PLUGIN DINÁMICO PARA FMOD

DYNAMIC PLUGIN FOR FMOD

TRABAJO FIN DE GRADO
CURSO 2023-2024

AUTOR
MARCO IVÁN MERINO HERNÁNDEZ

DIRECTOR
JAIME SÁNCHEZ HERNÁNDEZ

GRADO EN DESARROLLO DE VIDEOJUEGOS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

PLUGIN DINÁMICO PARA FMOD
DYNAMIC PLUGIN FOR FMOD

TRABAJO DE FIN DE GRADO EN DESARROLLO DE VIDEOJUEGOS

AUTOR
MARCO IVÁN MERINO HERNÁNDEZ

DIRECTOR
JAIME SÁNCHEZ HERNÁNDEZ

CONVOCATORIA: SEPTIEMBRE 2024

GRADO EN DESARROLLO DE VIDEOJUEGOS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

03 DE SEPTIEMBRE DE 2024

AGRADECIMIENTOS

A Pablo, por iniciarme en el maravilloso mundo de los objetos ScriptableObject y abrirme la mente a nuevas ideas en el tratamiento de datos.

A Daniel por servirme de escudero con las dudas de accesibilidad de usuario.

A Jaime, mi tutor, por su templanza ante las biblias de mis correos y mis inquietudes. Por su ayuda y su carisma en momentos tan delicados, siempre tan amable. Y como no, por sus consejos tan valiosos para guiar el proyecto por buen camino.

A mi familia, por hacerlo siempre lo mejor posible y por darme las mejores lecciones de vida.

Gracias a todos aquellos que, conscientes o no, han sido partícipes de este proyecto, aguantándome, escuchándome, aportándome ideas, aunque no se entendiera de qué hablara, con el simple fin de intentar ayudarme.

DFP me ha aportado momentos de felicidad, de curiosidad, de impaciencia, de inquietud, de frustración y también de ganas de querer afrontarlo. Con él se va un pedacito de mi alma, de mi experiencia, de mi tiempo, mis lágrimas, se lleva parte de mi superación, de afrontar miedos y de superar adversidades. Espero haber podido trasmitir todo ello en el documento y que te permita apreciarlo, querido lector.

Y recuerda, ante la adversidad, ante la incertidumbre, ante las dudas, ante el miedo, no te rindas, no decaigas, no flaquees, no permitas que nadie te diga que no puedes hacerlo: lucha, disfruta, vive.

Gracias por tu tiempo.

Y a ti, por tu infinita paciencia y por tu apoyo incondicional en tiempos tan difíciles. No habría llegado hasta aquí sin ti.

Simplemente gracias.

RESUMEN

PLUGIN DINÁMICO PARA FMOD

El objeto de este proyecto es desarrollar un plugin para el motor de videojuegos Unity que permita incorporar el sonido del videojuego con el soporte de Fmod como motor de audio. Se quiere ofrecer una alternativa de sonorización que tenga como objeto reducir los conocimientos de código relacionado con API's que el usuario necesita y sustituirlos por conocimientos a nivel de interfaz visual. Estas interfaces visuales deben ser sencillas de usar.

Como funcionalidad principal el plugin debe contar con una herramienta que permita la programación visual de eventos sonoros controlados por condiciones lógicas.

Como soporte a la depuración se debe incluir un tracker configurable que aporte información relevante relacionada con el proceso de desarrollo y la ejecución del contenido generado con el plugin.

El plugin debe asegurar la coherencia de los datos que trata y la seguridad de estos.

Palabras clave

Unity, Plugin Unity, Editor Unity, Editor Window Unity, FMOD, Programación Visual, Sonido en Videojuegos, Motor de Sonido, Eventos de Sonido

ABSTRACT

DYNAMIC PLUGIN FOR FMOD

The purpose of this project is to develop a plugin for the Unity video game engine that allows the incorporation of the video game sound with the support of Fmod as an audio engine. The aim is to offer a sound alternative that aims to reduce the knowledge of coding related to API's needed by the user and replace it with knowledge at the visual interface level. These visual interfaces should be simple to use.

As a main functionality, the plugin must have a tool that allows the visual programming of sound events controlled by logical conditions.

To support the debugging, a configurable tracker must be included that provides relevant information related to the development process and the execution of the content generated with the plugin.

The plugin must ensure the consistency of the data it processes and the security of the data.

Keywords

Unity, Unity Plugin, Unity Editor, Unity Window Editor, FMOD, Visual Programming, Video Game Sound, Sound Engine, Sound Events

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

1.1 Motivación

El sector de los videojuegos es una de las industrias en mayor auge de nuestra sociedad, y se prevé que lo siga siendo durante los próximos años, lo que ha llevado a muchas personas a intentar llevar a cabo el desarrollo de sus propios videojuegos de forma autónoma e independiente. Sin embargo, una gran parte de ellos no poseen los conocimientos mínimos para dominar las técnicas necesarias para la realización de un proyecto de manera semi-profesional, siendo necesario un equipo multidisciplinar. Así mismo, las pequeñas empresas que intentan afrontar un desarrollo con recursos limitados, una economía menor o un tiempo reducido, ven frenadas sus trayectorias por no poder competir por proyectos con los que comenzar un nuevo futuro.

A grandes rasgos, podemos definir tres áreas de las que se compone el desarrollo de videojuegos: el arte, el sonido y la programación.

El sonido juega un papel fundamental en la interacción de un videojuego, agrega una parte emocional y realista a la experiencia y ayuda a mantener la inmersión en la escena del juego. No solo provoca y despierta todo tipo de sentimientos en los usuarios como tensión, miedo, sorpresa, etc., sino que, además, sirve como apoyo para que las distintas herramientas, interfaces, mecánicas y dinámicas que puede ofrecer una aplicación o videojuego sean fácilmente reconocibles. Permite crear mapas mentales que el usuario forma de aquello que está utilizando para simplificar en el futuro la continuidad de su uso.

Si centramos el proceso creativo en plataformas de desarrollo como Unity, existe todo un mercado de assets y herramientas para ayudar al usuario. Sin embargo, algunas áreas, como la del sonido, están menos explotadas y requieren que el usuario se introduzca en las plataformas a un nivel de desarrollo mucho más profundo.

De forma alternativa, surgen motores de audio que se pueden acoplar a las plataformas de desarrollo para ofrecer una capa de más alto nivel dando la posibilidad de profundizar en la manipulación de sonidos, eventos y todo tipo de

mezclas de una forma mucho más centralizada. Esto puede llegar a ser un arma de doble filo, ya que se le da al usuario la posibilidad de disponer de una nueva fuente de recursos que aparentemente mejorará el resultado final, pero también se amplían los conocimientos requeridos y la dificultad de poder implementar esto en un pequeño proyecto.

Basándome en todo esto, decidí apostar por un Plugin que integre un motor profesional de audio en una plataforma de desarrollo. Debe ofrecer herramientas de sonorización y la posibilidad de crear eventos complejos a través de programación visual que ayuden a los usuarios con menos conocimientos a afrontar el proceso de sonorización de una escena.

1.2 Objetivos

Los objetivos generales del proyecto son:

1.- Implementación de un plugin de uso sencillo que conecte Unity como plataforma de desarrollo con Fmod como motor de audio. El plugin debe ser flexible y adaptarse a los diferentes ámbitos de trabajo (17).

2.- Permitir que el audio se manipule de forma dinámica mediante ajustes en interfaces sencillas de usar.

3. Ofrecer la posibilidad de crear y personalizar grupos de sonidos que se puedan manipular en ejecución. Estos grupos deben contar con parámetros de configuración que ayuden a que el sonido final sea el desaseado.

4.- Ayudar al usuario destino a desarrollar sus proyectos sin necesitar conocimientos de código y sin perder la potencia del motor de audio. Esto requiere del automatismo de todos los procesos que sean posibles y de ofrecer la posibilidad de crear distintos tipos de interacciones complejas entre objetos sonorizados a través de programación visual de eventos.

1.3 Plan de trabajo

Una vez identificados los objetivos más importantes a tratar, se pueden establecer unos pasos a seguir durante el proceso de análisis e implementación del plugin que inicialmente se dividen en dos hitos.

El primero cuenta con dos etapas principales, investigación y desarrollo. Tras ellas tiene lugar un segundo hito que cuenta con otras dos etapas, análisis y refactorización. Estas últimas tienen como objetivo optimizar la integración de las distintas herramientas del plugin y el posible tratamiento de los datos de usuario.

- Hito 1 - Etapa 1: Investigación

En esta etapa se realiza una investigación sobre las distintas versiones de Unity , su compatibilidad con el motor de audio Fmod y su integración (22).

Se analiza el proceso de desarrollo de un plugin basado en assets para la plataforma de desarrollo Unity además de la implementación de editores y ventanas de edición de forma que se pueda dar soporte a la creación de herramientas.

Se estudian y valoran las distintas jerarquías de actores implicados en el plugin y la estructura principal que sirve de soporte para sus herramientas. Se establece también la forma en la que las herramientas se presentan al usuario y cómo se soportan estas dentro del entorno de la plataforma de desarrollo.

Por otro lado, se investiga el soporte y tratamiento de datos a implementar en el plugin para su uso en los estados que se disocia la plataforma de desarrollo - "Editor Time y Run Time"-, en adelante, "estados de la plataforma de desarrollo", así como salvar estados de configuración cuando se cierra la aplicación y que afectan a secciones críticas como scripts, campos de configuración de los propios editores o datos producidos por el uso del plugin.

Así mismo, es necesario el diseño del proceso de control y gestión de eventos sonoros con los que se trata en el plugin (23), diseñar las estructuras necesarias para su manipulación, así como los algoritmos de evaluación y priorización de análisis en ejecución.

Por último, se estudia el proceso de implementación del sistema de depuración y comunicación de las distintas herramientas del plugin además de diseñar el sistema de tracking de mensajes de forma que sea compatible con los distintos estados de la plataforma de desarrollo.

- Hito 1 - Etapa 2: Desarrollo

Implementación de un prototipo básico del plugin de audio basado en assets y escrito en lenguaje de bajo nivel C#. En esta etapa se realiza la implementación del contenido generado en el Hito 1 - Etapa 1: Investigación.

