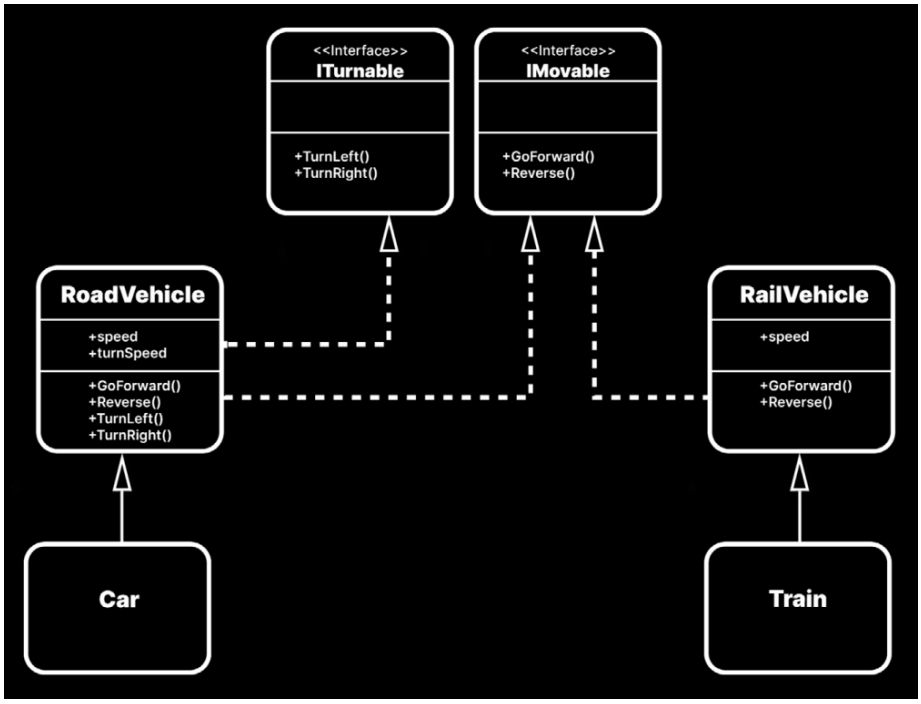


Figura 13: Diseño refactorizado de la clases usando el principio de sustitución de Liskov. (Unity, 2021a)

Al utilizar interfaces en lugar de métodos concretos en la clase base, logramos que las subclases sean sustituibles sin problemas y evitamos comportamientos inconsistentes. Además, esta solución nos brinda flexibilidad para agregar nuevos comportamientos o adaptarlos según las necesidades de cada tipo de vehículo.

6.1.3.2.4. Principio de segregación de la interfaz

El principio de segregación de interfaces establece que ningún clase debe depender de métodos que no utiliza. En otras palabras, debemos evitar la creación de interfaces demasiado grandes que obliguen a las clases a depender de funcionalidades que no necesitan. Esto nos permite mantener una estructura más flexible y fácil de

mantener.

Para ilustrar este principio, consideremos la creación de un juego en el que hay diferentes tipos de unidades con diversas estadísticas, como salud, defensa y fuerza. Podríamos crear una interfaz que garantice que todas las unidades implementen estas características comunes, como se muestra en la Figura 14.

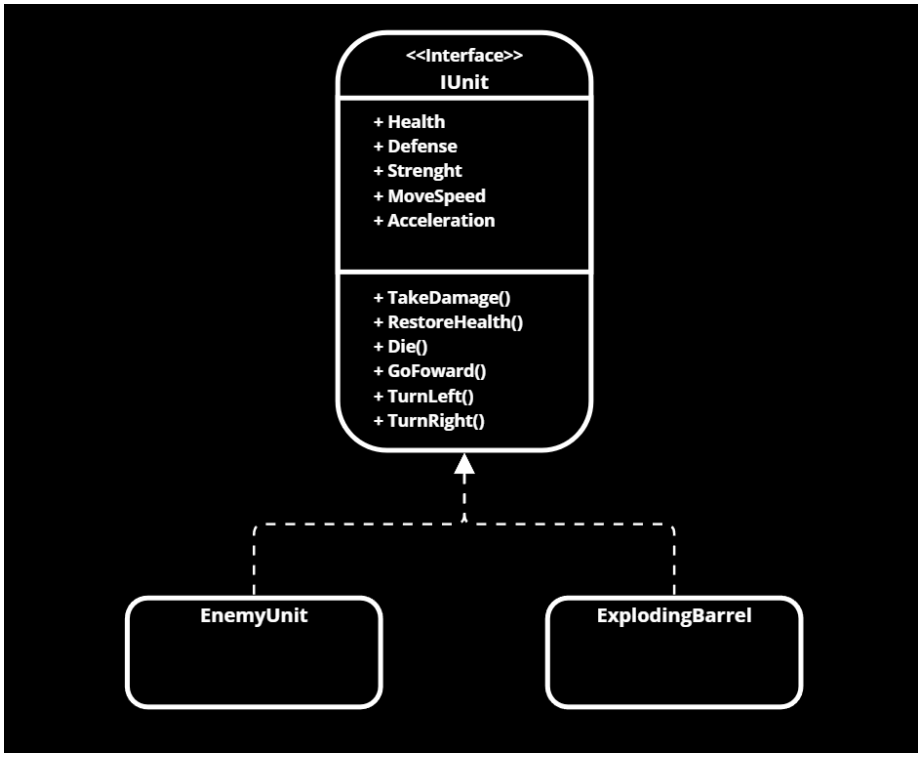


Figura 14: Estructura de clases implementando una única interfaz. Fuente propia

Sin embargo, esta implementación viola el principio de segregación de interfaces, ya que tenemos demasiados métodos dentro de esta interfaz que hacen que las unidades que puedan implementar dicha interfaz sean limitadas y le generan una complejidad innecesaria a dicha interfaz.

Para solucionar este problema, debemos dividir la interfaz en interfaces más pequeñas

y coherentes, cada una con responsabilidades más específicas. Esto se ilustra en la Figura 15.

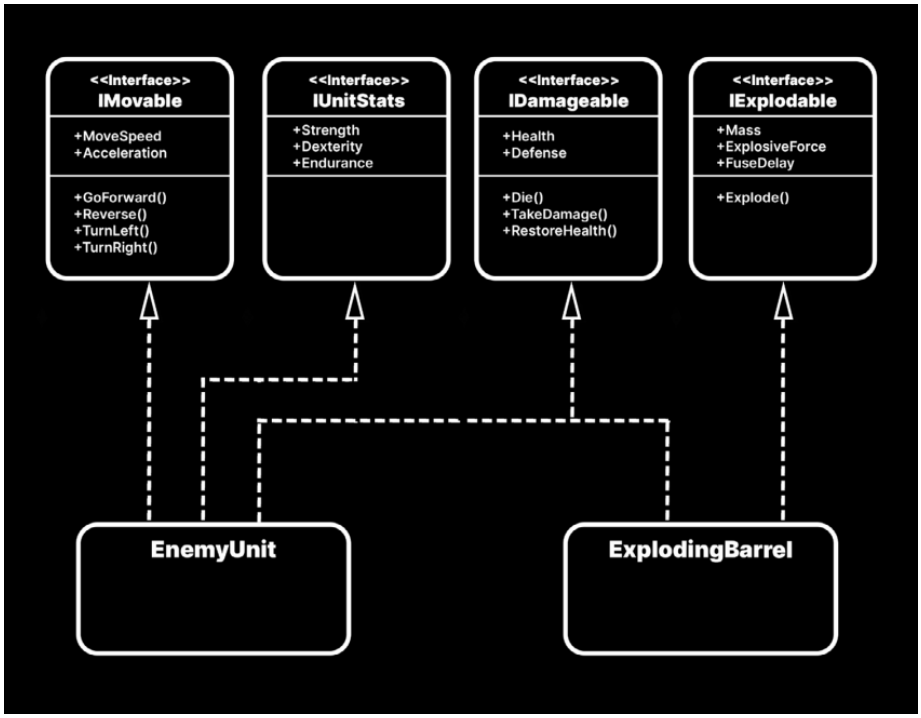


Figura 15: Diseño refactorizado de la interfaz dividida usando el principio de segregación de la interfaz. (Unity, 2021a)

6.1.3.2.5. Principio de inversión de la dependencia

El principio de inversión de dependencia establece que los módulos de alto nivel no deben depender directamente de los módulos de bajo nivel, sino que ambos deben depender de abstracciones. Esto implica que las clases deben depender de interfaces o clases abstractas en lugar de depender de implementaciones concretas.

Cuando hay una alta dependencia entre clases, es decir, un alto acoplamiento, cualquier modificación en una clase puede afectar el funcionamiento de otras clases y

viceversa. Por lo tanto, se considera una buena práctica reducir el acoplamiento entre clases.

Tomemos como ejemplo el caso en que deseemos crear una funcionalidad para abrir y cerrar una puerta. Para esto podríamos crear una clase **Switch** y otra **Door**.

Tomemos como ejemplo el escenario en el que queremos implementar la funcionalidad de abrir y cerrar una puerta. Para esto, creamos una clase llamada **Switch** y otra llamada **Door**.

A alto nivel, la clase **Switch** es responsable de controlar el evento de apertura y cierre de la puerta. A nivel bajo, la clase **Door** simplemente se encarga de ejecutar los métodos **Open()** o **Closed()**.

En este caso, podríamos tener un método llamado **Toggle()** dentro de la clase **Switch** que, dependiendo del valor del parámetro **isActivated**, llamará al método **Open()** si es verdadero o al método **Closed()** si es falso, pertenecientes a la clase **Door**. Esto se puede visualizar en la Figura 16.

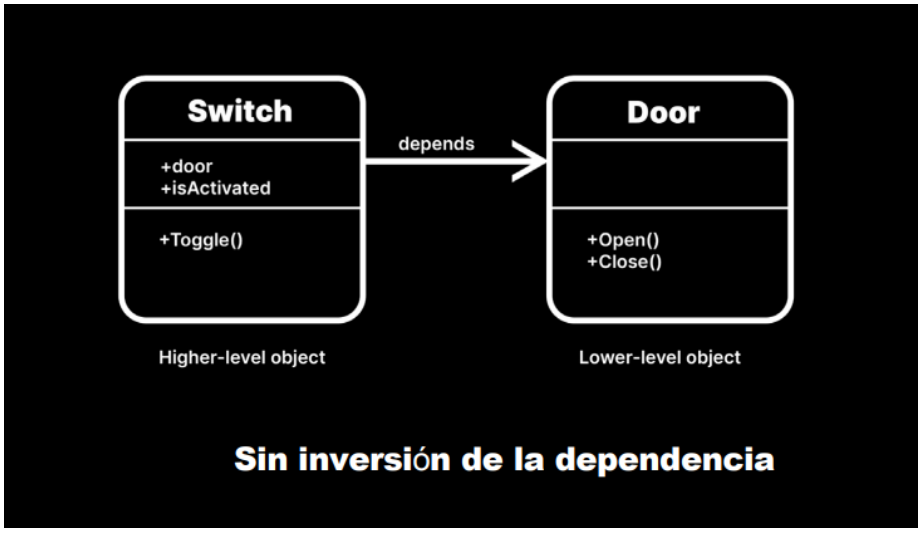


Figura 16: Diseño de clases con alto nivel de acoplamiento. (Unity, 2021a)

Si en el futuro deseamos agregar más funcionalidades a la clase **Switch**, como encender y apagar una luz, estaríamos violando el principio de abierto-cerrado que mencionamos anteriormente.

Para solucionar este problema de acoplamiento, podemos crear una interfaz llamada **ISwitchable** que contenga los métodos necesarios para activar y desactivar cualquier clase que la implemente. En este caso, tenemos la clase **Door**, pero fácilmente podría ser implementada en otras clases futuras como **Light** o **Window**.

En la Figura 17 se puede observar la nueva interfaz **ISwitchable** entre las dos clases.

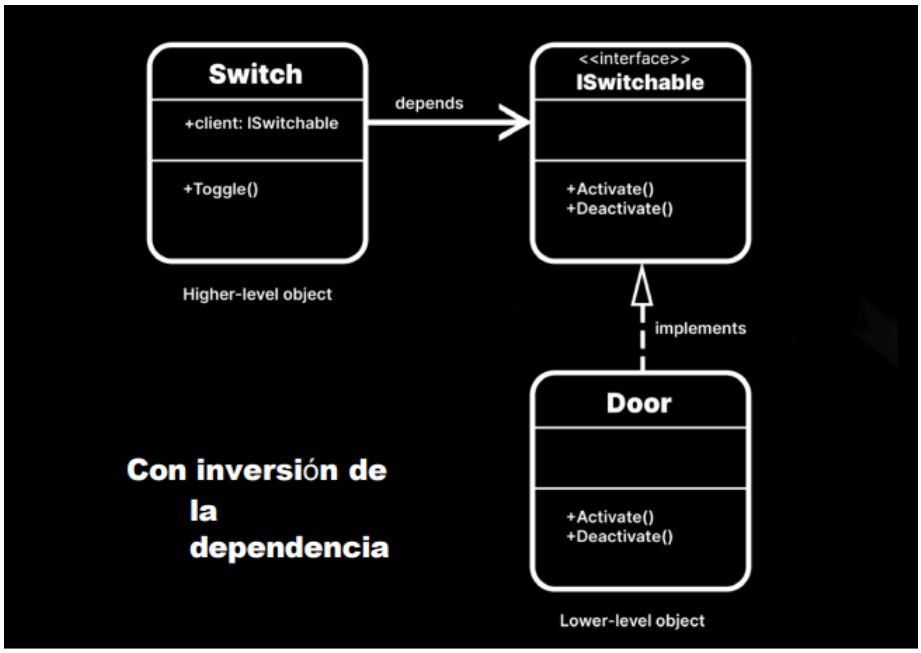


Figura 17: Diseño refactorizado con la interfaz usando el principio de inversión de la dependencia. (Unity, 2021a)

6.1.3.3. Patrones de diseño para desarrollo de videojuegos

A continuación, se mencionan algunos de los patrones más utilizados en el desa-

rrollo de videojuegos según libro publicado por Unity llamado “Level up your code with game programming patterns” (Unity, 2021a).

6.1.3.3.1. Factory Pattern

En ocasiones es útil contar con un objeto especial que se encargue de crear otros objetos. En los juegos, se generan diferentes elementos a lo largo del juego y, a menudo, no sabemos qué elementos necesitaremos en tiempo de ejecución hasta que realmente los necesitemos.

El **Factory Pattern** se refiere a un objeto especial llamado, como su nombre lo indica, una fábrica, que cumple con esta función. A un nivel básico, la fábrica encapsula los detalles involucrados en la creación de los “productos”. El beneficio inmediato de este patrón es que nos ayuda a mantener nuestro código más organizado y limpio.

Sin embargo, si cada producto sigue una interfaz común o una clase base, podemos llevar esto aún más lejos y permitir que los productos tengan su propia lógica de construcción, manteniéndola oculta a la fábrica. De esta manera, la creación de nuevos objetos se vuelve más flexible y adaptable.

Además, podemos crear subclases de la fábrica para tener fábricas dedicadas a productos específicos. Esto nos ayudaría a generar enemigos, obstáculos u otros elementos durante la ejecución del juego.

Un ejemplo practico de este patrón seria el caso en que necesitemos instanciar objetos en un nivel de un juego y ejecutemos un comportamiento personalizado por cada instancia. En lugar de usar declaraciones **If** o **Switch** para realizar esta lógica, lo ideal seria crear una interfaz llamada **IProduct** y una clase abstracta llamada **Factory**.

La estructura de lo anterior la podemos observar en la Figura 18.

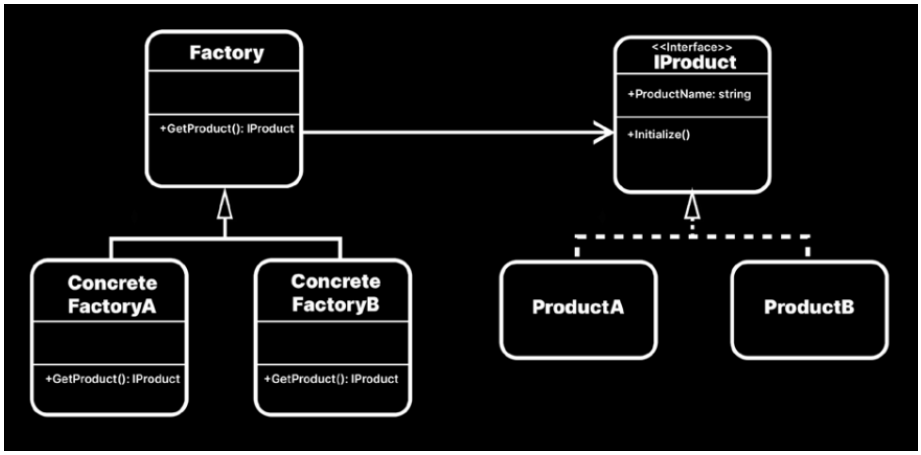


Figura 18: Ejemplo practico del uso del Factory Pattern. (Unity, 2021a)

La interfaz **IProduct** define las características comunes entre los diferentes productos, y con ella podemos definir cualquier cantidad de productos. En este ejemplo, se ejemplifica con las clases **ProductA** y **ProductB** que implementan esta interfaz.

La clase **Factory** es abstracta y contiene un método GetProduct, que devuelve un objeto de tipo **IProduct**. Dado que la clase **Factory** es abstracta y no se puede instanciar directamente, es necesario crear clases adicionales que implementen la clase **Factory**. En este caso, estas clases adicionales serían **ConcreteFactoryA** y **ConcreteFactoryB**.

Finalmente tenemos una **Factory** en sí que no contiene ninguna lógica específica para activar diferentes funcionalidades simplemente se invoca el método Initialize, que es común a todos los productos, y en cada producto podemos tener diferentes comportamientos que puedan ser necesarios.