- Hito 2 - Etapa 1: Análisis

Tras la implementación de una primera versión completa, se realiza un análisis profundo sobre el estado en que quedan implementadas las herramientas en este punto. Se valora el tratamiento de los datos y la liberación de carga de contenido relacionado con el tiempo de edición cuando se ejecuta la escena.

- Hito 2 - Etapa 2: Refactorización

Por último, se realiza una fase de refactorización que corrija y mejore las interacciones entre las herramientas del plugin, tratando de potenciar la sincronización y la seguridad de los datos producidos durante la fase de desarrollo en función de los resultados del Hito 2 - Etapa 1: Análisis.

1.4 Estructura de la memoria

Los restantes capítulos de la memoria están organizados como sigue:

El capítulo 2 pretende aclarar qué es DFP, para qué sirve y cuáles son los puntos fuertes que intenta abordar.

El capítulo 3, "Definición y funcionalidades", pretende dar una visión global de qué es el plugin y los diferentes problemas que intenta abordar.

El capítulo 4, "Conceptos Básicos", tiene como objetivo ofrecer una visión general de los conceptos que se consideran necesarios para facilitar la comprensión al lector de la documentación.

El capítulo 5, "Manual de Usuario", ofrece una visión global del uso de las herramientas DFP y sus interfaces enfocado al usuario.

El capítulo 6, "Implementación", trata aspectos más técnicos de desarrollo. En este capítulo se habla de cómo se ha implementado el plugin, tecnologías que le dan soporte y cómo funciona internamente.

El capítulo 7, "Trabajo Futuro", recopila ideas que han ido surgiendo durante el desarrollo para ampliar o extender las funcionalidades del plugin DFP.

El capítulo 8, "Conclusiones", pretende hacer un análisis del estado en el que se queda el proyecto tras finalizar el desarrollo.

El capítulo 10, "Bibliografía", tiene como objeto recoger las referencias más importantes utilizadas para aprendizaje o consulta que se han utilizado durante el transcurso total de los diferentes hitos del proyecto.

El capítulo 11, "Apéndices", recoge información que no tenía cabida en los otros capítulos pero que puede resultar útil al lector, "Lecciones aprendidas", "Guía de instalación", "Creación de bancos de sonido Fmod" y "Link FmodStudio a proyecto Unity"

Capítulo 2 - Introduction

2.1 Motivation

The video game industry is one of the fastest-growing sectors in our society, and it is expected to continue to be so in the coming years. This has led many individuals to attempt to develop their own video games independently and autonomously. However, a significant number of them lack the basic knowledge needed to master the techniques required for semi-professional project development, needing a multidisciplinary team. Similarly, small companies trying to undertake development with limited resources, lower budgets, or reduced time find their progress hindered by their inability to compete for projects that could mark the start of a new future.

Broadly speaking, we can define three areas involved in video game development: art, sound, and programming.

Sound plays a fundamental role in the interaction of a video game, adding an emotional and realistic component to the experience and helping to maintain immersion in the game scene. It not only evokes and stirs various feelings in users such as tension, fear, surprise, etc., but also supports the recognition of different tools, interfaces, mechanics, and dynamics that an application or video game may offer. It allows users to create mental maps of what they are using to simplify the continuity of its use in the future.

If we focus on the creative process of development platforms like Unity, there is a whole market of assets and tools to assist the user. However, some areas, such as sound, are less explored and require the user to delve into the platforms at a much deeper development level.

Alternatively, audio engines have emerged that can be integrated into development platforms to provide a higher-level layer, offering the possibility to delve into the manipulation of sounds, events, and various types of mixes in a much more centralized manner. This can be a double-edged sword, as it provides the user with a new resource that not also improves the final result, but also increases the required knowledge and difficulty of implementing it in a small project.

Based on all this, I decided to invest in a plugin that integrates a professional audio engine into a development platform. It should offer sound design tools and the

ability to create complex events through visual programming, assisting users with less knowledge in tackling the sound design process of a scene.

2.2 Goals

The general objectives of the project are:

Implementation of an easy-to-use plugin that connects Unity as the development platform with FMOD as the audio engine. The plugin should be flexible and adaptable to different work environments.

Allow audio to be dynamically manipulated through adjustments in user-friendly interfaces.

Provide the capability to create and customize sound groups that can be manipulated during runtime. These groups should have configuration parameters that help achieve the desired final sound.

Assist the target user in developing their projects without requiring coding knowledge and without losing the power of the audio engine. This necessitates the automation of all possible processes and the ability to create various types of complex interactions between sound-processed objects through visual event programming.

2.3 Work Plan

Once the most important objectives have been identified, steps can be established for the analysis and implementation process of the plugin, initially divided into two milestones.

The first milestone consists of two main stages: research and development. Following these, a second milestone is achieved, which includes two more stages: analysis and refactoring. These latter stages aim to optimize the integration of the plugin's various tools and the potential handling of user data.

• Milestone 1 - Stage 1: Research. In this stage, research is conducted on the different versions of Unity, its compatibility with the FMOD audio engine, and its integration. The development process of an asset-based plugin for the Unity development platform is analyzed, including the implementation of editors and editing windows to support tool creation. The various hierarchies of actors involved in the plugin and the main structure supporting its tools are studied and evaluated. The

way tools are presented to the user and how they are supported within the development platform environment is also established. Additionally, it is necessary to investigate support and handling of data to be implemented in the plugin for use in states that dissociate from the development platform—"Editor Time and Run Time," henceforth referred to as "development platform states"—, as well as saving configuration states when the application is closed, affecting critical sections such as scripts, editor configuration fields, or data produced by plugin use. Furthermore, it is necessary to design the process for controlling and managing sound events within the plugin, including designing the necessary structures for manipulation, as well as algorithms for evaluation and prioritization of analysis during runtime. Finally, the implementation of the debugging and communication system for the plugin's various tools is studied, along with designing a message tracking system compatible with the different development platform states.

• Milestone 1 - Stage 2: Development. Implementation of a basic prototype of the asset-based audio plugin written in the low-level language C#. This stage involves implementing the content generated in Milestone 1 - Stage 1: Research.

• Milestone 2 - Stage 1: Analysis. Following the implementation of a complete initial version, a thorough analysis is conducted on the state of the tools implemented at this point. The handling of data and the release of content load related to the editing time when the scene is executed are evaluated.

• Milestone 2 - Stage 2: Refactoring. Finally, a refactoring phase is carried out to correct and improve interactions between the plugin's tools, aiming to enhance synchronization and data security produced during the development phase based on the results of Milestone 2 - Stage 1: Analysis.

2.4 Structure of the memory

The remaining chapters of the report are organized as follows:

Chapter 2 aims to clarify what DFP is, what it is used for, and the key strengths it addresses.

Chapter 3, "Definition and Features," provides an overview of what the plugin is and the different problems it aims to address.

Chapter 4, "Basic Concepts," aims to offer a general view of the concepts considered necessary to facilitate the reader's understanding of the documentation.

Chapter 5, "User Manual," provides an overview of the use of DFP tools and their interfaces, focusing on the user.

Chapter 6, "Implementation," covers more technical aspects of development. This chapter discusses how the plugin has been implemented, the supporting technologies, and its internal workings.

Chapter 7, "Future Work," gathers ideas that have emerged during development to expand or extend the functionalities of the DFP plugin.

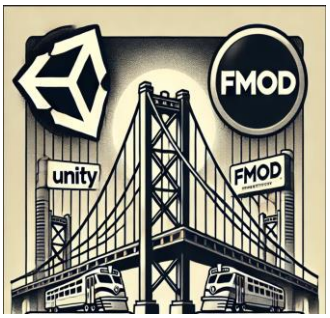
Chapter 8, "Conclusions," provides an analysis of the project's state after the development phase is completed.

Chapter 10, "Bibliography," aims to collect the most important references used for learning or consultation throughout the different milestones of the project.

Chapter 11, "Appendices," includes information that did not fit into the other chapters but may be useful to the reader, such as "Lessons Learned," "Installation Guide," "Creating FMOD Sound Banks," and "Linking FMOD Studio to Unity Project."

Capítulo 3 - Definición y funcionalidades

3.1 ¿Qué es DFP?



Las siglas "DFP" significan "Dynamic Fmod Plugin".

DFP es de un plugin creado en C# para la plataforma de desarrollo Unity que integra Fmod como motor de audio profesional.

Tiene como objetivo principal automatizar tareas y ofrecer alternativas de sonorización de escenas con herramientas basadas en interfaces visuales. Se busca profundizar en la idea de que sea simple y sencillo, implementando comportamientos similares a otras interfaces de programación visual como Scratch, que permitan a una persona sin conocimientos de programación crear eventos sonoros que simulen procesos programados.

3.2 ¿Qué trata de aportar DFP?

DFP se centra en abordar una serie de problemas y características concretas que se comentan a continuación.

1. Reducir carga de conocimientos de API o de Componentes del motor de audio (18) por conocimientos de interfaz.

Fmod es un motor de audio que está especializado en la creación de contenido de videojuegos y aplicaciones y está compuesto por diferentes API's (19):

- Fmod Studio
- API de Fmod Studio
- API de Bajo Nivel

Actualmente está integrado en plataformas como Unreal Engine o Unity.

Con este plugin se trata de reducir la carga de conocimientos de API's que debe conocer el usuario.

2. Control y gestión de forma grupal de unidades sonorizadas con DFP

Si dos objetos distintos se sonorizan con Fmod for Unity, estos no se relacionan de ninguna manera. A efectos prácticos son dos objetos individuales y cada uno cuenta con sus componentes.

Fmod cuenta en su API de Bajo Nivel con estructuras similares a un grupo (ChannelGroup / Bus). Estas estructuras están creadas para dar soporte a la parte de Studio y están preparadas para su gestión en tiempo de edición.

Este plugin trata de solucionar este aspecto con las herramientas Miembros DFP y Grupos DFP. Estos Grupos DFP operan con los Miembros DFP tanto en tiempo de edición como en tiempo de ejecución además de ofrecer una herramienta más de control.

3. Programación de eventos sonoros

Si se quiere ofrecer una alternativa a usuarios con menor experiencia en el ámbito del desarrollo se necesita una herramienta que simule un proceso similar y que su uso se presente de forma más amigable o sencillo para el usuario.

Este plugin trata de ofrecer una alternativa a la programación formal con la herramienta de programación visual que permite componer secuencias de evaluación complejas asociadas a acciones de configuración en forma de eventos. Eventos que pueden volverse más complejos añadiendo multi evaluaciones, multi acciones, tiempos de retardo y limitadores de evaluación.

4. Sistema de depuración

Cuando trabajamos con Fmod for Unity dependemos de programar las acciones que queremos depurar por consola o de las cuales queremos mantener un registro para evaluar.

DFP ofrece al usuario la posibilidad de depurar y filtrar información relativa al proceso de sonorización, también da la oportunidad de guardar un registro global de los informes para su posterior evaluación. Todo esto integrado en un tracker configurable.

5. Sincronización y protección de información y módulos sensibles

Las herramientas van registrando una gran cantidad de datos y puede resultar fatídico perderlos, por ejemplo, las referencias a los objetos sonorizados con el plugin,

los assets de las herramientas o los conjuntos de datos críticos que el usuario va generando.

Además, en el ámbito del desarrollo en Unity, se trabaja con un número grande de objetos y assets, por lo que puede ocurrir fácilmente que se borren objetos necesarios de forma no intencionada, por ejemplo, el objeto que contiene los managers del plugin.

Estas características de sincronización y protección surgen como necesidad del material que se quiere desarrollar, como soporte interno al plugin y como protección ante el mal uso por parte del usuario.

Además, al integrar varias herramientas que trabajan construyendo datos se requiere de una entidad que los unifique, que mantenga el control de estos y garantice la coherencia de datos en todas las herramientas, así como garantizar la no-duplicidad de registros.

El plugin trata de dar solución a este aspecto con la Caché DFP, una herramienta autónoma de gestión y protección de datos que cuenta con funcionalidades para proteger los módulos importantes del plugin además de dar soporte a la sincronización de datos.

Capítulo 4 - Conceptos básicos

Esta sección resume los conceptos más importantes que se usan en el desarrollo de este proyecto. No se incluye una definición detallada, ya que la mayoría de estos conceptos se tratan de manera extensa en sus respectivas secciones, sino que se definen los conceptos necesarios para la correcta comprensión de este documento.

4.1 Actores DFP

El plugin, de forma general, se soporta en figuras o "actores" que interactúan en todas las herramientas, cooperando para sonorizar la escena, se hará referencia a ellos constantemente a lo largo de la documentación. Estos actores son:

Oyente DFP, Miembro DFP, Grupo DFP, Atributo DFP, Entidades DFP, Operadores DFP, Trinket DFP, Action DFP, Evento DFP, Managers DFP (Core, Group, Tracker...) y Sistemas de protección y sincronización DFP (Caché, Recovery..).

4.2 Oyente DFP

Se conoce como Oyente DFP o Listener DFP, al centro o receptor de la sonorización 3D de la escena. Es el objeto para el cual se orienta toda la sonorización.

4.3 Miembro DFP

En términos generales, un Miembro DFP es un objeto que está sonorizado con el plugin y que puede interactuar con el total de las herramientas de este. A efectos prácticos, son los emisores de la escena.

4.4 Grupo DFP

Un Grupo DFP es un objeto que se define para agrupar Miembros DFP y poder controlar de forma grupal las acciones sonoras que ofrece el plugin sobre los distintos Miembros DFP, con independencia del ámbito de trabajo.

4.5 Atributo DFP

Los Atributos DFP se utilizan, entre otras cosas, para definir Operadores DFP. Para poder hacer una operación de evaluación, se necesita algo que evaluar y para poder ejercer una operación o acción sobre algo, se necesita definir sobre qué se ejecuta esa acción: Atributos DFP.

4.6 Evento DFP

En la figura del Evento DFP intervienen distintos Actores DFP (Entidades DFP, Operadores DFP, Trinket DFP, Action DFP, Miembros DFP, Grupos DFp…), pero en general, se puede decir que un Evento DFP representa la asociación de una serie de evaluaciones lógicas con una serie de operaciones de configuración sonora que se aplican a Miembros DFP y Grupos DFP.

4.7 Entidades DFP

Las Entidades DFP representan un conjunto de Miembros DFP y Grupos DFP que se definen para recibir un tratamiento especial, ya que son utilizadas en la creación de Eventos DFP.

Existen dos tipos de Entidades:

- Entidad Propietaria
- Entidad Objetivo

4.7.1 Entidad DFP Propietaria

La Entidad Propietaria agrupa a los sujetos de evaluación. Se pueden componer por Miembros DFP, Grupos DFP o una mezcla de estos.

4.7.2 Entidad DFP Objetivo

La Entidad Objetivo agrupa a los sujetos receptores de las acciones definidas en el Evento DFP. Se pueden componer por Miembros DFP, Grupos DFP o una mezcla de estos.

4.8 Trinket DFP

Trinket DFP forma parte de los Eventos DFP. Es el encargado de la gestión de las evaluaciones y de la Entidad Propietaria a la que se evalúa, es decir, definir qué evaluaciones se deben hacer y a quién dentro de un Evento DFP.

Las condiciones a evaluar se crean y configuran en forma de operadores.

Por lo tanto, Trinket lo conforman:

- Operadores de evaluación para definir las evaluaciones.
- Miembros DFP para definir la Entidad Propietaria.
- Grupos DFP para definir la Entidad Propietaria.

4.9 Action DFP

Action DFP forma parte de los Eventos DFP. Es el encargado de la gestión de las acciones y de la Entidad Objetivo a la que se le aplican dichas acciones, es decir, definir qué acciones se deben hacer y a quién dentro de un Evento DFP.

Las acciones se crean y configuran en forma de operadores.

Por lo tanto, Action lo conforman.

- Operadores de acción para definir las acciones.
- Miembros DFP para definir la Entidad Objetivo.
- Grupos DFP para definir la Entidad Objetivo.

4.10 Operadores DFP

Como se comenta en la sección del Trinket y Action, se necesitan los Operadores DFP para definir Eventos DFP. Estos son utilizados para definir instrucciones de evaluación o acción en conjunción con los Atributos DFP.

Se habla más detalladamente de esto en sus correspondientes secciones.

4.11 Managers DFP

El plugin contiene una serie de Managers que administran el comportamiento de las distintas herramientas en Editor Time o Run Time (consultar secciones "Tiempo de Edición" y "Tiempo de Ejecución").

Los Managers gestionan las partes más fundamentales del Plugin:

- Core Manager: Gestor de las acciones relacionadas con el sistema y el núcleo del plugin en Run Time.
- Group Manager: Gestor de Grupos DFP en Run Time.
- Tracker Manager: Gestor de Logs en Editor Time y Run Time. Gestor de Eventos DFP en Run Time.
- Caché: Gestor de sistemas de protecciones de módulos y respaldo de información sensible
- Makers: Editores o Ventanas de edición para ciertos actores como los Grupos DFP o los Eventos DFP.

4.12 Sistemas de sincronización

El desarrollo del plugin se presenta en forma de herramientas que están construidas sobre interfaces que ofrecen soporte en Editor Time y en Run Time.

El plugin cuenta con un sistema de sincronización de editores que hace que la información se mantenga coherente en el uso activo de las diferentes interfaces que manipulan información conjunta.

4.13 Sistemas de protección

Los sistemas de protección son una parte importante del Plugin, se ejecutan en segundo plano y realizan una cantidad importante de tareas de seguridad y administración del plugin para facilitar al usuario el uso del Plugin DFP.

Gestionan la protección de los módulos importantes del plugin, ofrecen cobertura para la resolución de incidencias leves y graves que pongan en peligro la integridad del plugin y consecuentemente la de la escena del proyecto.

4.14 Tiempo de edición (Editor time)

En el uso de la plataforma de desarrollo de Unity se pueden diferenciar dos grandes áreas de ejecución o de control por frames: Editor Time y Run Time.

Cuando se habla de EDITOR TIME, se hace referencia a todo lo relacionado al ciclo de ejecución del Editor (IDE) de Unity en tiempo de edición, entre los cuales se incluyen los ciclos de actualización con los que trabaja el plugin como OnGUI,

OnInspectorGUI, OnEnable. Está soportado por la API de "UnityEditor" y alimenta toda la parte del Frontend de la plataforma de desarrollo.

4.15 Tiempo de ejecución (Run time)

En el uso de la plataforma de desarrollo de Unity se pueden diferenciar dos grandes áreas de ejecución o de control por frames: Editor Time y Run Time.

Cuando se habla de RUN TIME, se hace referencia a todo lo relacionado al ciclo de ejecución de la escena, es decir, cuando la aplicación está en ejecución, por lo que incluye los ciclos de actualización como "Awake," "Start", "Update", etc con los que trabaja Unity. Está soportado por la API de "UnityEngine" y alimenta toda la parte de ejecución de escenas de la plataforma de desarrollo.

Capítulo 5 - Manual de usuario

5.1 Importando plugin DFP en Unity

Para trabajar con DFP es necesario abrir el proyecto en Unity e importar el plugin en el directorio "Plugins". Si este no existe puede crearse dentro de la carpeta "Assets" antes de importar.

También podemos hacerlo de forma manual copiando el contenido que se ha descargado del Plugin DFP, dentro del directorio del nuestro proyecto de Unity, sin olvidar hacerlo dentro de la carpeta "Plugins". (Ilustración 1)

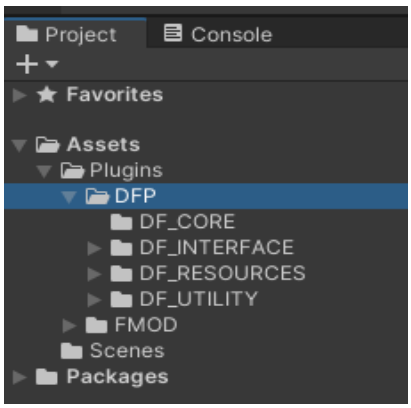


Ilustración 1 Importación plugin DFP

De cualquier forma, la ruta del plugin tras haberse importado debe ser:

- "/Assets/Plugins/DFP"

Si se han realizado correctamente los pasos anteriores, el resultado debe quedar parecido a la imagen (Ilustración 1)en función de los plugins que se tengan instalados en un proyecto de Unity.

5.2 Ventana principal DFP (Main Window)

Al importar el plugin se habilita una nueva opción en la barra de herramientas de nuestra escena en Unity, en la parte superior, concretamente en la pestaña de Fmod.