6.1.3.3.2. Command Pattern

El patrón de diseño Command es útil cuando se quiere realizar un seguimiento de una serie específica de acciones. Es común encontrar este patrón en juegos que tienen funcionalidad de deshacer/rehacer o mantienen un historial de acciones del usuario. Por ejemplo, en un juego de estrategia donde el usuario puede planificar varios turnos antes de ejecutarlos.

En lugar de invocar directamente un método, el patrón Command permite encapsular una o más llamadas de método en un **Command object**. Estos **Command object** se almacenan en una colección, como una cola o una pila, lo que permite controlar el momento de su ejecución. Esto funciona como un pequeño búfer donde se pueden retrasar acciones para reproducirlas en el futuro o deshacerlas.

Para implementar el patrón Command, se necesita un objeto general que contenga la acción a realizar. Este **Command object** incluye la lógica específica de la acción y también cómo deshacerla en caso de que sea necesario. Este proceso se ilustra en la Figura 19.

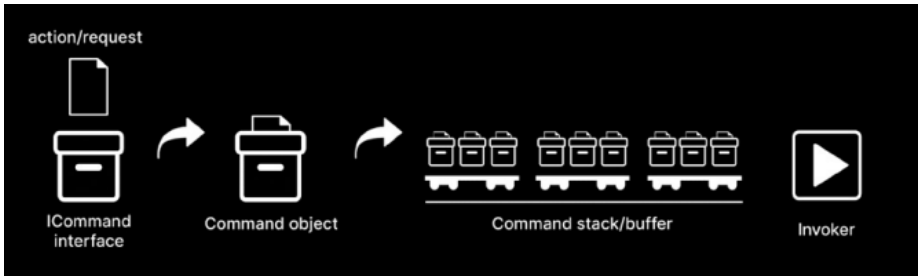


Figura 19: Proceso de guardar acciones mediante el patrón Command. (Unity, 2021a)

6.1.3.3.3. Observer Pattern

A menudo en los videojuegos, necesitamos un mecanismo que permita que algu-

nos objectos notifiquen a otros sin hacer referencia directa a ellos, evitando crear dependencias innecesarias.

El patrón observador es una solución para este problema, ya que permite que los objetos se comuniquen entre sí con un acoplamiento flexible utilizando una relación de "uno a muchos". Cuando un objeto cambia de estado, todos los objetos dependientes son notificados automáticamente.

El objeto que emite las notificaciones se conoce como el **Sujeto** (Subject), mientras que los objetos que están escuchando se denominan **Observadores** (Observers).

Este patrón logra un desacoplamiento efectivo del **Sujeto**, ya que este no tiene conocimiento de los **Observadores** ni le importa qué acciones realizan una vez que reciben la notificación. Aunque los **Observadores** dependen del **Sujeto**, no tienen conocimiento unos de otros.

En la Figura 20 dicha relación entre el **Sujeto** y los **Observadores**.

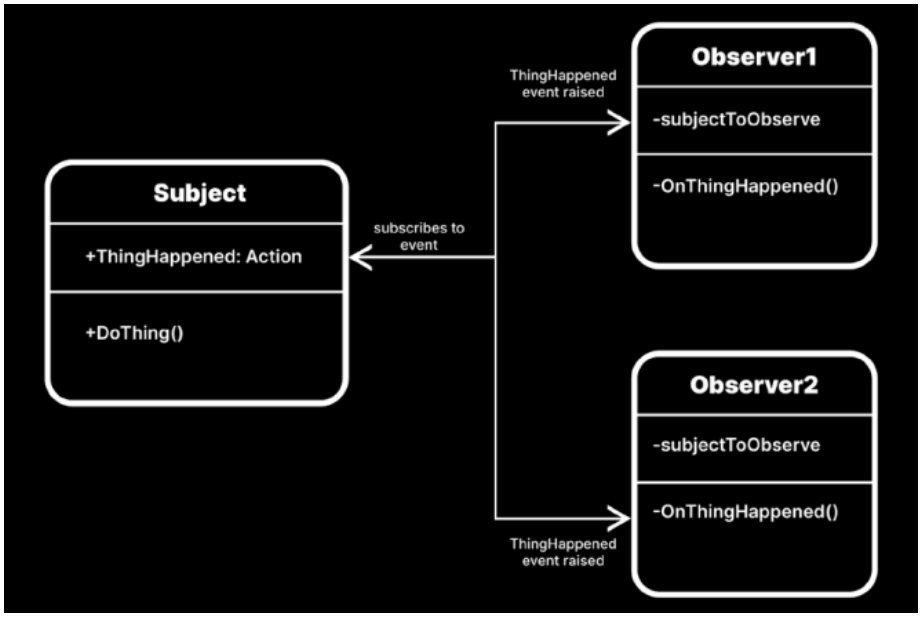


Figura 20: Relación entre el sujeto y los observadores. (Unity, 2021a)

6.1.3.3.4. Model View Presenter

El MVP (Model View Presenter) es una variación del patrón MVC (Modelo Vista Controlador) que mantiene la separación de responsabilidades en las tres capas de la aplicación. Sin embargo, cambia ligeramente las responsabilidades de cada parte.

En el MVP, el **Presenter** (llamado **Controller** en el MVC) actúa como intermediario entre las otras capas. Se encarga de obtener los datos del **Model** y formatearlos para su visualización en la **View**. A diferencia del MVC, en el MVP la **View** es la responsable de manejar la entrada del usuario, en lugar del **Controller**.

La capa de **View** envía las entradas de vuelta al **Presenter** a través de eventos de la interfaz de usuario, y el **Presenter**, a su vez, manipula el **Model**. Cuando ocurre un cambio en el estado del **Model**, se genera un evento que informa al **Presenter** que los datos se han actualizado. El **Presenter** luego transmite los datos modificados a

la **View**, que se encarga de actualizar la interfaz de usuario en consecuencia.

En la Figura 18 se puede observar este flujo de comunicación entre las diferentes capas.

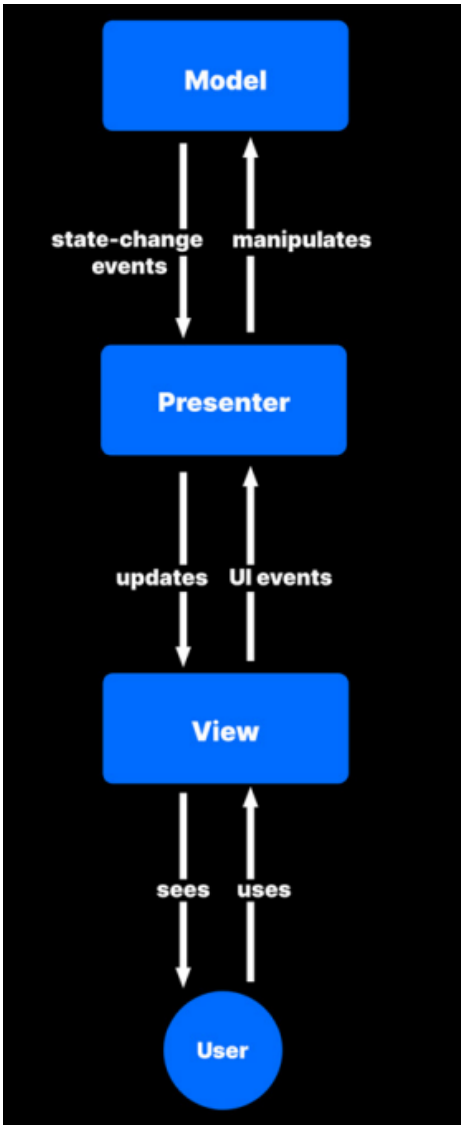


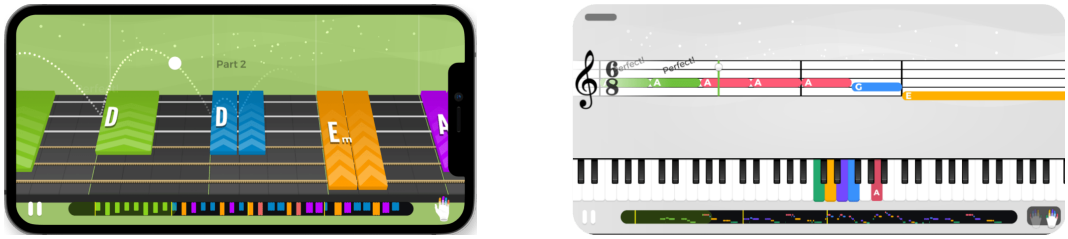
Figura 21: Flujo de comunicación entre capas usando MVP. (Unity, 2021a)

6.2. Estado del Arte

6.2.1. Antecedentes de aplicaciones de aprendizaje musical

6.2.1.1. Yousician

(Yousician, 2023) es una aplicación de enseñanza de música interactiva que utiliza tecnología para enseñar a tocar instrumentos musicales, como la guitarra, el piano, el bajo y el ukelele. La aplicación ofrece lecciones en video y ejercicios prácticos para ayudar a los usuarios a aprender a tocar su instrumento elegido de manera efectiva. También ofrece una función de reconocimiento de notas para ayudar a los usuarios a afinar su oído y mejorar su habilidad para reconocer las notas correctas. La aplicación es popular entre los principiantes y los músicos experimentados por igual debido a su enfoque práctico y personalizado para la enseñanza de la música.



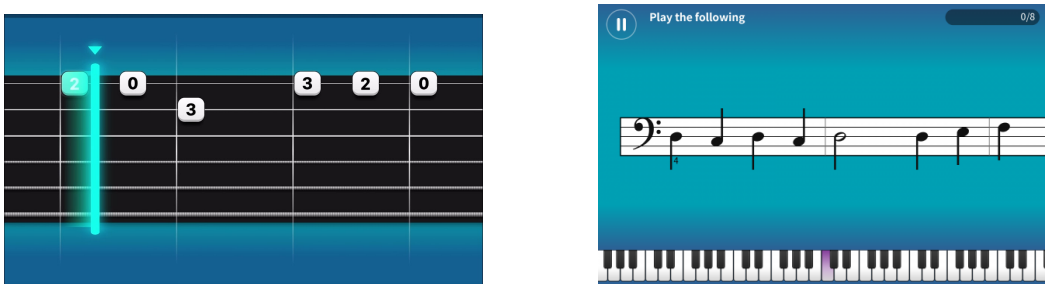
(a) Yousician para guitarra. **(b)** Yousician para piano.

Figura 22: Yousician Apps. Fuente (Yousician, 2023)

6.2.1.2. Simply

(Simply, 2023) es un ecosistema de aplicaciones de aprendizaje musical que utilizan tecnología de escucha para ayudar a los usuarios a aprender a tocar diferentes instrumentos como el piano, la guitarra, ukelele, bajo y hasta el canto. Las aplicaciones están diseñadas para principiantes y ofrecen una amplia variedad de canciones

para aprender, desde música clásica hasta pop y rock. Los usuarios pueden aprender a su propio ritmo y seguir su progreso a medida que avanzan en las lecciones. Simply también proporciona retroalimentación en tiempo real sobre la precisión y la velocidad de tocar, lo que ayuda a los usuarios a mejorar su técnica.



(a) Simply para guitarra. **(b)** Simply para piano.

Figura 23: Simply Apps. Fuente (Simply, 2023)

6.2.1.3. Sing Sharp

(Sing Sharp, 2023) es una aplicación de enseñanza de canto que proporciona a los usuarios ejercicios y lecciones diseñados para mejorar su técnica vocal y capacidad de interpretación. La aplicación ofrece una amplia variedad de canciones para practicar y permite a los usuarios grabar y escuchar sus propias interpretaciones para identificar áreas de mejora. Además, la aplicación incluye una función de retroalimentación en tiempo real que proporciona comentarios sobre la precisión y el tono de la voz del usuario.

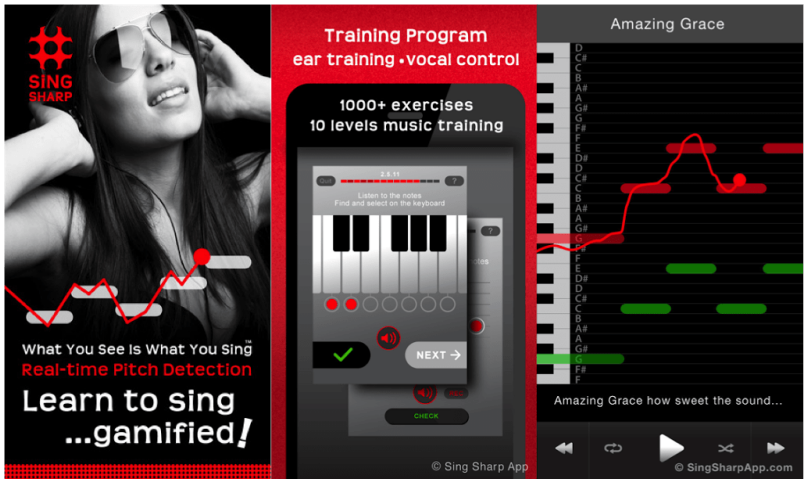


Figura 24: Sing Sharp App. Fuente (Sing Sharp, 2023)

6.2.2. Estudios relacionados con el aprendizaje musical a través de videojuegos

6.2.2.1. El videojuego Synthesia

El trabajo de (Lara Rodriguez, 2017) presenta un estudio sobre la incorporación de videojuegos como herramienta didáctica en la educación musical inicial en niños. Se evaluó el proceso de aprendizaje generado con la implementación del videojuego Synthesia en la Academia de enseñanza musical Baqueta ubicada en Bogotá. El estudio analiza cómo la utilización de videojuegos en la educación musical puede generar procesos de aprendizaje significativo en los niños, aportando conocimientos técnicos y teóricos que logren una adecuada interpretación y ejecución del piano como instrumento musical.

Además, se describe cómo la incorporación de las Tecnologías de la Información y la Comunicación, en este caso los videojuegos, ha modificado los currículos educativos, alternando en los planes de estudio el uso de videojuegos como instrumentos de

fortalecimiento de conocimiento práctico y teórico dentro y fuera del aula de clase.

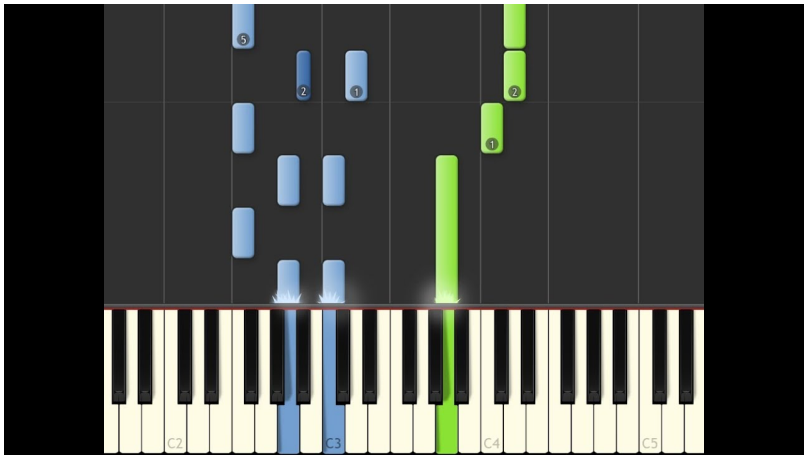


Figura 25: Synthesia App. Fuente (Synthesia, 2023)

6.2.2.2. Musikami

El trabajo de (Sánchez Rueda, 2017) presenta la creación de una solución informática llamada Musikami, que apoya la educación musical en niños a través de una aplicación web con soporte en dispositivos móviles. La idea surge de los beneficios comprobados de la educación musical en niños, como una mejora en las habilidades comunicativas y sociales, el desarrollo de la imaginación y el pensamiento flexible, entre otros.

Además, el rápido incremento en el uso de dispositivos móviles por parte de niños y jóvenes lleva a la necesidad de ofrecer herramientas educativas interactivas y atractivas. Por lo tanto, Musikami busca ser una herramienta de apoyo a la enseñanza musical en la población infantil, que tenga en cuenta distintos factores de la enseñanza musical y permita su visualización tanto en un navegador web como en dispositivos móviles.

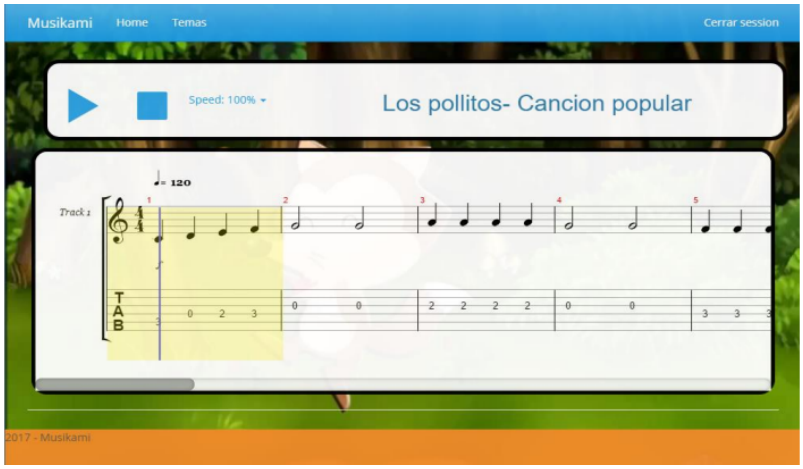


Figura 26: Musikami App. Fuente (Sánchez Rueda, 2017)

7. Metodología del proyecto

La metodología que he decidido adoptar para este proyecto es una variante simplificada de la metodología ágil, adaptada específicamente para un desarrollador en solitario. En lugar de seguir un enfoque estrictamente estructurado, he optado por una estrategia que sea flexible y con la cual sea posible adaptarse a las necesidades cambiantes del desarrollo de este prototipo del videojuego.