En esta pestaña podremos encontrar la opción encargada de abrir la ventana principal de trabajo: DFP - Main Window (Ilustración 2).

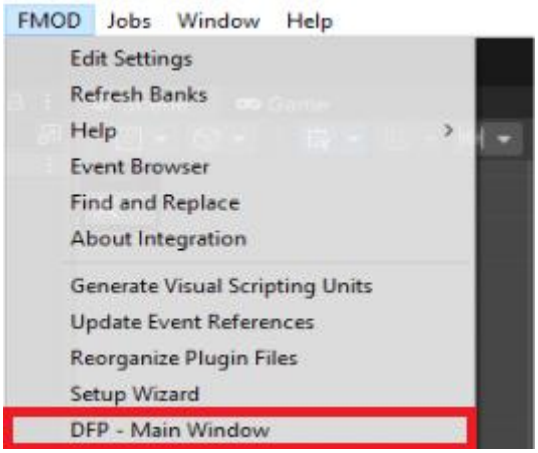


Ilustración 2 Acceso DFP Main Window

DFP - Main WIndow agrupa todas las funcionalidades principales del Plugin DFP (Ilustración 3):

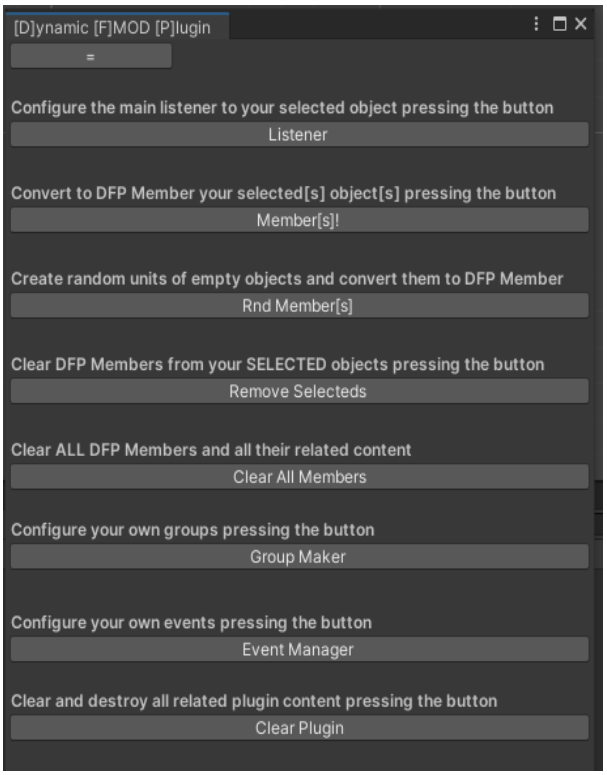


Ilustración 3 Funcionalidades DFP Main Window

- Listener - Configura el objeto seleccionado en escena para que se comporte como Oyente DFP principal de la sonorización del proyecto. Si ya existiera otro, lo reemplaza e intenta restaurar el primero a su estado anterior.
- Member[s]! - Convierte en Miembro/s DFP del plugin a los objetos que estén seleccionados en escena. Un objeto que se convierte en miembro del plugin pasa a estar sonorizado y preparado para trabajar con él.
- Rnd Member[s] - Crea un objeto aleatorio en escena y lo convierte en Miembro DFP del plugin.
- Remove selected - Quita el estatus de Miembro DFP a los objetos seleccionados en escena, si los hubiera, e intenta restaurarlos a su estado anterior.
- Clear All Members - Quita el estatus de Miembro DFP a todos los objetos en escena, si los hubiera, e intenta restaurarlos a su estado anterior.
- Group Maker - Abre el editor de Grupos DFP, con el que podemos crearlos, borrarlos o editarlos.
- Event Manager - Abre el editor de Eventos DFP con el que podemos crearlos, borrarlos o editarlos.
- Clear Plugin - Borra todo el contenido del plugin e intenta restaurar los objetos de la escena al estado anterior en el que estaban antes de entrar en contacto con el plugin.
- El botón "=" cambia el Wrapper (aspecto y disposición visual) de la ventana de trabajo.

5.3 Herramienta Editor de Miembro DFP

Los objetos que cuentan con el status de Miembro DFP cuentan con un editor personalizado (2).

Para acceder al editor seleccionamos cualquiera de los objetos en escena que previamente hayamos convertido en Miembro DFP; en el inspector de Unity, podremos encontrar la zona del Editor de Miembros. Los miembros se pueden distinguir porque aparecerá el acrónimo "[DFP]" delante de sus nombres y sus identificadores dentro del plugin al final de sus nombres dentro de símbolos "<>".

Inicialmente se muestran dos secciones en la zona de Miembros DFP: Sound y Settings (Ilustración 4).

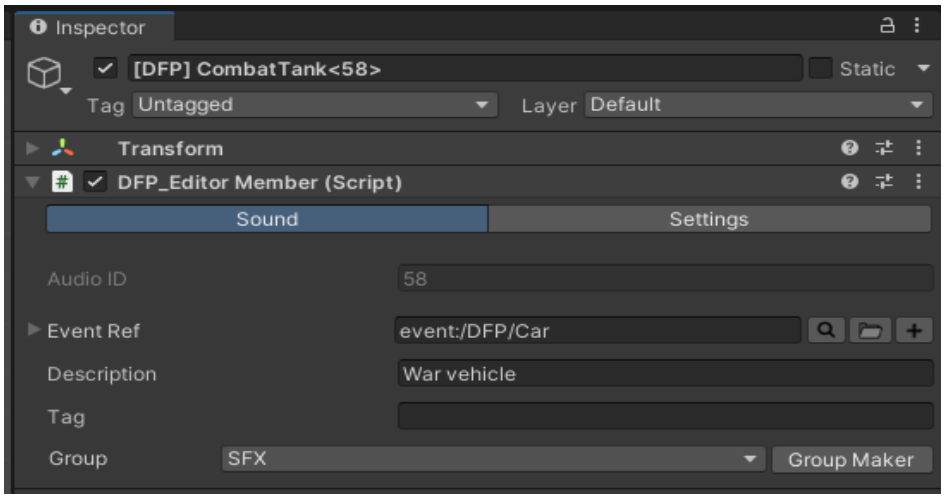


Ilustración 4 Ventana editor de miembros

Desde la opción Settings podemos encontrar dos opciones configurables para habilitar las secciones: Expert Mode y Developer Mode (Ilustración 5).

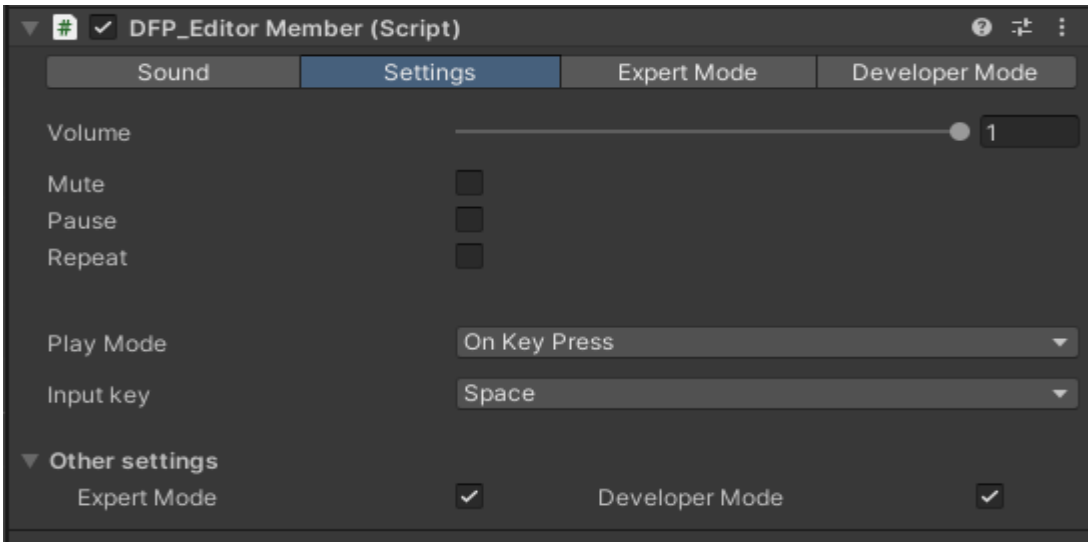


Ilustración 5 Expert Mode y Developer Mode

5.3.1 Miembro DFP, sección Sound

Esta sección agrupa las opciones de configuración básicas para el evento de Fmod y los eventos DFP que se van a asociar al objeto miembro DFP en escena (Ilustración 6).

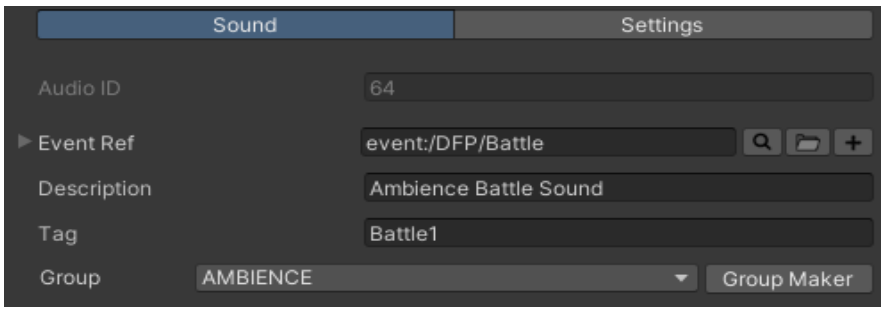


Ilustración 6 Sección Sound

Además, podremos ver el ID que tiene para el plugin, abrir el editor de grupos, etc.

En esta sección encontramos el campo "Event Ref", único campo obligatorio que el usuario debe rellenar para que el objeto pueda interactuar correctamente con el plugin. Este "Event Ref" hace referencia al path del Evento Fmod que queremos utilizar en el Miembro DFP. Para ello, el usuario simplemente debe escribir la ruta del evento de Fmod que quiere usar o usar uno de los buscadores de eventos para seleccionarlo dentro de los posibles bancos que contenga la escena.

5.3.2 Miembro DFP, sección Settings

Sección que agrupa opciones de configuración adicionales que no requieren de conocimientos avanzados y que determinan el estado inicial en el que debe arrancar en escena dicho objeto. También define algunos de sus comportamientos, como el modo de reproducción (Ilustración 7).

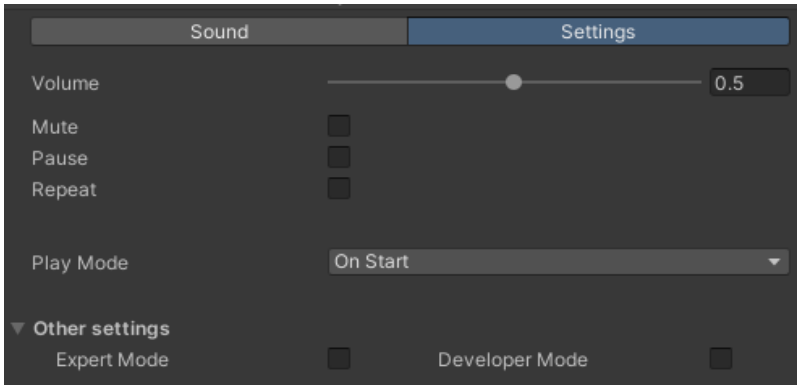


Ilustración 7 Sección settings

Además, en la pestaña de "Other Settings" podemos marcar las opciones "Expert Mode" y "Developer Mode" que nos habilitarán dichas secciones (Ilustración 8):

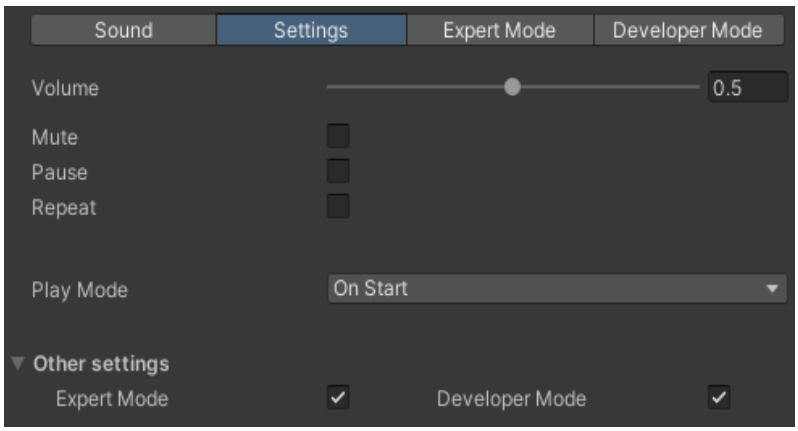


Ilustración 8 Secciones Expert Mode y Developer Mode

5.3.3 Miembro DFP, sección Expert Mode

Esta sección nos permite configurar los valores iniciales de parámetros que requieren conocimientos un poco más avanzados de sonido. Está diseñada para añadir aspectos de configuración avanzada En esta sección podemos remover el estatus de miembro (Ilustración 9).

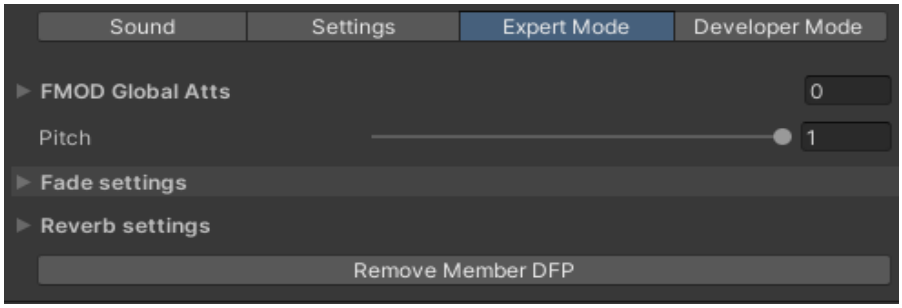


Ilustración 9 Sección Expert Mode

5.3.4 Miembro DFP, sección Developer Mode

Sección que nos muestra los Eventos DFP en los que el objeto está implicado y nos muestra información sobre ellos, como el rol que adoptan en dichos eventos (Ilustración 10).

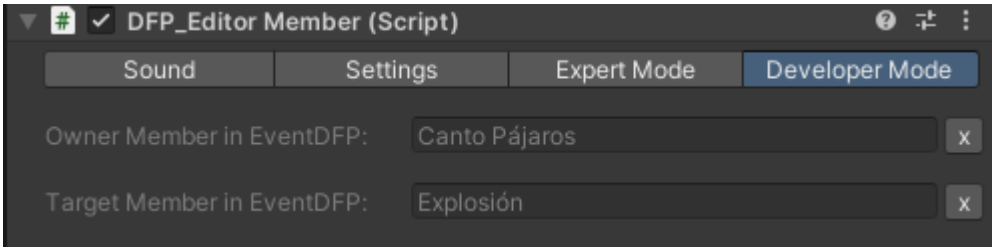


Ilustración 10 Sección Developer Mode

Además, nos permite borrar dichas relaciones o abrir el editor de eventos.

En este caso, nos está indicando que este Miembro DFP forma parte de la Entidad Propietaria en el Evento DFP "Canto Pájaros" y además forma parte de la Entidad Objetivo en el Evento DFP "Explosión"

5.4 Herramienta Group Maker

El editor/creador de Grupos trabaja con un tipo de asset que se focaliza en gestionar y serializar datos relacionados con Grupos DFP. El asset se podría asociar a un "grupo de grupos" que se puede editar para crear, borrar y modificar los propios Grupos DFP que pertenezcan al asset a través del Group Maker.

5.4.1 Creación Group Maker

Podemos crear un Asset de tipo "Group Maker" vacío desde "Create", el menú de acceso rápido de Unity. Ahí podremos encontrar el submenú "DFP", el cual está diseñado para agilizar y facilitar el proceso de creación de recursos al usuario y entre los cuales se encuentra "GroupMaker" (Ilustración 11).

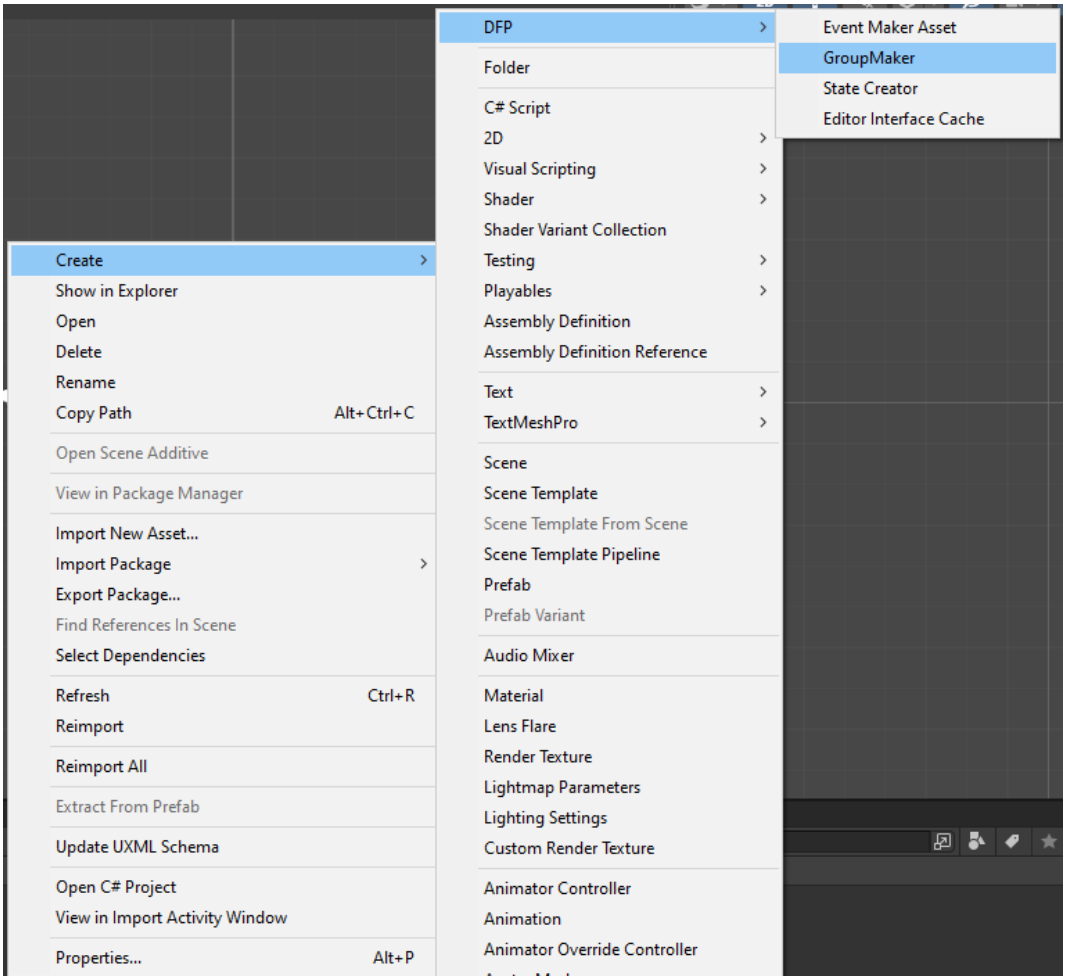


Ilustración 11 Creación Group Maker

A efectos prácticos, como su nombre indica, Group maker es un asset que permite crear, modificar y borrar Grupos DFP.

NO es necesario que el usuario cree un asset de grupo, el plugin se encarga de crear uno por defecto, esta herramienta se ofrece como alternativa para la carga y descarga de asset de grupos personalizados en escena.

5.4.2 Assets de grupos DFP

Los assets o recursos que creamos con Group Maker se crearán en la carpeta que tengamos abierta en el proyecto (Ilustración 12). Estos assets pueden moverse de ubicación como cualquier fichero de recurso, sin embargo, se recomienda guardar los del tipo "Group Maker" en el directorio:

"/Assets/Plugins/DFP/DF_RESOURCES/GROUPS/"

Ilustración 12 Assets

El plugin está preparado para trabajar en esta ruta con este tipo de recursos, aunque se puede utilizar desde cualquier otro directorio.

Se apuesta por la serialización de datos en este caso para ofrecer versatilidad en la creación y reutilización de este tipo de asset en diferentes escenas o diferentes proyectos, ya que podemos exportar estos assets simplemente copiándolos a un nuevo proyecto.