7.1. Planificación

Siguiendo las directrices de (Keith, 2010) en su obra "Agile Game Development with Scrum", he adoptado un enfoque de planificación del proyecto basado en la creación de pequeños hitos que permiten entregas incrementales de las diversas funcionalidades del juego. Con este propósito, he establecido milestones mensuales que nos permitirán disponer de estas funcionalidades para pruebas de manera gradual.

La estructura de milestones se ha diseñado de la siguiente manera:

Figura 27: Milestones del proyecto. Fuente propia

- **01-July:** Durante este período, se enfocó en la recolección de datos clave, la elaboración del Documento de Diseño del Juego (GDD) y la planificación del diseño de la interfaz gráfica.
- **02-August:** En el segundo mes, nos dedicamos a investigaciones adicionales, la configuración inicial del proyecto y, adicionalmente, definimos la arquitectura, desarrollamos y realizamos las pruebas correspondientes del Módulo de Autenticación.
- **03-September:** El tercer milestone se enfocó en la definición de la arquitectura para el Módulo de Menús, seguido del desarrollo de este módulo, las pruebas correspondientes y la documentación detallada del Módulo de Autenticación.
- **04-October:** En octubre, continuamos definiendo la arquitectura, esta vez para el Módulo de Mapas, y procedimos con su desarrollo y las pruebas correspondientes. Además, documentamos el Módulo de Menús y empezamos a definir la arquitectura para los Módulos de Juego, tanto Desafío como Práctica.
- **05-November:** Durante este mes, completamos la documentación del Módulo de Mapas y nos concentramos en iniciar el desarrollo del Módulo de Juego - Desafío y Práctica.
- **06-December:** En diciembre, el enfoque fue terminar el Módulo de Juego - Desafío y Práctica y realizar sus respectivas pruebas.
- **07-January:** Nuestro último milestone se centró en la documentación del Módulo de Juego - Desafío y Práctica, así como en las pruebas de desempeño con los usuarios.

7.2. Notaciones

Para la notación de las historias de usuario, se ha decidido implementar un sistema de etiquetas utilizando el término **"userstory"**. A cada historia de usuario se le de-

signará su propio label, siguiendo el formato **US-001**, **US-002**, y así sucesivamente, para todas las historias de usuario necesarias.

En cuanto a la notación de las tareas, se ha establecido un label denominado **"task"**. Cada tarea estará vinculada su respectiva historia de usuario mediante la asignación del label correspondiente.

7.2.1. Historia de Usuario

- **Etiqueta:** userstory
- **Identificador de la Historia de Usuario:** US-XXX

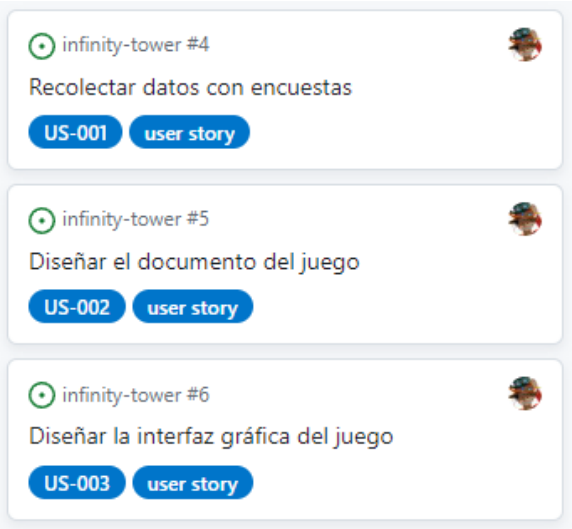


Figura 28: Etiquetas de historias de usuario. Fuente propia

7.2.2. Tarea

- **Etiqueta:** task

- **Identificador de la Historia de Usuario:** US-XXX

Figura 29: Etiquetas de tareas. Fuente propia

7.3. Ventajas y beneficios

- **Flexibilidad y Adaptabilidad:** Como único desarrollador, puedo tomar decisiones rápidas y ajustar la dirección del proyecto según sea necesario.
- **Iteraciones Constantes:** Al dividir el proyecto en milestones mensuales, estoy realizando iteraciones constantes en el desarrollo del juego. Esto significa que puedo centrarme en objetivos específicos durante un período de tiempo determinado, lo que facilita la gestión del progreso.
- **Entregas Incrementales:** Comenzar con una versión mínima viable (MVP) y luego construir sobre ella permite una entrega incremental de características.

Esto significa que puedo tener una versión jugable temprana del juego y agregar características gradualmente, lo que facilita las pruebas y la retroalimentación temprana.

- **Retroalimentación Continua:** La realización de pruebas frecuentes y la solicitud de retroalimentación permiten mejorar constantemente el juego y adaptarlo a las preferencias de los usuarios. Esto ayuda a garantizar una experiencia de juego de alta calidad.

8. Desarrollo trabajo de grado

8.1. Diseño del documento del juego

Como desarrollador, es fundamental crear un Documento de Diseño del Juego (GDD) que sea sólido y completo y en el cual esté definida la visión y los elementos clave del juego. El GDD sirve como una guía central para el desarrollo del juego y garantiza una comprensión clara de las mecánicas, la narrativa y los objetivos de aprendizaje.

- **Resumen:** El resumen proporciona una visión general y concisa de todo el proyecto. Es importante ya que permite a los lectores obtener rápidamente una comprensión básica del juego sin tener que sumergirse en detalles más específicos. Esto es especialmente útil para los interesados que puedan no estar directamente involucrados en el desarrollo, como patrocinadores o inversores.
- **Información general del juego:** Esta sección establece el contexto fundamental del proyecto. Proporciona detalles esenciales sobre la naturaleza del juego, su género, plataforma objetivo y público objetivo. Ayuda a todos los involucrados a alinear sus expectativas y comprender el propósito general del juego.
- **Historia:** La historia es un componente crucial de muchos juegos, ya que crea la trama y motiva la acción del jugador. Proporciona un marco narrativo que guía la experiencia del jugador, lo que hace que esta sección sea esencial para establecer la base de la inmersión y la conexión emocional con el juego.
- **Personajes:** Los personajes son a menudo el corazón de una narrativa de juego. Detallar sus personalidades, motivaciones y roles en la historia es importante para desarrollar relaciones significativas entre los jugadores y los personajes. Esto puede afectar significativamente la empatía y la inversión emocional del jugador en el juego.

- **Escenarios:** La ambientación y el mundo del juego son fundamentales para la inmersión. Describir el escenario y sus elementos únicos ayuda a los diseñadores a crear un mundo coherente y envolvente. Esto afecta directamente la experiencia del jugador y su capacidad para sentirse parte del juego.
- **Jugabilidad:** La jugabilidad es la piedra angular de cualquier juego. Detallar las mecánicas, controles, objetivos y sistemas de progresión es esencial para asegurarse de que el juego sea divertido y desafiante. Esto afecta directamente la experiencia del jugador y su capacidad para comprender y disfrutar el juego.
- **Arte y sonido:** El arte y el sonido son componentes clave para la estética y el ambiente del juego. La elección de la paleta de colores, el estilo de arte y la música influye en la atmósfera del juego y en cómo se percibe. Esto puede determinar si un juego se siente atractivo y envolvente o no.
- **Interfaz de usuario:** La interfaz de usuario es la conexión directa entre el jugador y el juego. Diseñar menús, mapas y elementos de juego de manera efectiva afecta la jugabilidad y la accesibilidad. Una interfaz intuitiva y atractiva mejora la experiencia del usuario y garantiza que el juego sea fácil de usar.
- **Diseño de niveles:** El diseño de niveles influye en la fluidez de la experiencia del jugador. Una estructura de niveles bien planificada, objetivos claros y una progresión adecuada mantienen a los jugadores comprometidos y evitan que se sientan abrumados o aburridos.
- **Requerimientos técnicos:** Esta sección especifica las herramientas y tecnologías necesarias para desarrollar y ejecutar el juego. Es vital para garantizar que todos los miembros del equipo estén en la misma página en cuanto a las herramientas a utilizar y las restricciones técnicas. Esto evita problemas durante el desarrollo y garantiza un flujo de trabajo eficiente.

En el Anexo 3, se encuentra una descripción completa de cada elemento, desde el

resumen y la información general del juego hasta los detalles sobre la historia, personajes, escenarios, jugabilidad, arte y sonido, interfaz de usuario, diseño de niveles y requisitos técnicos.

8.2. Diseño de interfaz gráfica del juego

El diseño de la interfaz gráfica de "Infinity Tower" juega un papel crucial en la creación de una experiencia de usuario funcional y envolvente. Para el desarrollo de esta interfaz, se tomaron en cuenta principios y prácticas de diseño establecidos en (Unity, 2021b), que sirvió como guía fundamental para la implementación de las diferentes secciones de la interfaz del juego.

- **Módulo de Autenticación:** La interfaz de autenticación es la primera experiencia que los usuarios tienen con el juego. Debe ser intuitiva y segura para garantizar que los jugadores puedan acceder fácilmente a sus cuentas y comenzar a jugar sin problemas.

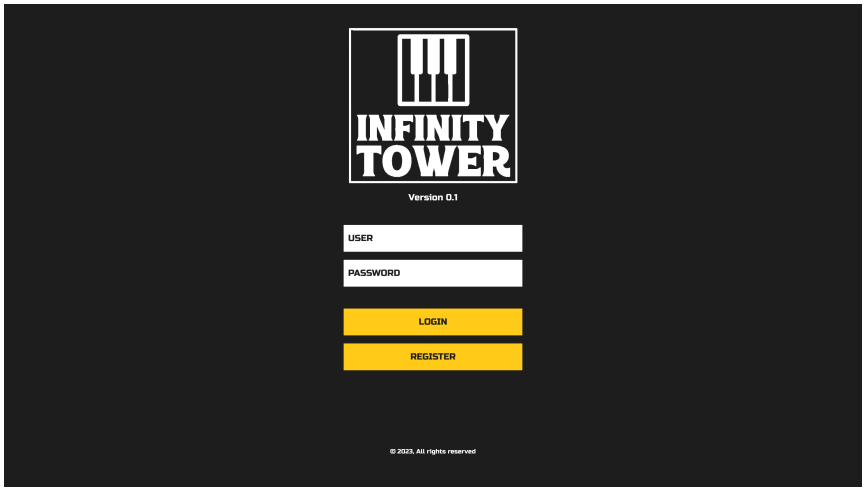


Figura 30: Diseño módulo de autenticación. Fuente propia

- **Módulo de Menús:** Los menús principales y de configuración son elementos fundamentales en el juego. Los menús principales permiten la navegación por el juego y el acceso a funciones clave, mientras que los menús de configuración posibilitan la personalización de la experiencia de juego. Un diseño eficiente y claro en ambos casos es esencial para la satisfacción del usuario.

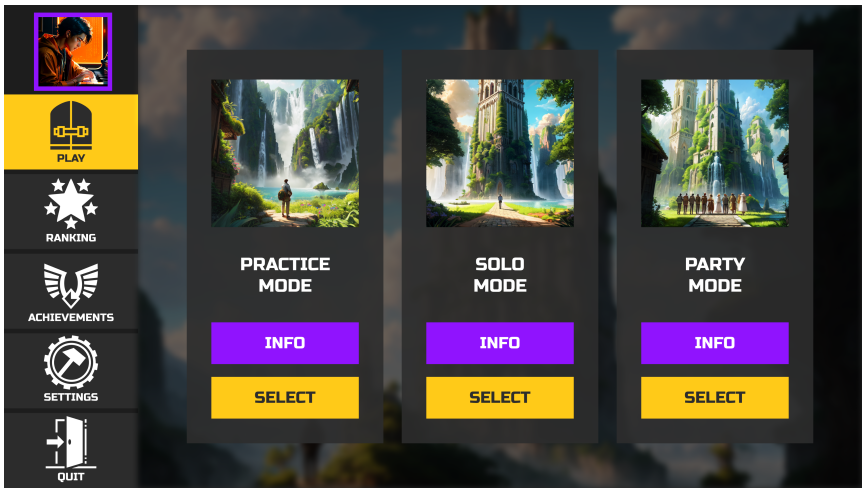


Figura 31: Diseño módulo de menus. Fuente propia

- **Módulo de Mapas:** Los mapas son la puerta de entrada a la exploración en el mundo del juego. Un diseño efectivo de la interfaz de mapas permite a los jugadores navegar por los diferentes pueblos de Melody y acceder a desafíos y aventuras. También debe proporcionar información contextual sobre cada ubicación.

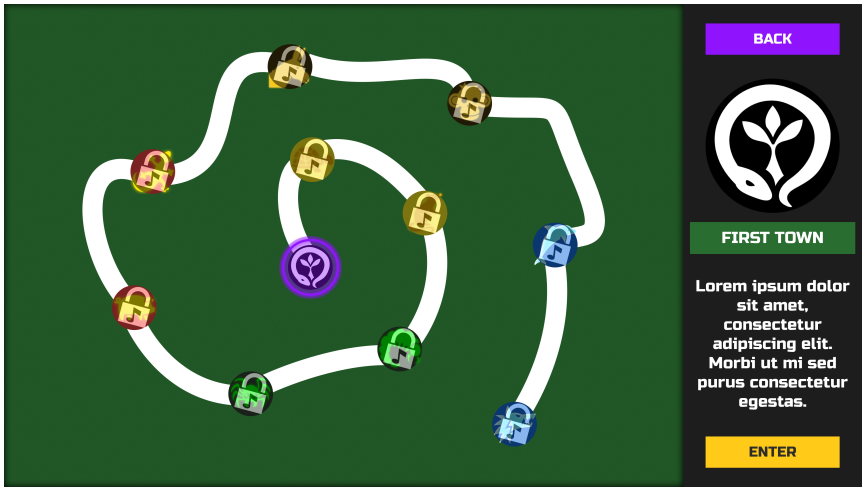


Figura 32: Diseño módulo de mapas. Fuente propia

- **Módulo de Juego - Desafío:** La interfaz gráfica durante los desafíos musicales es fundamental para que los jugadores monitoreen su progreso y rendimiento en tiempo real. Debe ser claro y proporcionar información esencial, como la puntuación, los combos y la precisión. Un buen diseño de la interfaz gráfica permite una jugabilidad fluida y una conexión más profunda con la música.

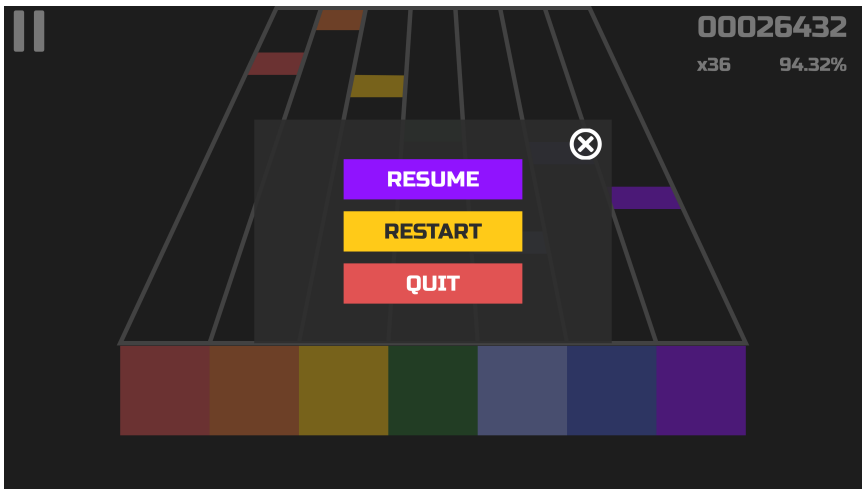


Figura 33: Diseño módulo de juego - Desafío. Fuente propia

- **Módulo de Juego - Práctica:** Al igual que en los desafíos, la interfaz gráfica en el modo de práctica debe ofrecer información relevante sobre el rendimiento del jugador. Esto ayuda a los jugadores a mejorar sus habilidades musicales. Además, debe incluir consejos y orientación para la mejora continua.

Para obtener una visión detallada y completa del diseño de la interfaz gráfica en su totalidad, se recomienda consultar el Anexo 4. En este anexo, se encuentra toda la información sobre la planificación y la implementación de cada uno de los elementos que componen la interfaz del juego.

8.3. Diseño de la arquitectura de software

Para diseñar la arquitectura de software del proyecto, hemos optado por adoptar la metodología de Attribute Driven Design (ADD) para abordar el diseño de cada uno de los módulos. Esta elección se basa en la naturaleza modular del sistema, lo que nos permite enfocarnos individualmente en atributos específicos y requisitos clave de cada módulo. Esta aproximación facilita una comprensión más clara y detallada de las responsabilidades y comportamientos esperados para cada componente.

8.3.1. Módulo core

El Módulo core tiene como objetivo principal supervisar aspectos esenciales para el funcionamiento del sistema. Entre sus responsabilidades se incluye el manejo de configuraciones como video, audio e idioma, permitiendo cambios dinámicos en tiempo de ejecución desde cualquier otro módulo. Además, este componente se encargará de crear, obtener y editar el contenido del juego, que engloba elementos como textos, modos de juego, roles y logros.

Dado que la función principal de este módulo es servir a todos los demás módulos, se hace imprescindible la implementación de un sistema basado en eventos. La finalidad es que estos eventos provenientes de otros módulos puedan ser escuchados por el módulo core, simplificando así la comunicación y la gestión de eventos en el sistema.

Con el objetivo de lograr un rendimiento óptimo en este módulo, se ha optado por la implementación del patrón Singleton para cada uno de los diversos Managers esenciales, tales como Settings, Video, Audio, TextContent y Game. Esta elección se fundamenta en la necesidad de tener una única instancia de estos Managers, asegurando su disponibilidad en tiempo de ejecución y permitiendo su acceso desde distintos módulos según sea necesario. Además, para favorecer la extensibilidad del sistema, se ha incorporado el principio Open/Close, asegurando que el código pueda ser ampliado para introducir nuevas funcionalidades o Managers sin requerir modificaciones en el código existente.