Si no existe ningún Asset de grupo en el proyecto, el plugin nos creará uno automáticamente con los siguientes datos:

- Nombre: "DFP_GroupMaker"
- Ruta: "Assets/Plugins/DFP/DF_RESOURCES/GROUPS/"

Dicho Asset contará con los Grupos DFP mínimos necesarios para trabajar (Ilustración 13).

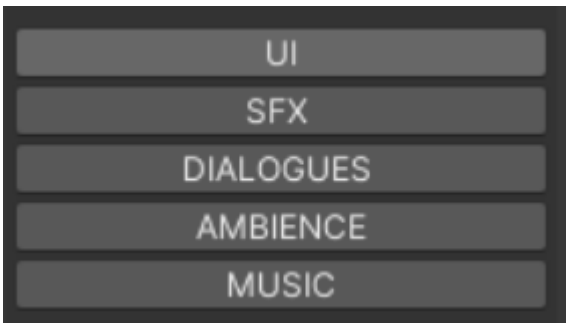


Ilustración 13 Grupos DFP de cada asset

Estos Grupos DFP están prediseñados en el plugin y sirven de soporte para que el asset siempre sea funcional a la vez que ofrece los grupos más típicos usados en el desarrollo como elementos iniciales para agilizar el proceso al usuario. Estos Grupos DFP predefinidos se pueden editar, pero no borrar.

5.4.3 Carga de Group Maker en Escena

Los Assets creados con Group Maker cuentan con su propio editor.

A través de la ventana principal (Main Window) podemos acceder al Group Maker que controla nuestra escena. En la parte superior se encuentra el cargador de assets, desde el que bastará con escribir la ruta y presionar el botón "Load" (Ilustración 14).

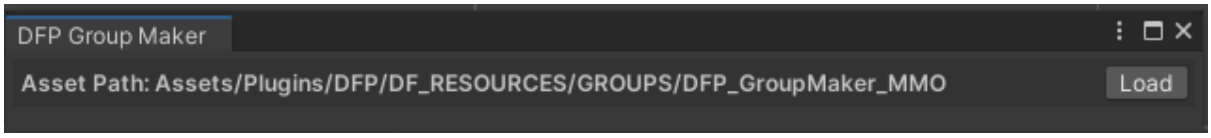


Ilustración 14 Carga de Group Maker

5.4.4 Editor de grupos DFP

Si se intenta editar un asset de grupo que no esté inicializado a través del editor, este nos mostrará la información del asset de la siguiente manera (Ilustración 15):

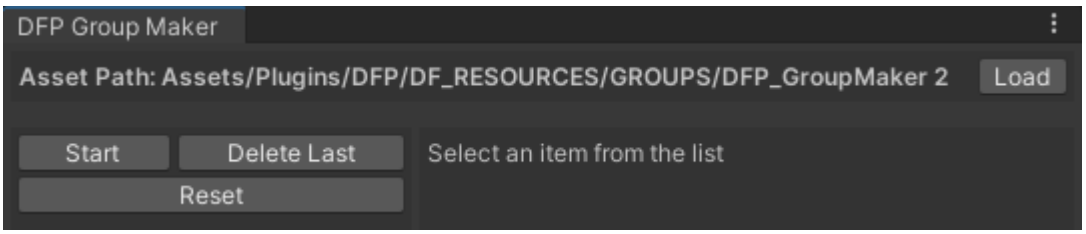


Ilustración 15 Assets no inicializados

En este caso, el botón Create es sustituido por Start. Las funciones que encontremos en este menú son:

- Start: Inicializa un "Asset" de grupo no inicializado (entre otros datos) los grupos básicos que aparecen en la imagen X de este tutorial.
- Load: Nos permite cargar un "Asset de grupo" a través de su ruta o "Path".
- Create: Crea un nuevo grupo dentro del "Asset de grupo" que esté cargado.
- Delete Last: Borra el último grupo que se ha creado.
- Reset: Reinicia el "Asset de grupo" borrando todos los datos que haya en él.

Una vez inicializado, el editor invierte la serialización del asset y nos muestra un listado de los diferentes Grupos DFP que contiene el asset (Ilustración 16):

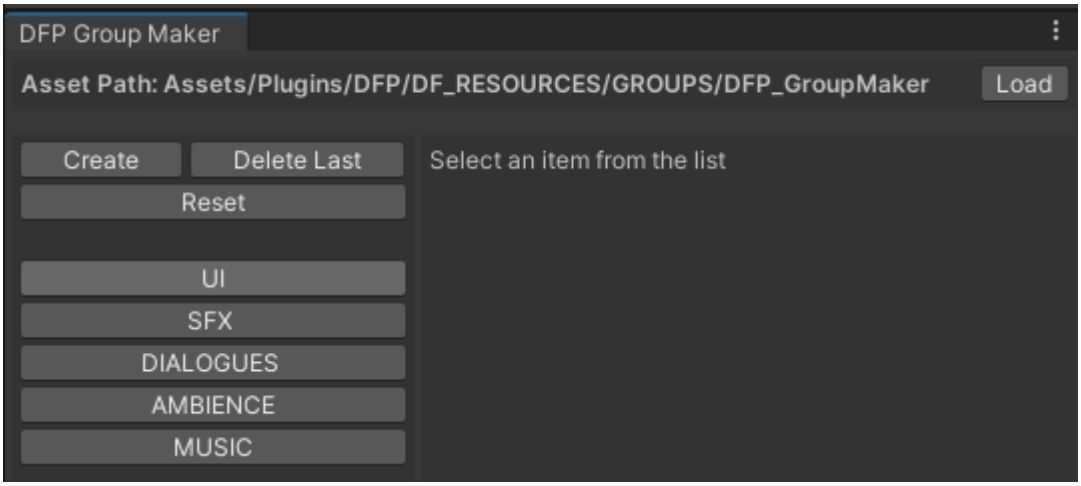


Ilustración 16 Grupos de cada asset

Si seleccionamos cualquiera de los Grupos DFP del listado se mostrarán en el editor sus datos serializados (Ilustración 17):

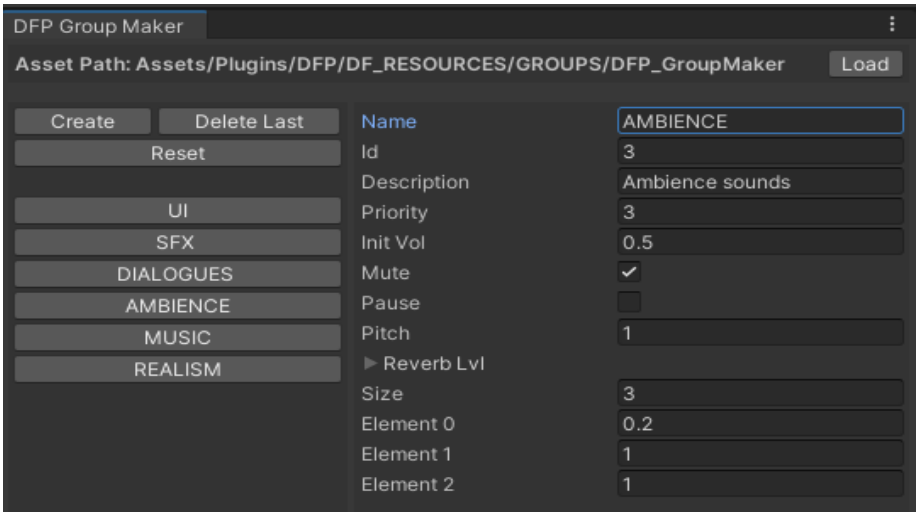


Ilustración 17 Datos serializados grupos DFP

La mayoría de los campos se explican con su propio nombre o la descripción, teniendo que prestar especial atención al campo "Priority".

El plugin utiliza "Priority" en sus algoritmos para priorizar unos grupos sobre otros a la hora de resolver cualquier tipo de acción en que se vean involucrados, por ejemplo, la prioridad de ejecución entre dos Eventos DFP que quieren ser lanzados al mismo instante. Un número menor indica una mayor prioridad.

Los grupos pueden compartir valores numéricos de prioridad. Cuando dos grupos comparten prioridad y el plugin necesita gestionar dichos grupos para alguna

actividad en común, será el plugin quien decida de forma autónoma cómo resolver en cada caso los órdenes de prioridad, siendo en la mayoría de ellos por tiempo en la herramienta o tiempo en el sistema.

5.5 Herramienta Event Maker

La herramienta Editor o creador de Eventos DFP (Event Maker) se abre desde la ventana principal o Main Window. Esta herramienta cuenta con cuatro secciones que se explican de forma independiente (Ilustración 18).

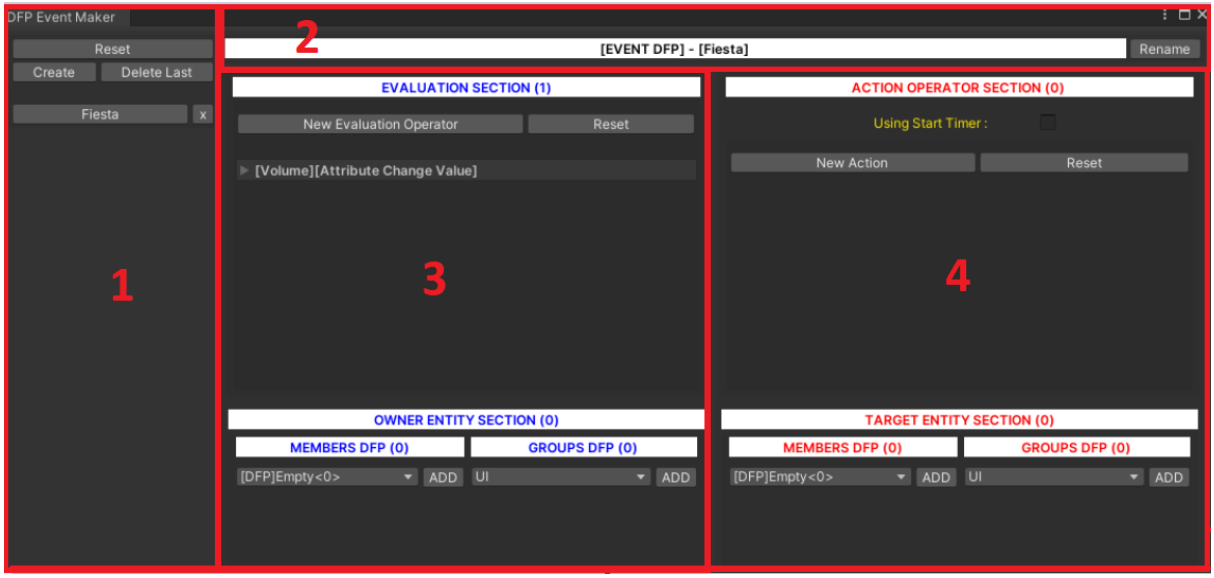


Ilustración 18 Ventana Event Maker

5.5.1 Sección Eventos DFP

Contiene el listado de Eventos DFP y una serie de botones que manipulan la creación, activación o destrucción de éstos. Está situado en la parte izquierda del editor (Ilustración 19).