En el diagrama de contexto que se observa en la Figura 34, se pueden visualizar todos los componentes asociados con los diversos procesos, actores y tecnologías que se emplearán en este módulo. Para obtener una comprensión más detallada del diseño de la arquitectura de este módulo, se recomienda consultar el Anexo 5, donde se incluyen todos los diagramas necesarios.

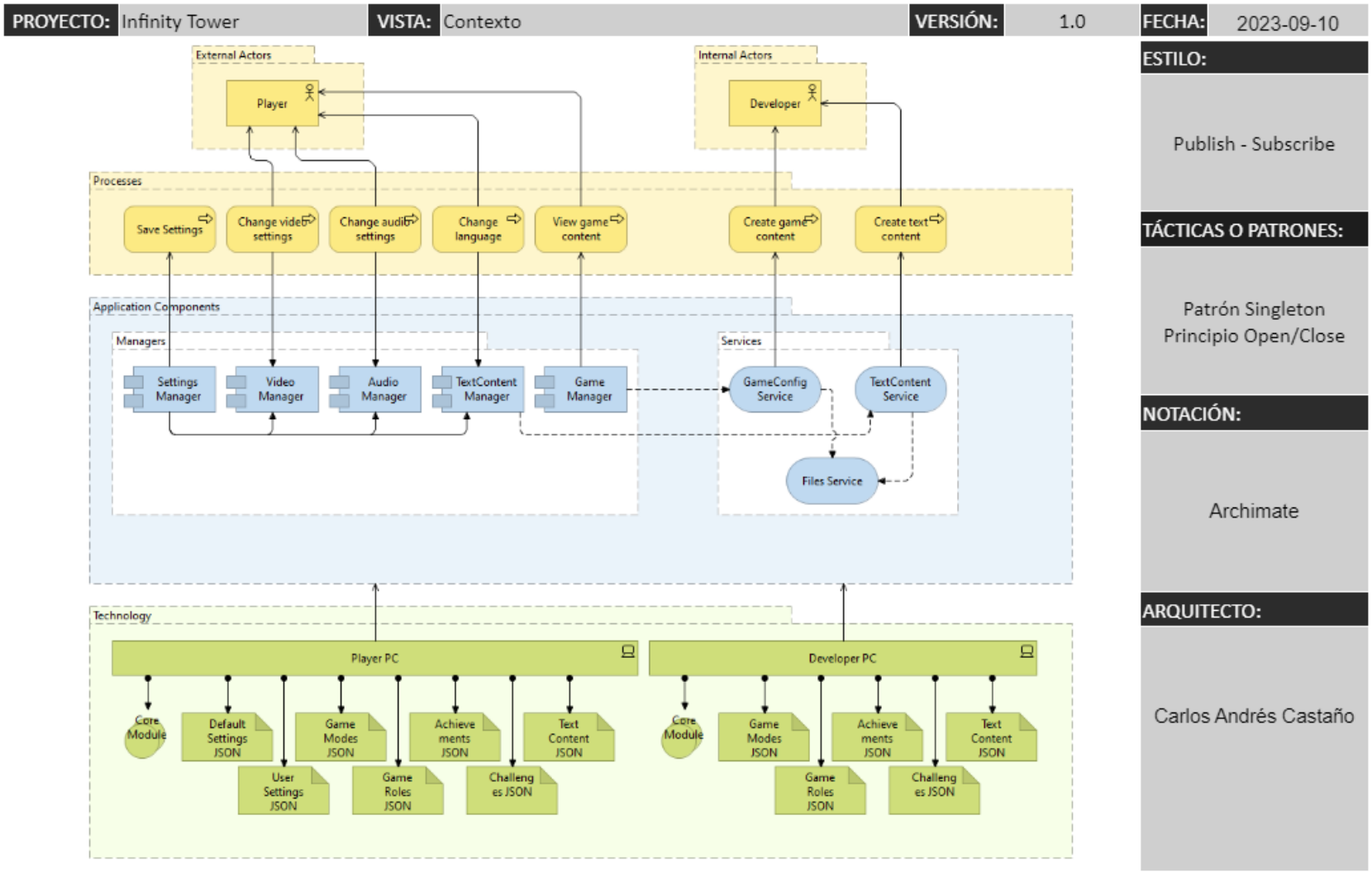


Figura 34: Diagrama de contexto módulo core. Fuente propia

8.3.2. Módulo de autenticación

El objetivo principal del módulo de autenticación (Módulo Auth) es facilitar las funciones relacionadas con la identificación de usuarios en el sistema. Esto incluye permitir a los usuarios registrarse, iniciar sesión y cerrar sesión. Además, el módulo ofrece la capacidad de cambiar dinámicamente el servicio de autenticación utilizado, brindando flexibilidad en la implementación.

Lo que se quiere lograr es una implementación modular y escalable que permita la incorporación de nuevos servicios de autenticación en el futuro sin que esto implique modificaciones en el código principal del módulo.

Para la implementación efectiva en este módulo, se ha optado por la utilización de patrones de diseño específicos. En primer lugar, se ha aplicado el patrón de diseño Strategy para definir diversas estrategias de autenticación, adaptadas a las necesidades del proyecto, como por ejemplo, la integración con Unity Services y Firebase. Este enfoque permite la flexibilidad de incorporar nuevas estrategias según sea necesario.

Adicionalmente, se ha empleado el patrón de diseño Factory para la creación de objetos de autenticación, basándose en la estrategia seleccionada. La función principal de esta fábrica es actuar como intermediario entre el cliente y los distintos servicios de autenticación. Esto posibilita que el cliente solicite una instancia de autenticación sin necesidad de conocer los detalles específicos de su implementación, garantizando así un bajo acoplamiento y una mayor flexibilidad en la integración de nuevos servicios de autenticación.

En cuanto a la integración de estos servicios con la interfaz visual, se ha adoptado el patrón MVP (Model-View-Presenter). Este enfoque resulta particularmente idóneo para proyectos en Unity. En el patrón MVP, los modelos se encargan de la estructura de los datos, las vistas se centran exclusivamente en la presentación visual de la

información, y los presentadores facilitan la comunicación fluida entre los modelos, las vistas y los servicios de autenticación.

En el diagrama de contexto que se observa en la Figura 35, se pueden visualizar todos los componentes asociados con los diversos procesos, actores y tecnologías que se emplearán en este módulo. Para obtener una comprensión más detallada del diseño de la arquitectura de este módulo, se recomienda consultar el Anexo 6, donde se incluyen todos los diagramas necesarios.

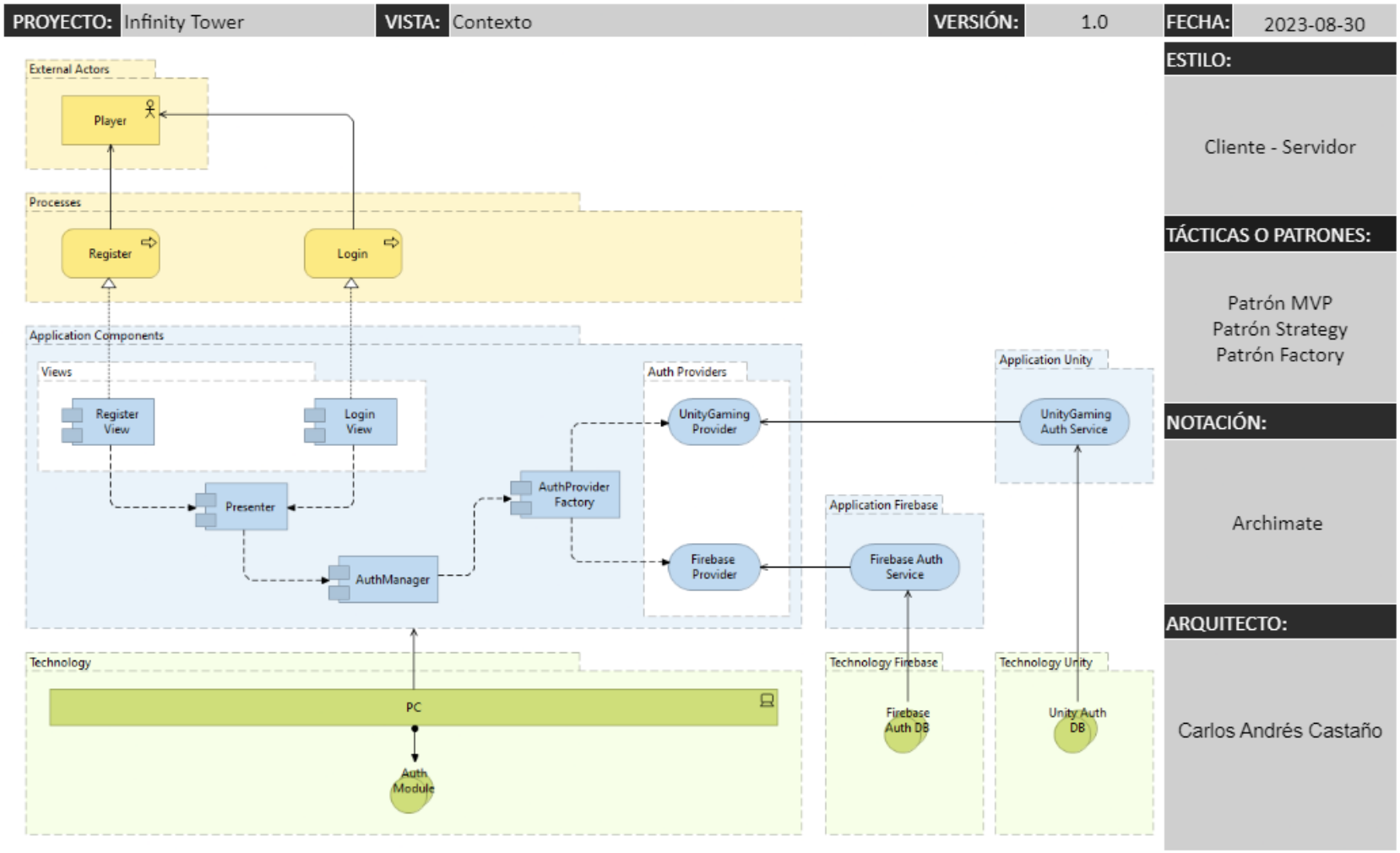


Figura 35: Diagrama de contexto módulo auth. Fuente propia

8.3.3. Módulo de menus

El módulo de Menús tiene como objetivo principal proporcionar una interfaz intuitiva para que los usuarios interactúen con las diferentes funcionalidades del juego.Entre sus responsabilidades, se encuentra obtener la información del usuario y presentarla en la sección de Perfil, mostrar los diversos modos de juego disponibles en la sección de Juego, exhibir rankings asociados a dichos modos en la sección de Rankings y visualizar los logros del juego, tanto bloqueados como desbloqueados, en la sección de Logros. Además, el módulo gestiona las configuraciones de Video, Audio e Idioma del usuario, permitiéndole editarlas y guardarlas en la sección de Configuración. Finalmente, proporciona la funcionalidad de cerrarse mediante la opción de Salir.

Al igual que en el diseño del módulo de autenticación, se ha optado por la utilización de patrones de diseño Strategy, Factory y MVP(Model-View-Presenter). En primer lugar, se ha aplicado el patrón de diseño Strategy en este caso para definir diversas estrategias de obtención de los datos, de la misma forma que se realizo en el módulo de autenticación, para este prototipo se decidió integrar con Unity Services y Firebase permitiendo también la flexibilidad de incorporar nuevas estrategias según sea necesario.

También, se ha empleado el patrón de diseño Factory para la creación de objetos de obtención de los datos, basándose en la estrategia seleccionada. La función principal de esta fábrica es actuar como intermediario entre el cliente y los distintos servicios de obtención de los datos. Esto como se menciono anteriormente garantiza un bajo acoplamiento y una mayor flexibilidad en la integración de nuevos servicios de obtención de los datos.

En cuanto a la integración de estos servicios con la interfaz visual, tampoco hay novedad y se ha adoptado el patrón MVP (Model-View-Presenter). Sin mas que añadir, este enfoque es idóneo para proyectos en Unity, por lo cual se ha usado en

los diferentes módulos.

En el diagrama de contexto que se observa en la Figura 36, se pueden visualizar todos los componentes asociados con los diversos procesos, actores y tecnologías que se emplearán en este módulo. Para obtener una comprensión más detallada del diseño de la arquitectura de este módulo, se recomienda consultar el Anexo 7, donde se incluyen todos los diagramas necesarios.

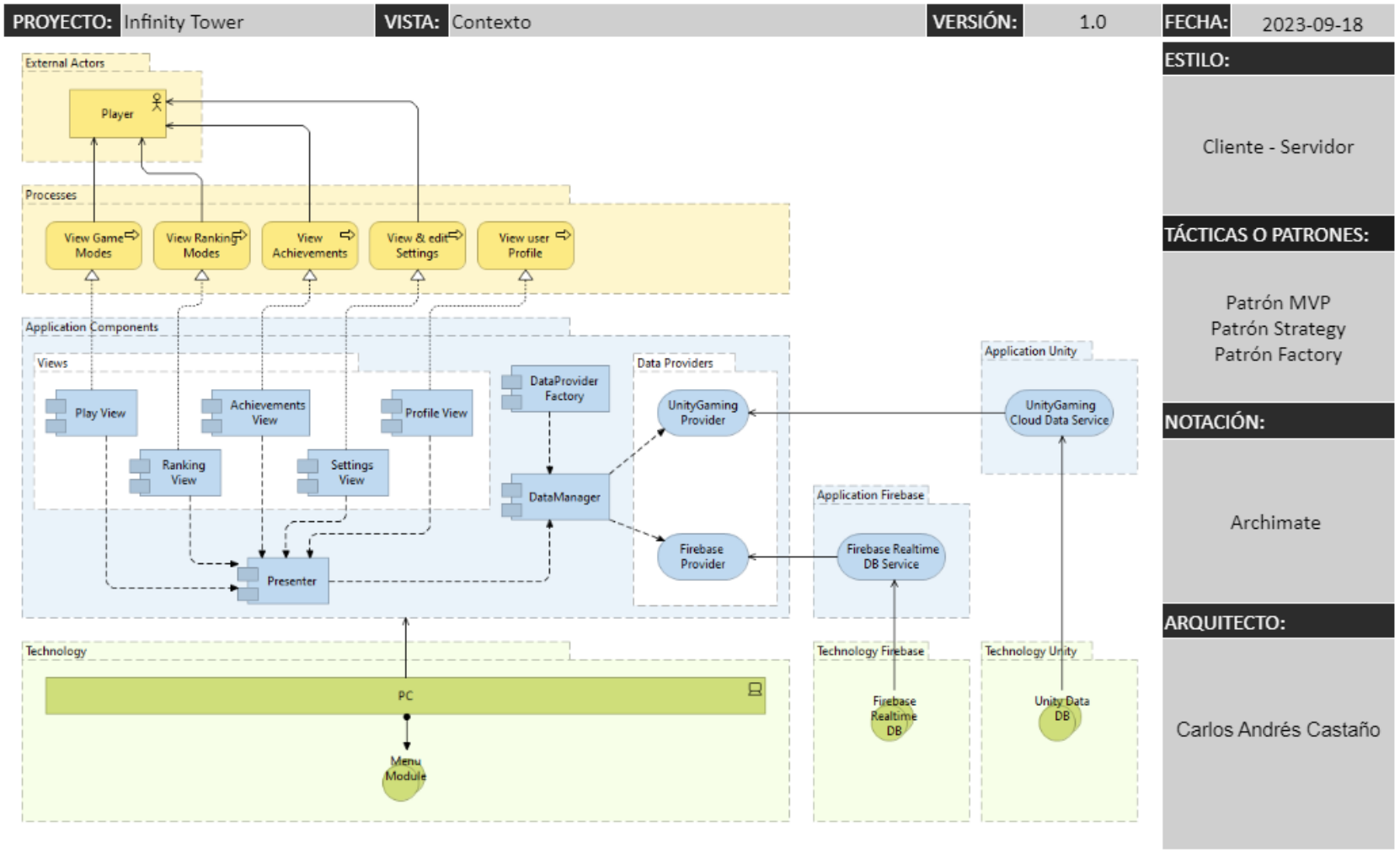


Figura 36: Diagrama de contexto módulo menus. Fuente propia

8.3.4. Módulo de mapas

El módulo de Mapas tiene como objetivo principal proporcionar una interfaz visual que permita a los usuarios explorar las distintas ubicaciones dentro del mundo del juego Melody. Entre sus funciones clave, se incluye la representación gráfica de los pisos y pueblos en el mapa del juego. Además, el módulo debe permitir gestionar la creación, actualización y eliminación de los pueblos y pisos de la torre.

Al igual que en el diseño del módulo de autenticación y menus, se ha optado por la utilización de patrones de diseño Strategy, Factory y MVP(Model-View-Presenter). En primer lugar, se ha aplicado el patrón de diseño Strategy en este caso para definir diversas estrategias de obtención de los datos, de la misma forma que se realizo en el módulo de autenticación y menus, para este prototipo se decidió integrar con Unity Services y Firebase permitiendo también la flexibilidad de incorporar nuevas estrategias según sea necesario.

Del mismo modo, se ha empleado el patrón de diseño Factory para la creación de objetos de obtención de los datos, basándose en la estrategia seleccionada. La función principal de esta fábrica es actuar como intermediario entre el cliente y los distintos servicios de obtención de los datos. Esto como se menciono anteriormente garantiza un bajo acoplamiento y una mayor flexibilidad en la integración de nuevos servicios de obtención de los datos.

Por ultimo, para la integración de estos servicios con la interfaz visual, nuevamente se ha adoptado el patrón MVP (Model-View-Presenter). Como ya se ha mencionado, este enfoque es idóneo para proyectos en Unity, por lo cual se ha usado en los diferentes módulos.

En el diagrama de contexto que se observa en la Figura 37, se pueden visualizar todos los componentes asociados con los diversos procesos, actores y tecnologías

que se emplearán en este módulo. Para obtener una comprensión más detallada del diseño de la arquitectura de este módulo, se recomienda consultar el Anexo 8, donde se incluyen todos los diagramas necesarios.

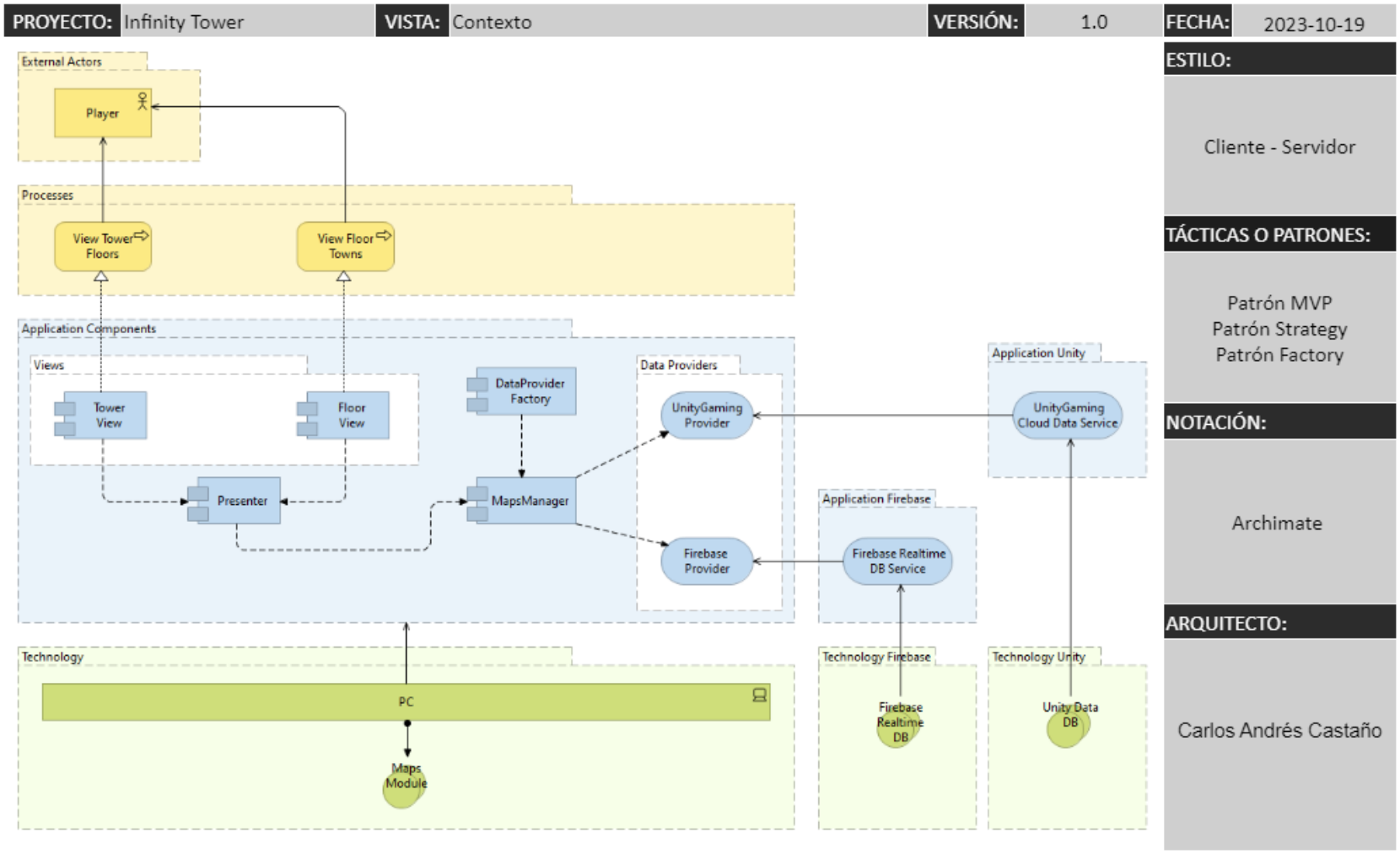


Figura 37: Diagrama de contexto módulo mapas. Fuente propia

8.3.5. Módulo de juego

El Módulo de Juego tiene como objetivo central proporcionar una experiencia interactiva y envolvente para los usuarios mientras exploran y participan en desafíos dentro del mundo de Melody. Este módulo se enfoca en la representación dinámica de los distintos desafíos, niveles y elementos del juego. Además, permitirá la creación, actualización y eliminación de desafíos y niveles, gestionando esta información a través de configuraciones basadas en ScriptableObjects.

Al igual que en todos los módulos de este proyecto, se ha aplicado el patrón de diseño Strategy para definir estrategias flexibles de obtención de datos, integrando servicios como Unity Services y Firebase, y permitiendo la incorporación de nuevas estrategias según sea necesario.

Asimismo, se empleó el patrón de diseño Factory para la creación de objetos de obtención de datos, actuando como intermediario entre el cliente y los distintos servicios, garantizando un bajo acoplamiento y flexibilidad en la integración de nuevos servicios.

La integración de servicios con la interfaz visual sigue el enfoque del patrón MVP (Model-View-Presenter), adecuado para proyectos en Unity, proporcionando una estructura eficaz para la comunicación entre modelos, vistas y servicios.

En el diagrama de contexto que se observa en la Figura 38, se pueden visualizar todos los componentes asociados con los diversos procesos, actores y tecnologías que se emplearán en este módulo. Para obtener una comprensión más detallada del diseño de la arquitectura de este módulo, se recomienda consultar el Anexo 9, donde se incluyen todos los diagramas necesarios.

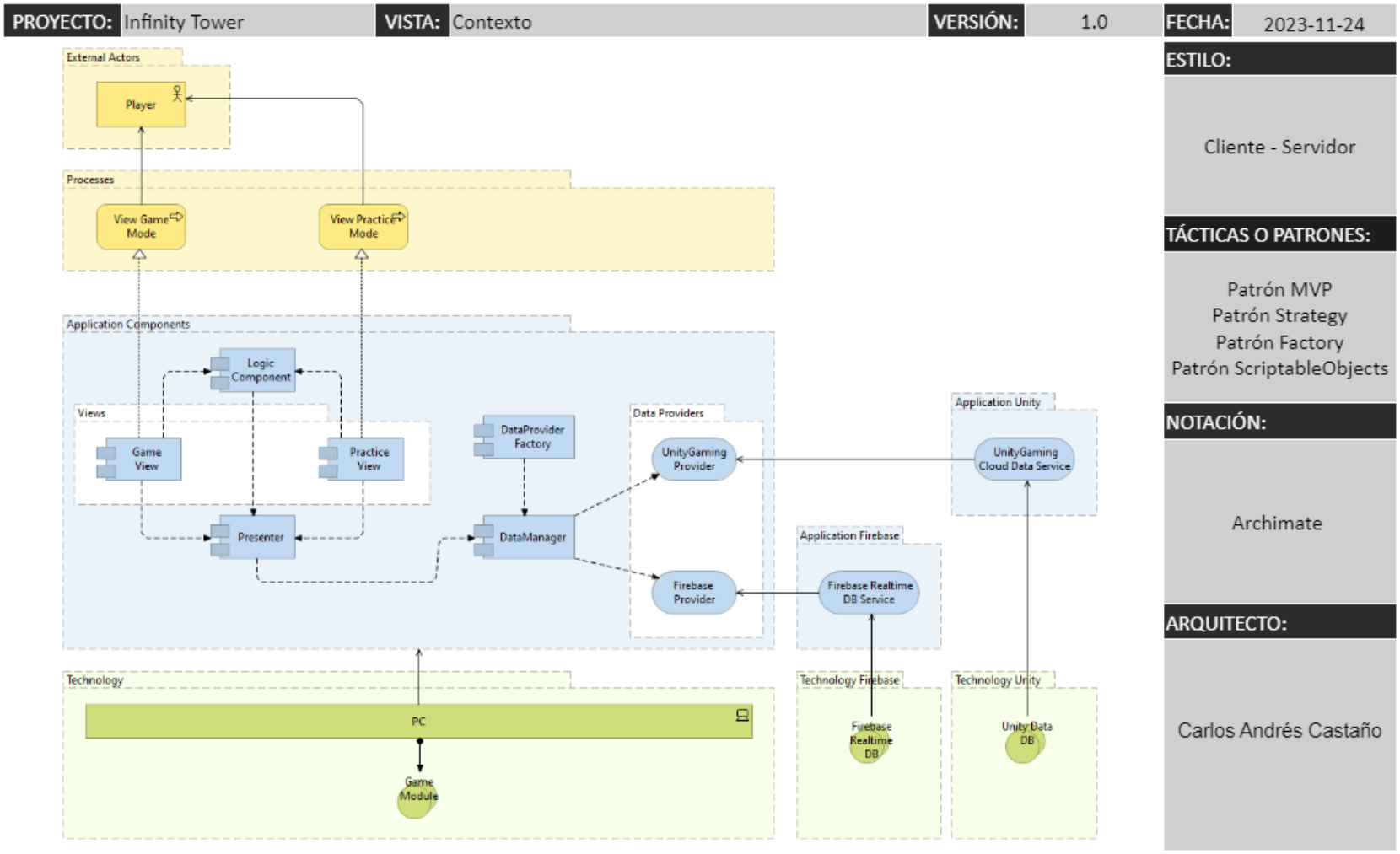


Figura 38: Diagrama de contexto módulo de juego. Fuente propia

Hasta este punto, podríamos concluir que los patrones utilizados no presentan novedades. No obstante, para el desarrollo de este módulo, se hace imprescindible incorporar ScriptableObjects como un patrón fundamental para potenciar la eficiencia y flexibilidad del sistema. Detallaré esto más exhaustivamente en la sección de implementación de este módulo.

8.4. Implementación del prototipo

8.4.1. Estructura del proyecto

Vamos a detallar la estructura del software para facilitar la comprensión. En la sección de la estructura del proyecto, encontramos los diversos módulos del juego. Los módulos que hemos desarrollado incluyen AuthModule, CoreModule, GameModule, MapsModule y MenuModule.

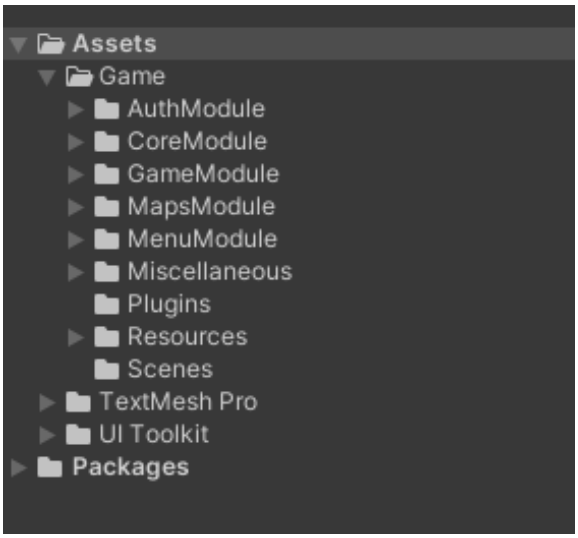


Figura 39: Estructura del proyecto. Fuente propia

Dentro de la carpeta "Miscellaneous", se encuentran las fuentes de texto, mientras que en la sección "Plugins"se alberga la biblioteca que estamos utilizando para la entrada de dispositivos MIDI. En este caso, la librería en uso se denomina "DryWetMidi".

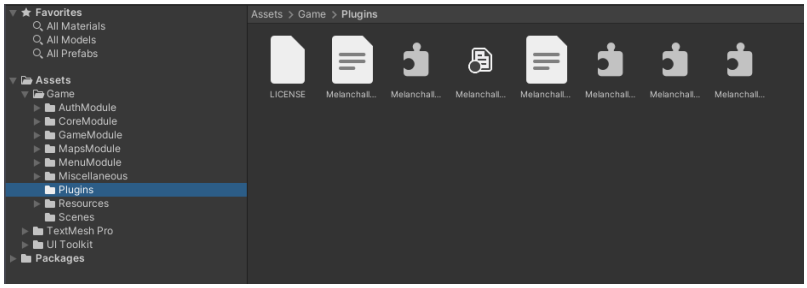


Figura 40: Plugins del proyecto. Fuente propia

Adicionalmente, en la carpeta Resources"se encuentran los elementos relacionados con audio, configuraciones, imágenes, iconos y algunos de los ScriptableObjects, los cuales exploraremos más detalladamente más adelante.

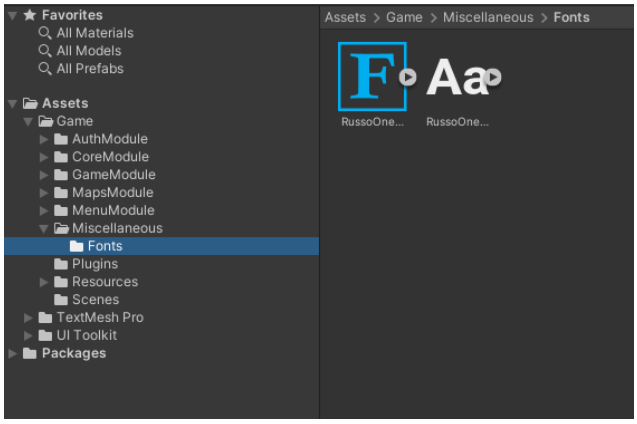
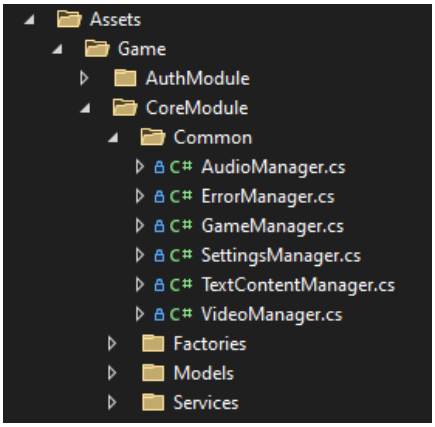


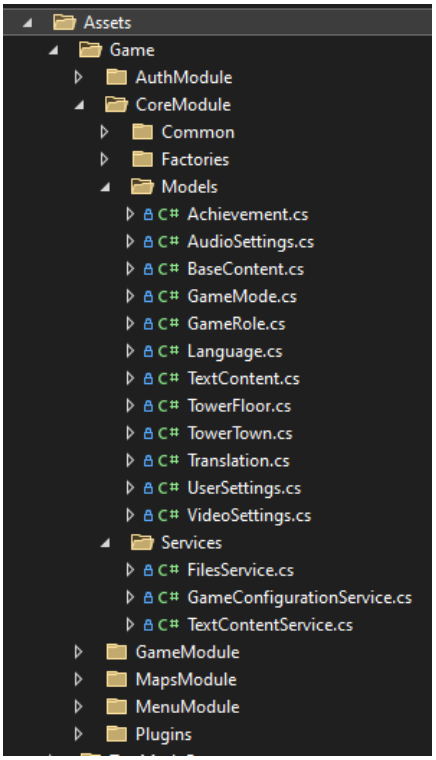
Figura 41: Recursos del proyecto. Fuente propia

8.4.2. Módulo core

En el CoreModule se encuentran los elementos comunes y base del juego. Al diseñar la arquitectura, especialmente al trabajar en el MenuModule después del AuthModule, noté la necesidad de compartir ciertos componentes. Este módulo alberga el AudioManager, ErrorManager, GameManager, SettingsManager, TextContentManager y VideoManager. Todos estos managers siguen el patrón singleton para asegurar una única instancia en el juego, ya que son esenciales en prácticamente todos los módulos.



(a) Managers módulo core.



(b) Servicios y Modelos del módulo core.

Figura 42: Estructura módulo core. Fuente propia

En la sección del editor, se incluye el código utilizado para realizar configuraciones dentro de Unity. Para la gestión de contenidos del juego, se optó por crearlos directa-

mente en Unity para facilitar y minimizar el consumo, ya que no justificaba el uso de servicios externos para este propósito. Por ejemplo, en el editor, se pueden realizar configuraciones del juego, como crear, actualizar y eliminar modos de juego. Aunque algunos modos, como el tutorial y el modo Band o Party, están deshabilitados por el momento, están configurados con nombres y descripciones en varios idiomas.

Las configuraciones incluyen información sobre roles (pianista, guitarrista, cantante), niveles, desafíos, idiomas y logros, cada uno con sus traducciones correspondientes. Cabe destacar que, aunque se han creado múltiples niveles y desafíos, actualmente solo se utiliza el primer nivel en el prototipo.