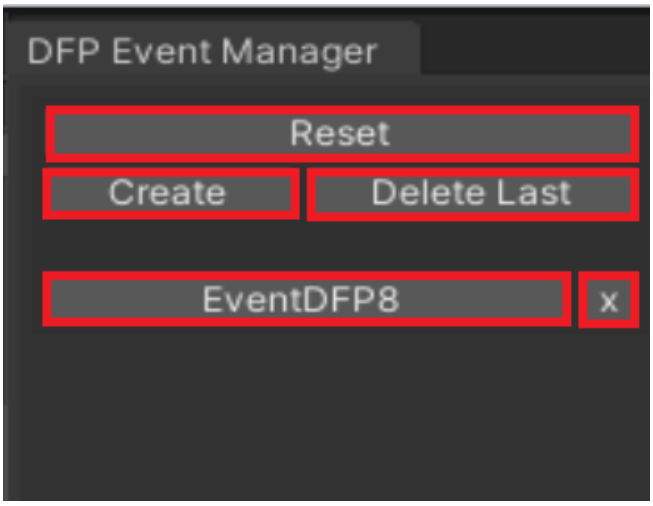


Ilustración 19 Eventos DFP

- Reset: Borra todos los eventos y toda la configuración que se haya podido añadir a ellos. No tiene confirmación.
- Create: Crea un evento vacío con un nombre por defecto.
- Delete Last: Borra el último evento añadido a la lista. No tiene confirmación.
- EventDFP8**:** Nombre del único evento que hay en el listado en el momento de la captura. Estos identificadores actúan como botones que se crean dinámicamente que cargan en el editor los datos serializados del evento.
- x: Borra el evento que contiene a su izquierda. No tiene confirmación.

5.5.2 Sección Eventos Superior

Aparece información sobre el Evento DFP que se está manipulando con el Editor de Eventos y cuenta con un botón para renombrarlo, se encuentra en la parte superior del editor (Ilustración 20).

Ilustración 20 Sección Eventos DFP Superior

5.5.3 Sección Trinket DFP

Trinket se encarga de controlar las condiciones que se van a evaluar y a la Entidad Propietaria (Ilustración 21).

- EVALUATION SECTION: Las condiciones a evaluar se crean y configuran en forma de operadores, en la parte superior de la zona del Trinket.
- OWNER ENTITY SECTION: La entidad propietaria se define con los Miembros DFP y Grupos DFP que se añaden en la parte inferior de la sección.

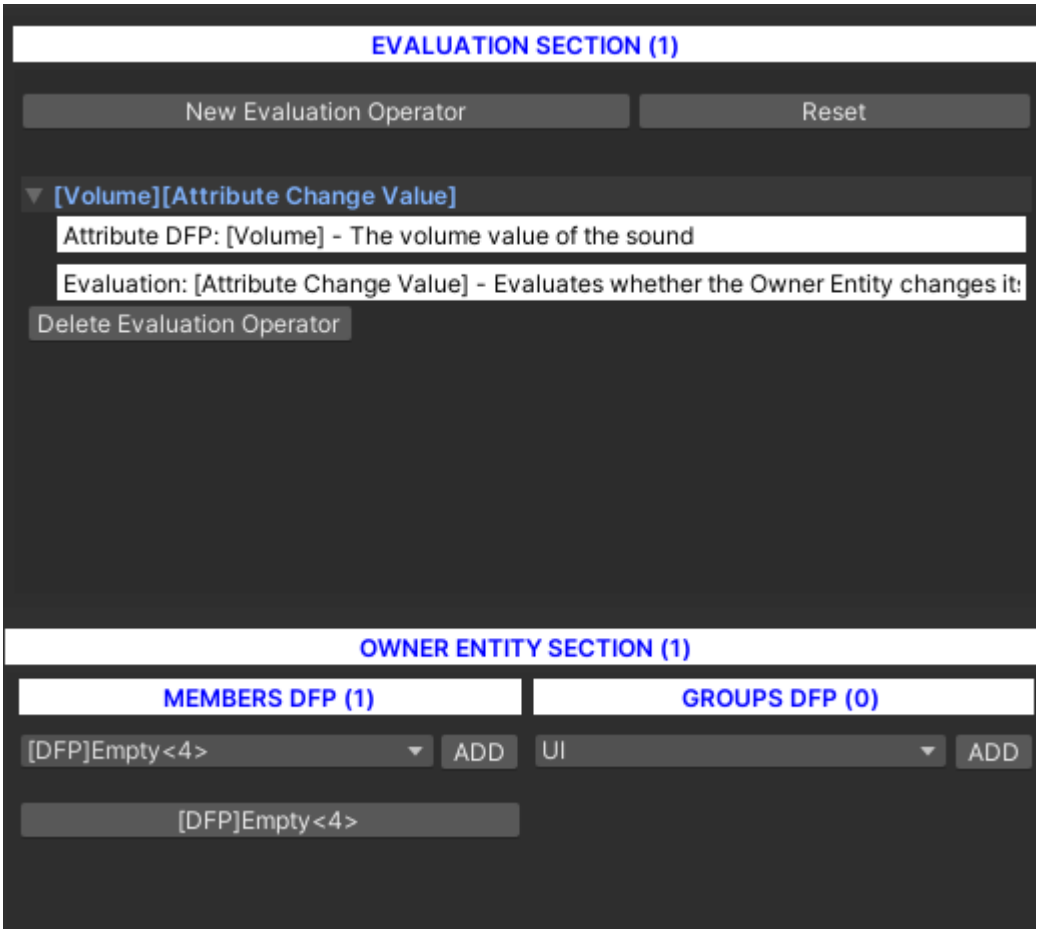


Ilustración 21 Trinket DFP

Funciones principales de los botones de la sección de Trinket:

- Using Limiter: Activa o desactiva el uso del limitador. Esta opción aparece cuando se introducen al menos 2 operadores de evaluación.

- Limiter: Valor numérico del limitador. Determina el porcentaje de operadores de evaluación que deben cumplirse para que se considere que se ha cumplido la evaluación de los operadores.
- New Evaluation Operator: Añade, si es posible, un nuevo operador para configurar una nueva evaluación.
- Reset: Borra todos los operadores creados en el trinket. No tiene confirmación.
- AddMember: Añade a la entidad propietaria el miembro que esté seleccionado en el DropDown de miembros. En esta lista aparecerán todos los miembros DFP que contenga la escena.

  AddGroup: Añade a la entidad propietaria el grupo que esté seleccionado en el DropDown de grupos. En esta lista aparecerán todos los Grupos DFP que contenga el Asset de gestión de grupos (Group Maker) que esté cargado.

Al hacer click en cualquiera de los Miembros DFP o Grupos DFP añadidos, se borrarán de la lista de la Entidad Propietaria.

5.5.4 Creación de un nuevo operador de evaluación

Al iniciar el proceso de crear un nuevo operador de evaluación con el botón "New Evaluation Operator" aparecerán nuevos botones (Ilustración 22):

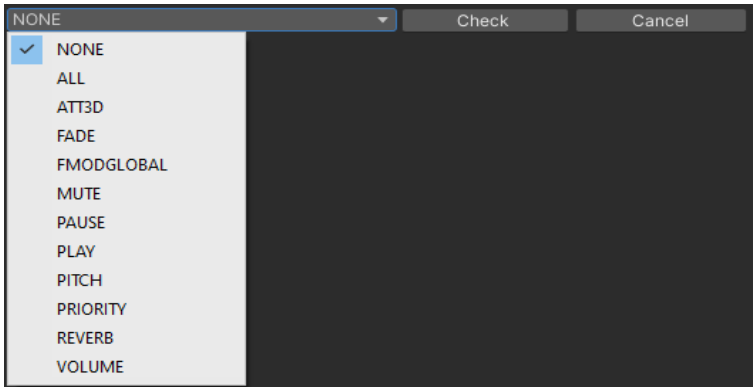


Ilustración 22 Atributos DFP

El listado nos muestra los distintos "Atributos DFP" con los que podemos construir evaluaciones. Cuando elegimos uno y pulsamos el botón "Check" nos mostrará un listado de evaluaciones posibles para el Atributo DFP seleccionado (Ilustración 23):

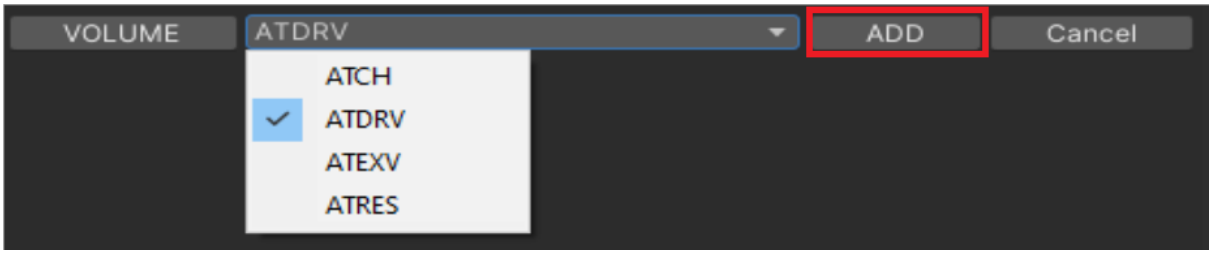


Ilustración 23 Evaluaciones de un atributo DFP

Al pulsar el botón ADD, el plugin intentará crear el operador de evaluación e irá creando un listado con los diferentes operadores que se mostrará en el editor (Ilustración 24).