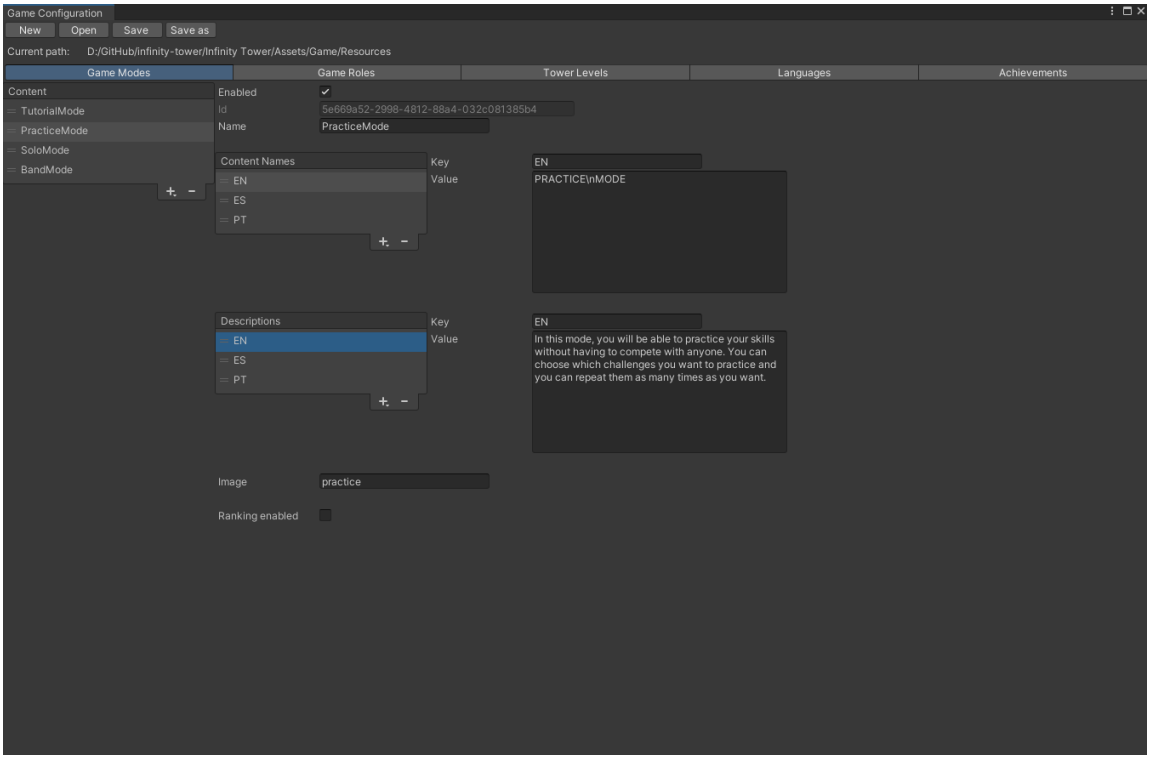


Figura 43: Contenidos módulo core en Unity Editor. Fuente propia

Los idiomas admitidos son inglés, español y portugués. La sección de Achievements

maneja contenido con sus respectivas traducciones. Es importante señalar que cualquier modificación en estas configuraciones requiere una nueva versión del juego (Release) para que los usuarios dispongan de los nuevos contenidos.

El único servicio externo utilizado es Unity Game Services, que almacena datos del usuario, como UserInfo (nombre de usuario, correo electrónico) y GameResults (resultados y logros desbloqueados). Esta información se almacena en Unity Cloud.

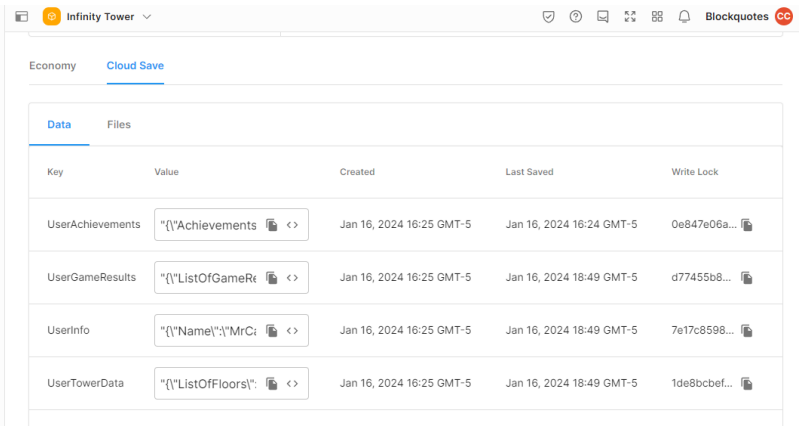


Figura 44: Datos en Unity Cloud. Fuente propia

En cuanto al TextContent, se encarga específicamente de los textos del juego, como botones, títulos y diálogos, con sus respectivas traducciones. En el CoreModule, no se utilizan servicios externos, sino servicios de lectura y escritura de archivos para gestionar las configuraciones locales del usuario.

8.4.3. Módulo de autenticación

En el AuthModule, siguiendo la arquitectura patrón MVP (Modelo, Vista y Presentador), tal como se ilustra en la Figura 45. En los modelos, encontramos únicamente AuthUser. En cuanto a los presentadores, estas clases se encargan de dirigir las vis-

tas. Las vistas comprenden el LoginScreen, el RegisterScreen y el RoleScreen, este último redirige al MenuModule después del registro.

El DefaultScreen es esencialmente la pantalla inicial que nos indica si deseamos iniciar sesión o registrarnos. Dependiendo de la elección, se dirige a una de estas dos vistas. El LoginScreen, tras iniciar sesión, nos lleva directamente al MenuModule, mientras que el registro nos lleva al RoleScreen. En este último, se elige el rol que se utilizará en el juego, y una vez seleccionado, también nos redirige al MenuModule.

Además, se han incorporado con éxito los patrones Factory y Strategy para la inclusión de diversos servicios de autenticación con el módulo. Esta elección se fundamenta en la necesidad de contar con una estructura flexible que permita al módulo cambiar sin dificultades el proveedor de autenticación.

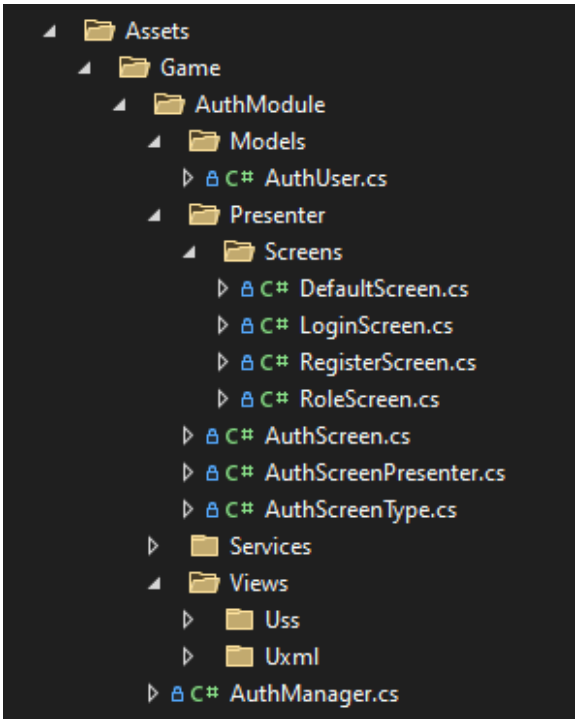


Figura 45: Estructura módulo de autenticación. Fuente propia

Para llevar a cabo este proceso, simplemente necesitamos crear un nuevo directorio dentro de **"Services"** con el nombre del nuevo proveedor de autenticación. En este ejemplo, realizaremos el ejercicio con Firebase. Una vez creado, procedemos a generar una nueva clase descriptiva del nuevo servicio a implementar, como por ejemplo **"FirebaseAuthService"**. Esta nueva clase debe implementar la interfaz **"IAuthProvider"**, la cual proporciona todos los métodos necesarios para el correcto funcionamiento del módulo. Basta con revisar la documentación del servicio, e implementar la lógica necesaria dentro de cada uno de estos métodos para garantizar su correcto funcionamiento.

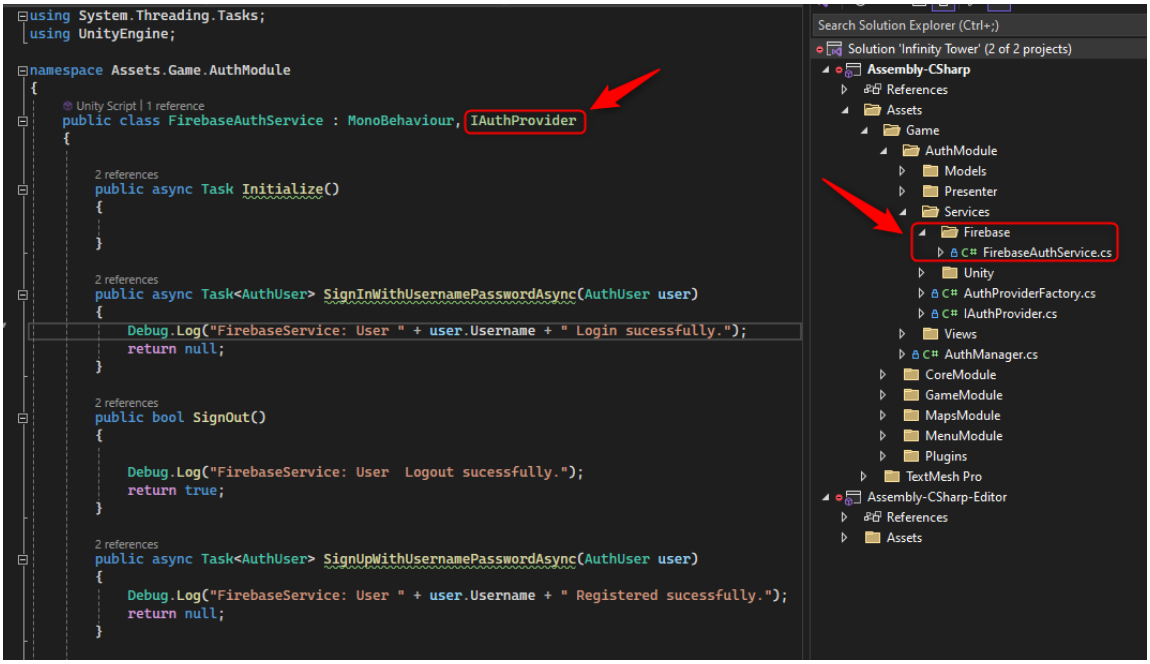


Figura 46: Implementación nuevo servicio de autenticación. Fuente propia

8.4.4. Módulo de menus

En el desarrollo del MenuModule, que fue la siguiente etapa en el proceso, también se implementó el patrón MVP manteniendo una estructura clara que se evidencia en

la Figura 47. En todos los módulos, excepto el CoreModule, se aplicó este patrón. En este contexto, los modelos incluyen UserInfo y UserAchievements, al igual que el presentador y las vistas. En comparación con otros módulos, el MenuModule presenta una variedad más extensa de vistas, que comprenden PlayScreen, ProfileScreen, RankingScreen, SettingsScreen y AchievementsScreen. Además, se utilizaron objetos reutilizables entre estas diversas vistas.

En cuanto a los servicios, en este caso, el UnityService se encarga de la conexión con UnityCloud para almacenar y recuperar la información del usuario. Por ejemplo, en la sección de perfil, los usuarios pueden editar su información, y este servicio se utiliza para guardar dicha información en UnityCloud.

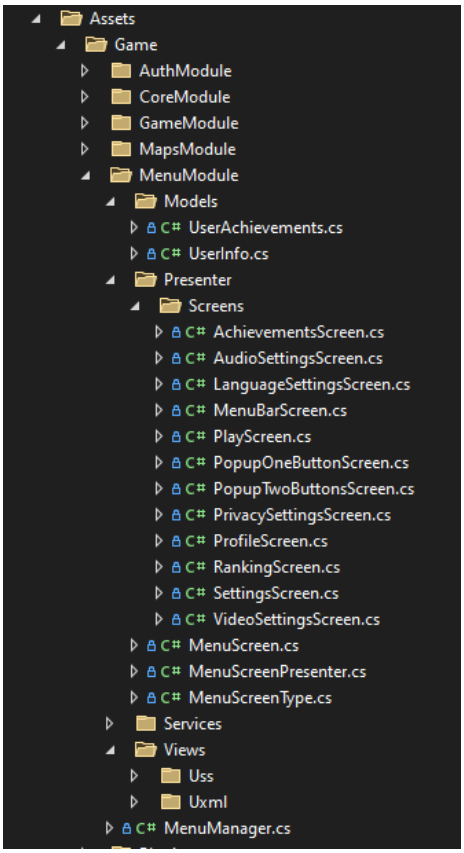


Figura 47: Estructura módulo de menus. Fuente propia

Al igual que en los otros módulos, simplemente necesitamos crear un nuevo directorio dentro de **"Services"** con el nombre del nuevo proveedor de datos. En este ejemplo, realizaremos el ejercicio con Firebase. Una vez creado, procedemos a generar una nueva clase descriptiva del nuevo servicio a implementar, como por ejemplo **"FirebaseDataService"**. Esta nueva clase debe implementar la interfaz **"IDataProvider"**, la cual proporciona todos los métodos necesarios para el correcto funcionamiento del módulo. Basta con revisar la documentación del servicio, e implementar la lógica necesaria dentro de cada uno de estos métodos para garantizar su correcto funcionamiento.

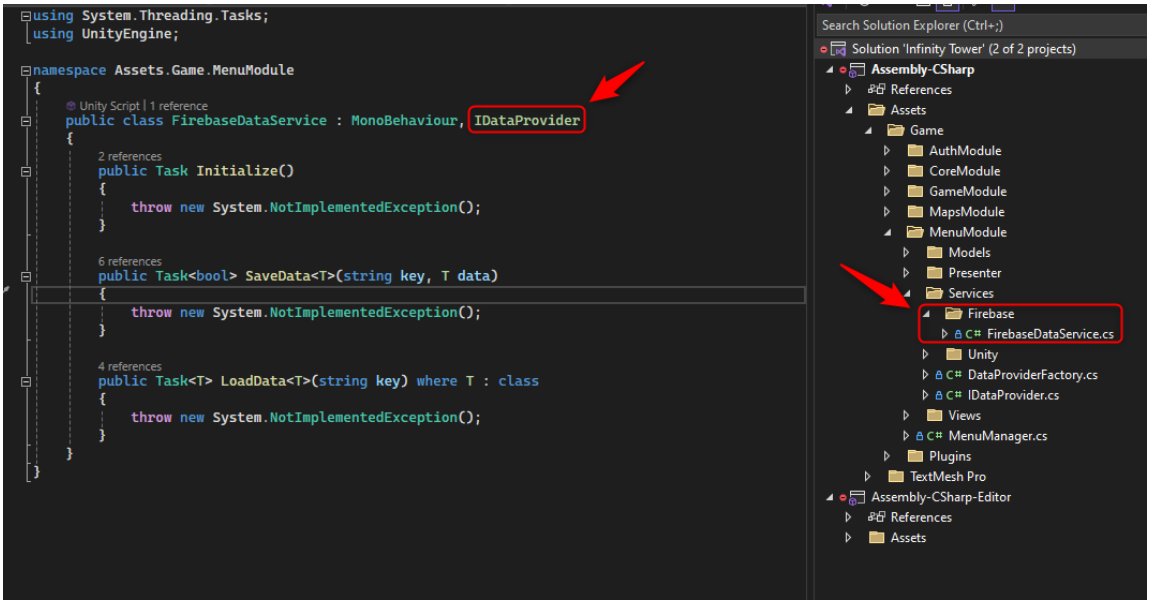


Figura 48: Implementación nuevo servicio de datos en módulo de menus. Fuente propia

8.4.5. Módulo de mapas

En el MapsModule, que es el siguiente módulo en consideración, se sigue el mismo patrón MVP manteniendo una estructura clara que se puede apreciar en la Figura 49. Los módulos comparten una estructura similar, y lo que varía son las vistas. En este

caso, contamos con dos vistas: la TowerScreen y la FloorScreen. En la TowerScreen se presentan los diferentes pisos, mientras que en la FloorScreen se exhiben los distintos pueblos o desafíos.

En cuanto a los servicios, nuevamente se utiliza Unity. En este escenario, el servicio se encarga de la lectura de los datos de configuración de la Torre, haciendo uso del modelo UserTowerData.

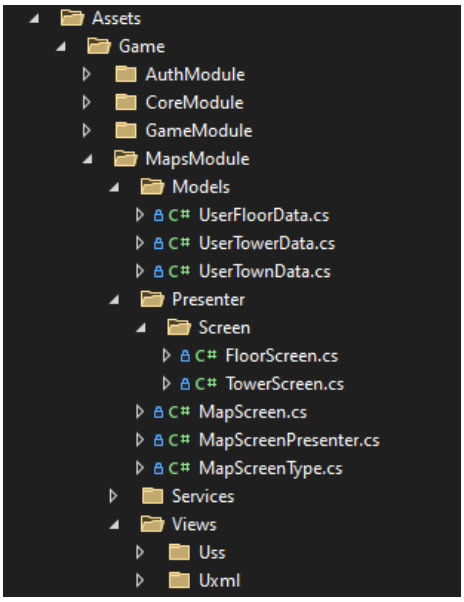


Figura 49: Estructura módulo de mapas. Fuente propia

De la misma forma que en los otros módulos, necesitamos crear un nuevo directorio dentro de **"Services"** con el nombre del nuevo proveedor de datos. En este ejemplo, realizaremos el ejercicio con Firebase. Una vez creado, procedemos a generar una nueva clase descriptiva del nuevo servicio a implementar, como por ejemplo **"FirebaseDataService"**. Esta nueva clase debe implementar la interfaz **"IDataProvider"**, la cual proporciona todos los métodos necesarios para el correcto funcionamiento del módulo. Basta con revisar la documentación del servicio, e implementar la lógica necesaria dentro de cada uno de estos métodos para garantizar su correcto funciona-

miento.

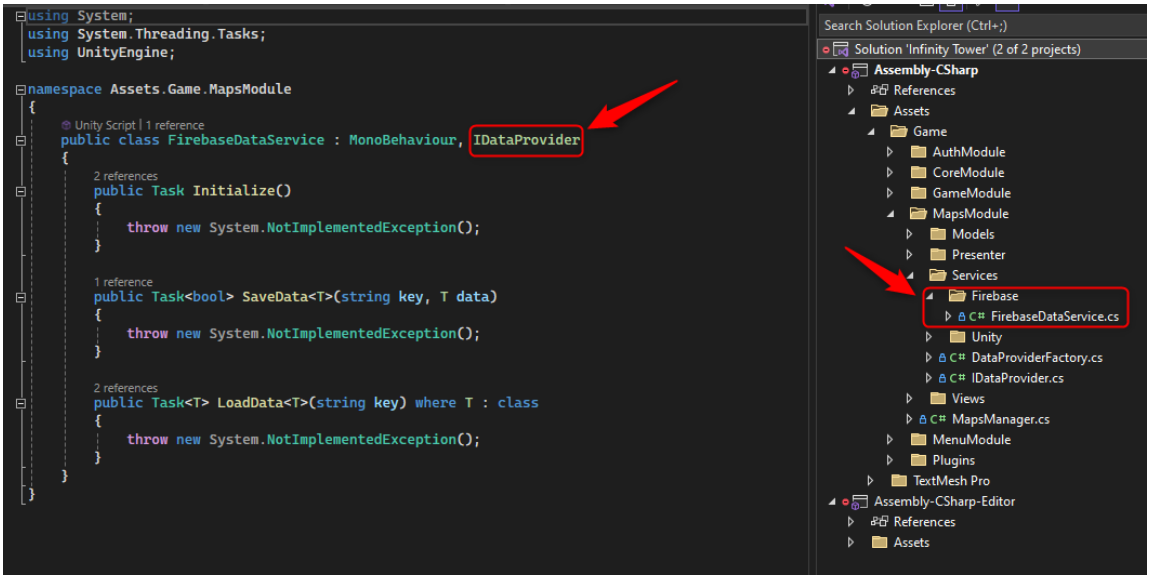


Figura 50: Implementación nuevo servicio de datos en módulo de mapas. Fuente propia

8.4.6. Módulo de juego

Finalmente, el módulo de juego se podría considerar como el más complejo de todos. En esencia, sigue un patrón MVP para la interfaz gráfica manteniendo una estructura clara que se puede apreciar en la Figura 51, especialmente en la parte interactiva que incluye elementos como botones, porcentajes, puntajes, textos, entre otros. Sin embargo, la gestión de las pistas de notas musicales y de los periféricos de entrada para tocarlas no sigue estrictamente el patrón MVP debido a la simplicidad y eficiencia al trabajar con las físicas y recursos ofrecidos por Unity.

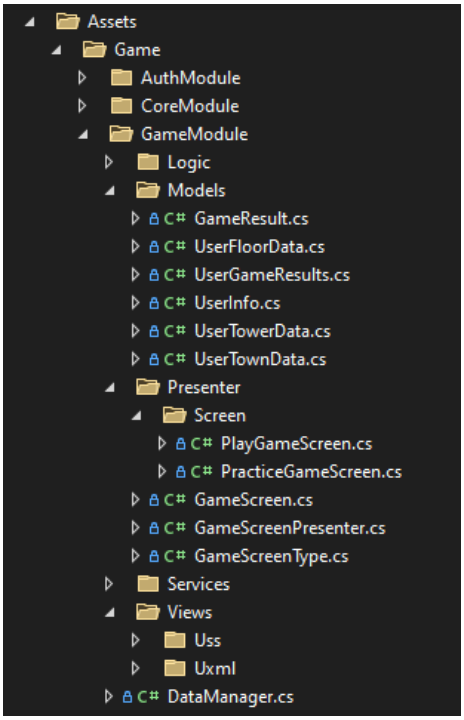


Figura 51: Estructura módulo de juego. Fuente propia

En este módulo, contamos con una estructura basada en el patrón MVP, representada por el GamePresenter, y, por otro lado, la parte relacionada con el motor de Unity, representada por el GameCore, donde se configuran las físicas y otros aspectos. Ambos se conectan mediante eventos para cambiar puntajes y otros elementos interactivos del módulo.

Dado que la idea central del juego es permitir la inclusión de diferentes instrumentos, se ha utilizado Prefabs para incorporar elementos físicos del juego. Estos Prefabs son archivos predefinidos con una base que puede tener variantes con las mismas funcionalidades básicas. Por ejemplo, el Prefab llamado GameCore maneja la lógica de físicas, límites y la posibilidad de añadir una cantidad específica de pistas o notas.

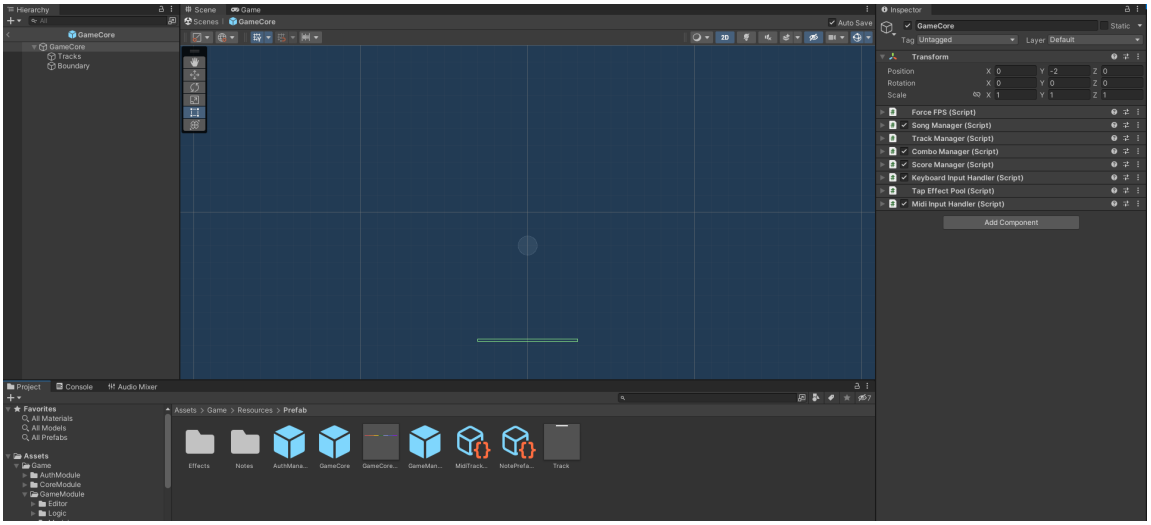


Figura 52: Prefab base GameCore. Fuente propia

Se ha creado una variante del GameCore, llamada GameCore7Tracks, a la cual se le han añadido 7 pistas (notas), permitiendo la creación de diversas variantes de este objeto con más pistas para incluir diferentes instrumentos o modos de juego. Además, los Tracks también son Prefabs con lógica de físicas y eventos para la detección en el presenter del juego.

Por ejemplo, cada Track cuenta con su LineArea (límite donde se puede tocar una nota), NoteArea (parte donde se debe tocar la nota) y TriggerArea (parte donde se detecta si la nota se tocó correctamente o no).

Para el prototipo, se ha creado la variante GameCore7Tracks con las notas principales (Do, Re, Mi, Fa, Sol, La y Si). Sin embargo, se podría desarrollar variantes adicionales para incluir sostenidos y bemoles, agregando 5 pistas adicionales, lo que daría una octava completa. Este enfoque se considera viable para desarrollos futuros.

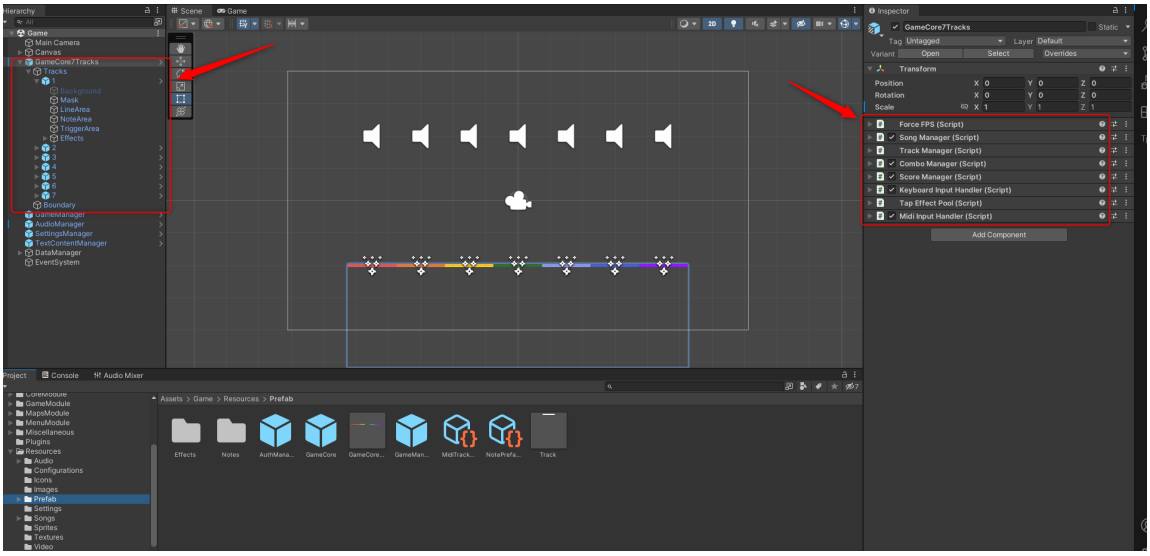


Figura 53: Variante GameCore implementación de pista de notas en el módulo de juego. Fuente propia

En la Figura 53, se presentan los objetos que actúan como elementos físicos en la escena del juego. En el lado izquierdo, se pueden identificar elementos como Tracks, NoteArea, TriggerArea, LineArea, entre otros. Todos estos objetos cuentan con propiedades físicas, detalladas en la Figura 54, que son esenciales para la detección de colisiones y otros eventos a través del motor Unity. Estos eventos son gestionados por los diferentes Managers que se observan en el lado derecho de la Figura 53.

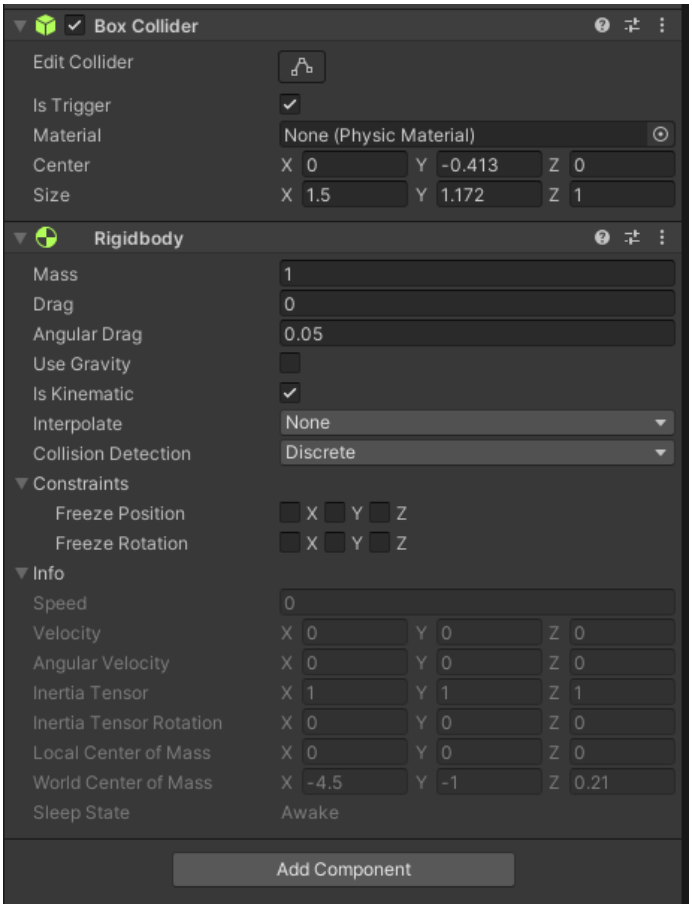


Figura 54: Propiedades físicas de objetos - Unity. Fuente propia

Al igual que en los otros módulos, necesitamos crear un nuevo directorio dentro de **"Services"** con el nombre del nuevo proveedor de datos. En este ejemplo, realizaremos el ejercicio con Firebase. Una vez creado, procedemos a generar una nueva clase descriptiva del nuevo servicio a implementar, como por ejemplo **"FirebaseDataService"**. Esta nueva clase debe implementar la interfaz **"IDataProvider"**, la cual proporciona todos los métodos necesarios para el correcto funcionamiento del módulo. Basta con revisar la documentación del servicio, e implementar la lógica necesaria dentro de cada uno de estos métodos para garantizar su correcto funcionamiento.

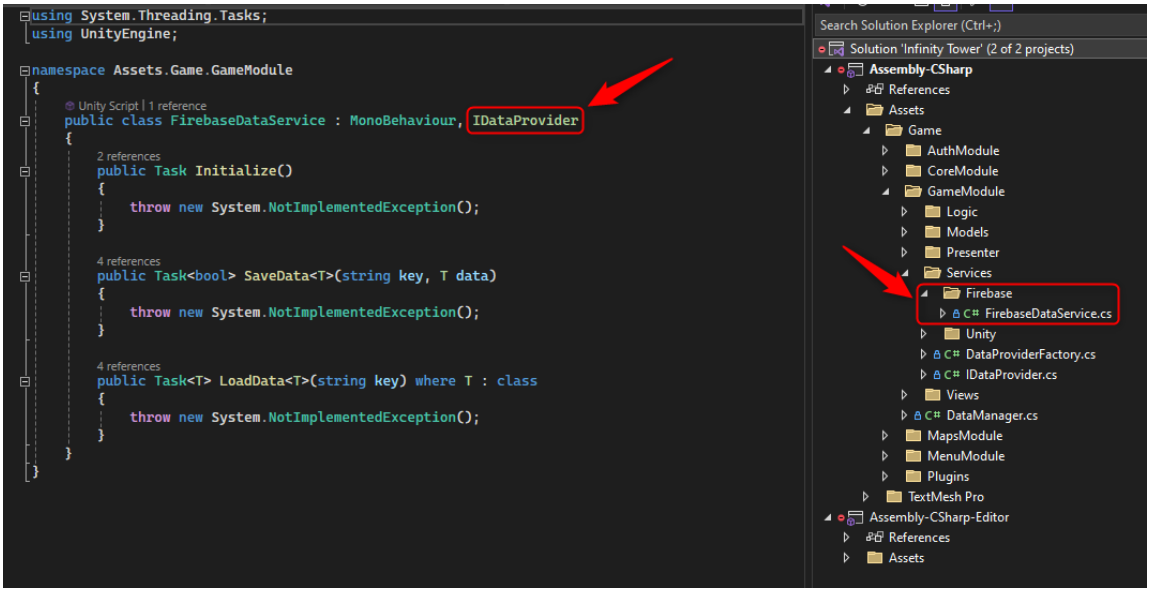


Figura 55: Implementación nuevo servicio de datos en módulo de juego. Fuente propia

Finalmente, se utilizan ScriptableObjects para generar objetos JSON con datos que permiten representar las notas musicales como un mapa de una canción. Estos objetos permiten mapear las notas según una partitura. Para este prototipo, se han seleccionado partituras del método Suzuki tomadas de (Suzuki, 1995), enfocado en la precisión y repetición. Estas notas se han plasmado en el ScriptableObject, considerando solo las notas principales de una sola octava en este prototipo. La flexibilidad del enfoque permite decidir cuántas notas se quieren pintar, establecer el tiempo de la nota (blanca, negra, redonda, etc.), y brinda oportunidades para ajustes y mejoras en el desarrollo futuro.

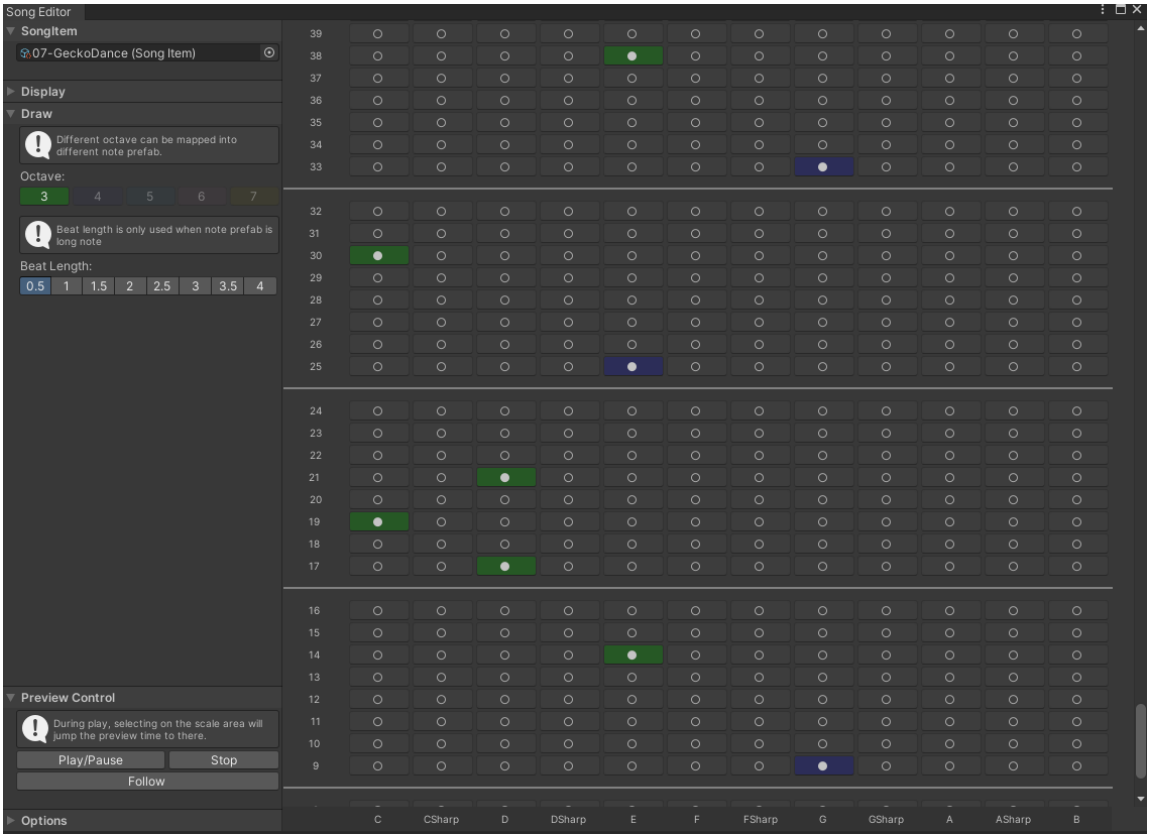


Figura 56: Mapeo de ScriptableObject en Editor de Unity. Fuente propia

8.5. Resultados del prototipo

Durante un período de prueba de 4 días, un sujeto de prueba utilizó el prototipo del juego, dedicando 15 minutos diarios para intentar superar los desafíos o niveles. La expectativa era observar un avance a lo largo de los 10 niveles creados para este prototipo y detectar alguna mejora en la precisión del sujeto a medida que se familiarizaba con el juego.

Al revisar los resultados de los niveles 1 al 5, se notó que el sujeto dominó estos

niveles con facilidad, completándolos en el primer o segundo intento con una tasa de éxito cercana al 100 %. Por lo cual, Decidí centrarme solamente en los niveles del 6 al 9, donde la complejidad era un poco mas compleja y se esperaba un mayor desafío.

Los datos recopilados revelan un progreso en la precisión del sujeto a lo largo de estos niveles superiores. A raíz de estos datos, he generado un conjunto de gráficas que detallan los intentos realizados en cada nivel junto con la precisión correspondiente. La tendencia claramente ascendente en la línea trazada a través de estos puntos indica una mejora constante en la precisión a lo largo de los 4 días de prueba.

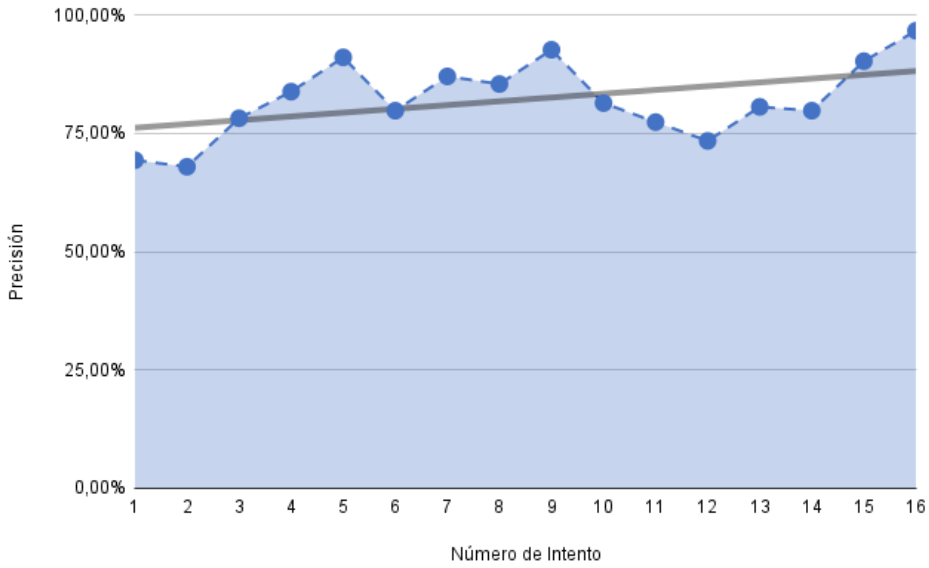


Figura 57: Progreso - Nivel 6. Fuente propia

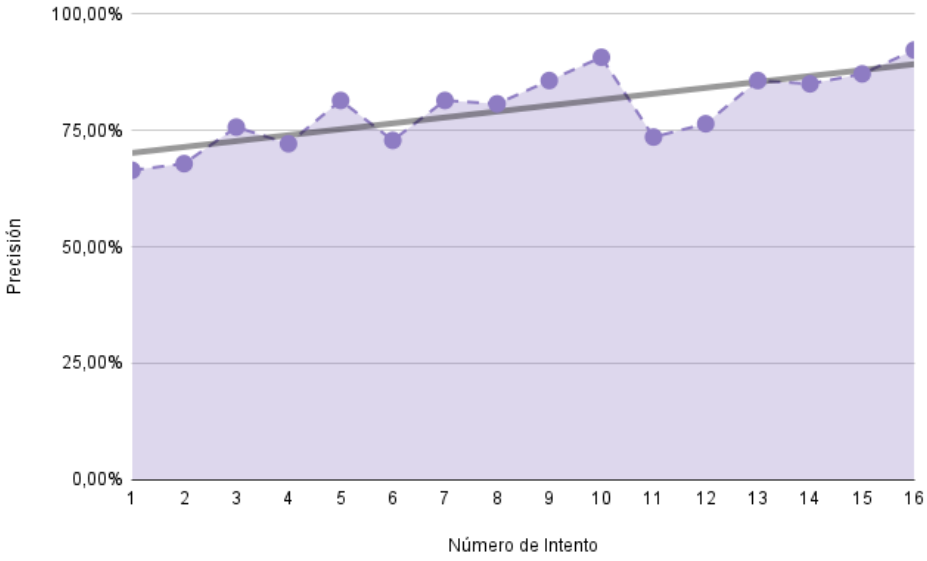


Figura 58: Progreso - Nivel 7. Fuente propia

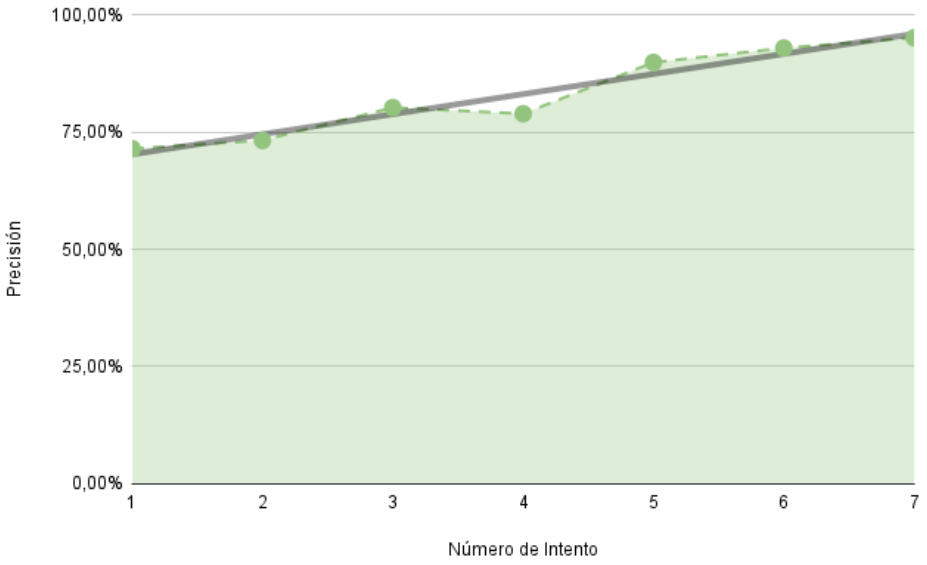


Figura 59: Progreso - Nivel 8. Fuente propia

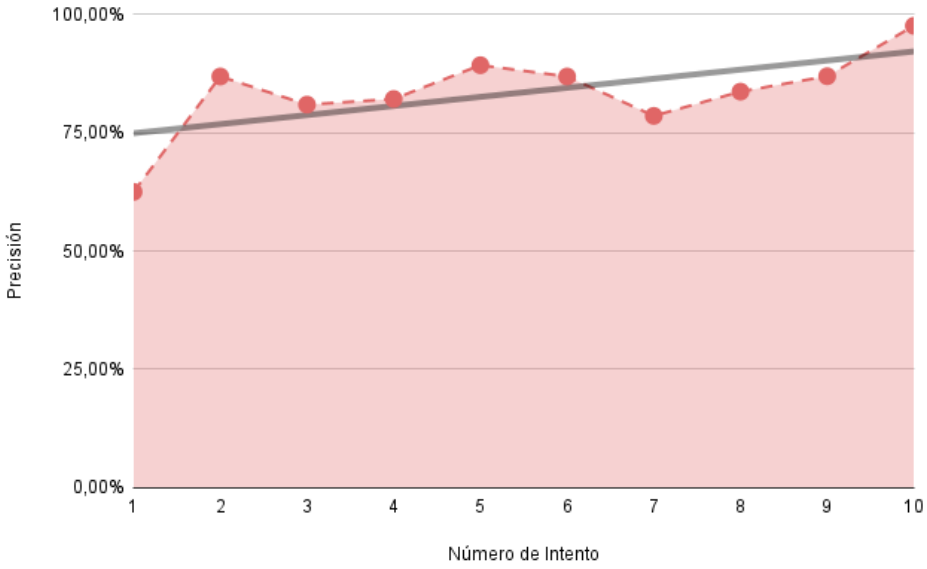


Figura 60: Progreso - Nivel 9. Fuente propia

Es importante destacar que, debido al tamaño de la muestra, aunque el sujeto mostró una mejora significativa en la precisión y capacidad para superar niveles más complejos, no se pueden generalizar estos hallazgos para afirmar la efectividad del prototipo sin realizar estudios adicionales con una muestra más amplia y diversa. Por lo tanto, estos resultados son preliminares pero prometedores, sugiriendo que el prototipo tiene potencial como herramienta para mejorar habilidades específicas en un contexto controlado.

9. Conclusiones

La arquitectura de software diseñada para este proyecto ha sido concebida con un enfoque modular, adoptando la filosofía de que cada módulo fuese como una pieza de Lego intercambiable. Este enfoque modular proporciona flexibilidad y reusabilidad, permitiendo que cada componente sea reemplazado o extraído sin afectar el conjunto, facilitando así la adaptación de estos módulos para futuros proyectos sin muchos inconvenientes.

La aplicación de patrones de diseño como el Modelo-Vista-Presentador (MVP), el Strategy y el Factory han contribuido a una estructuración eficiente de los diferentes componentes del software. El patrón MVP ha permitido una clara separación de responsabilidades entre la lógica de presentación y los datos, la Strategy ha posibilitado la definición de algoritmos intercambiables, y el Factory ha facilitado la creación de objetos sin acoplamientos innecesarios.

Además, la adhesión a las buenas prácticas de desarrollo, particularmente los principios SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion), ha contribuido a la creación de un código robusto y mantenible. La implementación de estos principios ha promovido la cohesión y la baja dependencia entre los componentes, lo que a su vez ha resultado en un software más fácil de entender, modificar y escalar.

En retrospectiva, debo reconocer que al comenzar con el diseño y desarrollo de los módulos, comenzar con el módulo de juego en una fase inicial del proyecto, podría haber generado mayor valor al proyecto, llevando a cabo más pruebas con usuarios desde el principio. Este enfoque habría proporcionado valiosos comentarios y datos desde las etapas iniciales, permitiendo ajustes continuos y mejoras iterativas.

Las pruebas realizadas con un usuario se centraron en evaluar mejoras en la precisión

rítmica y la velocidad de interpretación musical utilizando el prototipo desarrollado. Los resultados obtenidos indicaron una tendencia positiva en la precisión rítmica del sujeto de prueba. Aunque estos hallazgos son alentadores y sugieren que el prototipo podría ser efectivo para mejorar habilidades musicales específicas, es importante señalar que estos resultados son preliminares y provienen de una muestra muy limitada. Por lo tanto, no se pueden hacer afirmaciones definitivas sobre la efectividad general del prototipo sin estudios adicionales que incluyan un número mayor y más diverso de participantes. Estos resultados preliminares son, sin embargo, un indicativo prometedor del potencial del prototipo para fomentar mejoras tangibles en la precisión musical y la velocidad de ejecución en un entorno controlado.

En resumen, el proyecto ha alcanzado sus metas establecidas, demostrando no solo la viabilidad del prototipo del juego, sino también su capacidad para contribuir al aprendizaje musical de manera efectiva.

10. Anexos

1. Diseño de la encuesta
2. Resultados Encuesta
3. Documento de diseño del juego
4. Documento de diseño de la interfaz gráfica
5. ADD - CoreModule
6. ADD - AuthModule
7. ADD - MenuModule
8. ADD - MapsModule
9. ADD - GameModule
10. Metodologías musicales

11. Referencias Bibliográficas

Referencias

Fontelles Rodríguez, V. L. (2019). Metodologías musicales. siglos xx y xxi.

Keith, C. (2010). *Agile Game Development with Scrum*. Addison-Wesley.

Lara Rodriguez, F. J. (2017). El video juego synthesia: una herramienta didactica para el aprendizaje músical. Master's thesis, Pontificia Universidad Javeriana.

Lierse, S. (2010). The kodaly method in the twenty first century. *Bulletin of the Transilvannia University of Brasov*, 3.

Newzoo (2023). A new era of engagement in media & entertainment. Technical report, Newzoo.

Schell, J. (2008). *The Art of Game Design: A Book of Lenses*. Elsevier.

Simply, L. (2023). Bring creativity and fun into your home. https://www.hellosimply.com/ [Accessed: (09-06-2023)].

Sing Sharp, L. (2023). Sing sharp, learn to sing. https://www.singsharp.com/ [Accessed: (09-06-2023)].

Suzuki, S. (1995). *Suzuki Piano School*. Summy-Birchard.

Synthesia, L. (2023). A fun way to learn how to play the piano. https://synthesiagame.com/ [Accessed: (14-06-2023)].

Sánchez Rueda, O. R. (2017). Diseño e implementación de una arquitectura de software con soporte para dispositivos móviles que apoye la enseñanza musical en niños. Master's thesis, Pontificia Universidad Javeriana.

Unity (2021a). *Level up your code with game programming patterns*. Unity, lts edition edition.

Unity (2021b). *User interface design and implementation in Unity*. Unity, lts edition edition.

Yousician, L. (2023). Learn to play music at home, at your own pace. https://yousician.com/ [Accessed: (09-06-2023)].

12. Glosario de Términos

Infinity Tower Es el nombre del videojuego en desarrollo que tiene como objetivo combinar las aventuras de los juegos de rol y aprendizaje musical.

Videojuego Es una aplicación orientada al entretenimiento interactivo que combina elementos visuales, auditivos y jugables para crear una experiencia virtual. Este tiene como objetivo brindar actividades de resolución de problemas, en la cual los jugadores ingresan de manera voluntaria para satisfacer su curiosidad y buscar placer a través de la manipulación de un entorno virtual.

Aprendizaje musical El proceso de adquirir conocimientos y habilidades relacionadas con la música, como la teoría musical, el manejo de instrumentos y la interpretación de piezas musicales.

Habilidades musicales Competencias relacionadas con la interpretación, composición o apreciación musical que el jugador puede desarrollar y mejorar a través de la interacción con los instrumentos musicales en el juego.

Instrumentos reales Instrumentos musicales físicos, como guitarras, pianos o la voz, que se utilizan en el mundo real y pueden ser integrados en el videojuego para interactuar con la música y el juego.

Interfaz de usuario La forma en que los jugadores interactúan con el videojuego, a través de elementos visuales y controles diseñados para facilitar la experiencia del usuario.

Arquitectura de software El diseño estructural y funcional del software que define cómo los diferentes componentes y módulos del juego interactúan entre sí.

Desafíos musicales Secciones o pruebas dentro del juego que requieren que el jugador demuestre su habilidad musical, como tocar notas en el momento adecuado, seguir ritmos o realizar composiciones musicales.

Sonido Es la sensación o efecto que produce en el oído, el movimiento vibratorio de un cuerpo. Las cualidades del sonido son cuatro y nos ayudan a describir el mismo.

Ritmo Es todo aquello que pertenece al movimiento que impulsa a la música en el tiempo.

Solfeo Es un método de enseñanza musical enfocado a la lectura, escritura y reconocimiento de notas musicales a través de símbolos.

Fonomímia Método de enseñanza musical en el que cada nota está asociada a un determinado gesto realizado con la mano.

Clase Es una estructura que define las propiedades y comportamientos de un objeto.

Objeto Es una instancia particular de una clase. Representa una entidad con características específicas y tiene su propio estado.

Encapsulación Es la capacidad que tiene un objeto de esconder partes de su estado y comportamiento de otros objetos, exponiendo únicamente una interfaz limitada al resto del programa.

Abstracción Es el modelo de un objeto o fenómeno del mundo real, limitado a un contexto específico, que representa todos los datos relevantes a este contexto con gran precisión, omitiendo el resto.

Herencia Es la capacidad de crear nuevas clases sobre otras existentes.

Interfaz Es un conjunto de métodos y/o propiedades que define el comportamiento y la comunicación que un objeto puede tener con el mundo exterior. Sirve como un contrato o especificación de los métodos que una clase debe implementar, sin definir la implementación concreta de dichos métodos.

Patrón Es una solución probada y reutilizable para un problema común en el diseño de software. Proporciona un enfoque estructurado y pautas para resolver de manera efectiva situaciones similares en diferentes contextos.

Acoplamiento Se refiere al grado de dependencia entre dos componentes o módulos. Un acoplamiento fuerte significa que los cambios en un componente pueden afectar a otros, mientras que un acoplamiento débil indica una menor dependencia.

Cohesión Se refiere al grado en que los elementos dentro de un componente o módulo están relacionados y se centran en una única tarea o responsabilidad. Una alta cohesión implica que los elementos están estrechamente relacionados y trabajan juntos de manera eficiente.

Flexibilidad Se refiere a la capacidad de un sistema, componente o diseño para adaptarse y responder de manera efectiva a cambios, sin generar efectos no deseados o costos excesivos.

Editor de Unity El Editor de Unity es la interfaz gráfica de usuario que proporciona el entorno de desarrollo para crear y editar proyectos en Unity. Este editor es una herramienta integral que permite a los desarrolladores y diseñadores trabajar en la construcción de videojuegos y aplicaciones interactivas de manera visual.