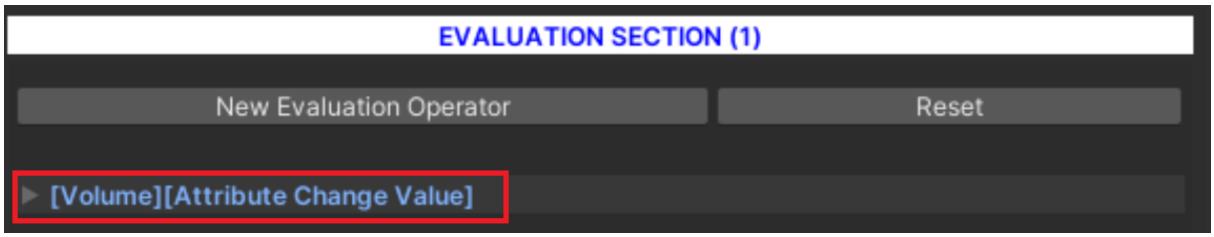


Ilustración 24 Operadores de evaluación de un atributo DFP

Los operadores se definen de forma automática con los nombres:

- [ATRIBUTO] [CÓDIGO OPERACIÓN EVALUACIÓN]

Cada operador tiene su propia sección de configuración y puede contener campos editables. Estos campos estarán agrupados y englobados dentro de su operador de evaluación separados por marcadores (Ilustración 25).

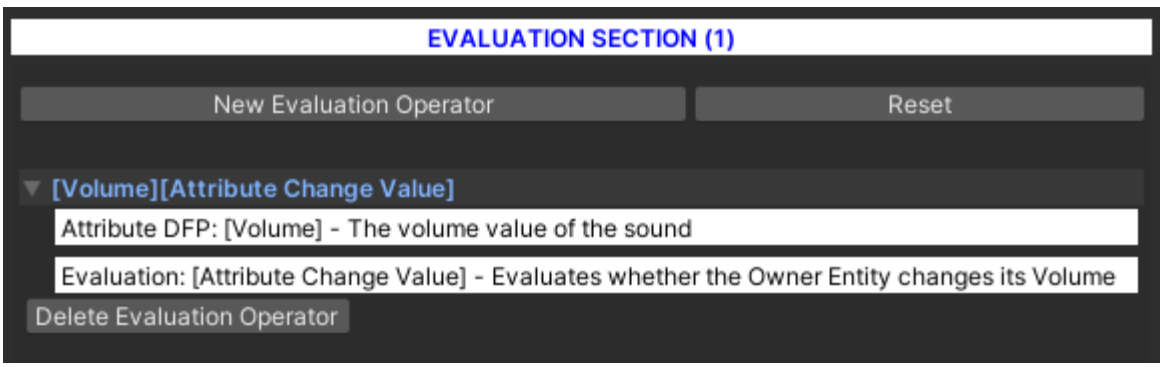


Ilustración 25 Configuración operadores de evaluación

En este ejemplo, el operador [VOLUME][Attribute Change Value] incluye la siguiente información y campos editables:

- Atributo DFP: VOLUME

- Evaluación: ATCH (Attribute Change Value), la evaluación del operador será válida cuando el valor del Atributo DFP cambie.
- Delete Evaluation Operator: Botón para borrar el operador de evaluación.

De forma global las Secciones nos van mostrando el recuento general de items que contienen. Evaluation Section indica que existe un operador de evaluación en el Evento DFP que se está tratando.

Al crear al menos dos Operadores de Evaluación dentro de la Sección de Evaluación, aparece el uso del Limitador.

Este limitador representa el % de Operadores de Evaluación que se deben de dar en un instante determinado para que se considere válida la Evaluación General (Ilustración 26).

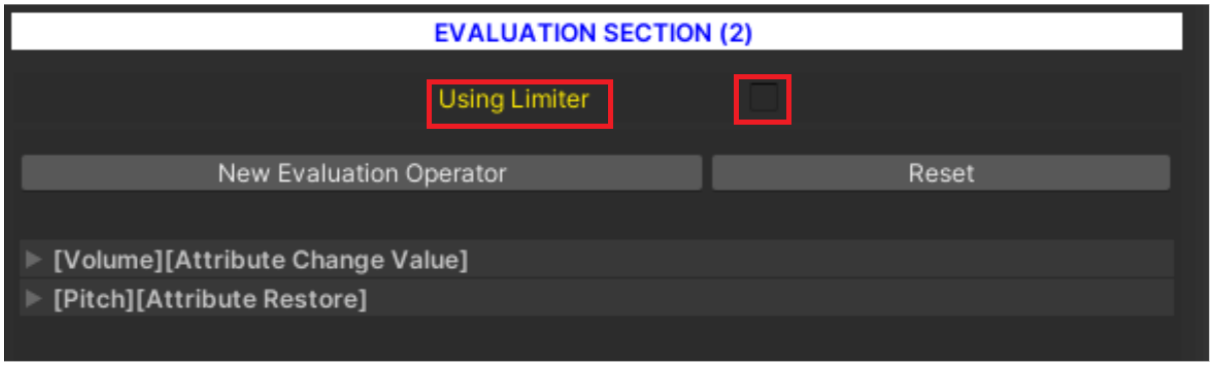


Ilustración 26 Limitador operadores evaluación

Al activar el limitador, la herramienta nos dará opción a introducir el valor del porcentaje del limitador (Ilustración 27).

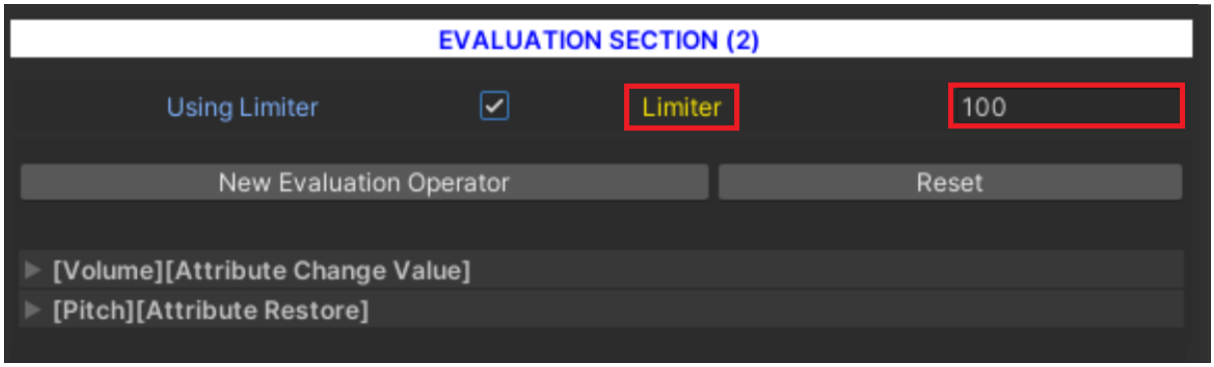


Ilustración 27 Configuración limitador operadores de evaluación

5.5.5 Creación de Entidad Propietaria

En la parte inferior del Trinket podemos observar la sección de la Entidad Propietaria. Como todas las secciones del creador de Eventos nos muestra un recuento global de los items que forman la sección (Ilustración 28).

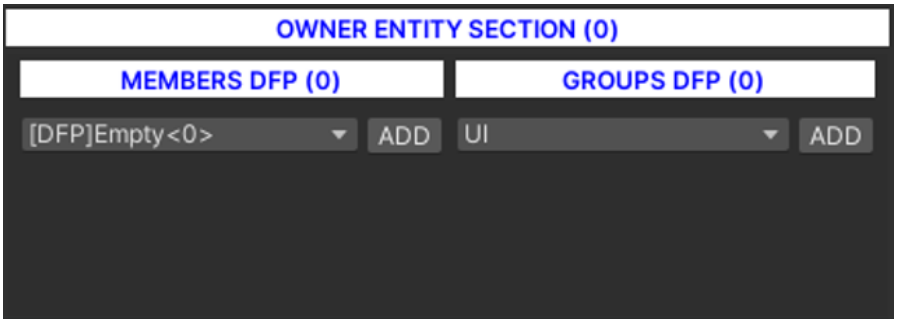


Ilustración 28 Entidad propietaria

Al pulsar cualquiera de los listados se abrirá un desplegable, uno con los Miembros y otro con los Grupos de la escena (Ilustración 29).

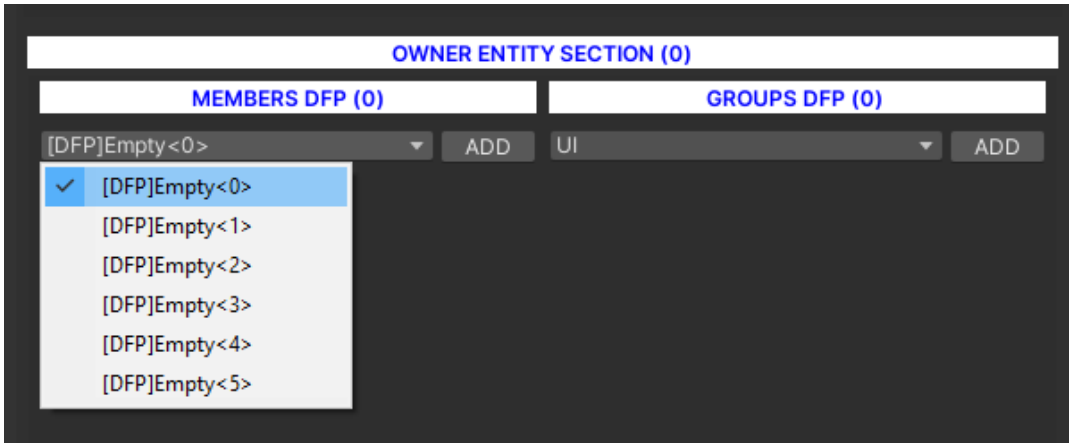


Ilustración 29 Opciones menú entidad propietaria

Al hacer click en cualquiera de ellos se seleccionará como primera opción en el DropDown. Si se activa el botón "ADD" se nos añade como parte de la Entidad Propietaria y la sección se actualiza (Ilustración 30):

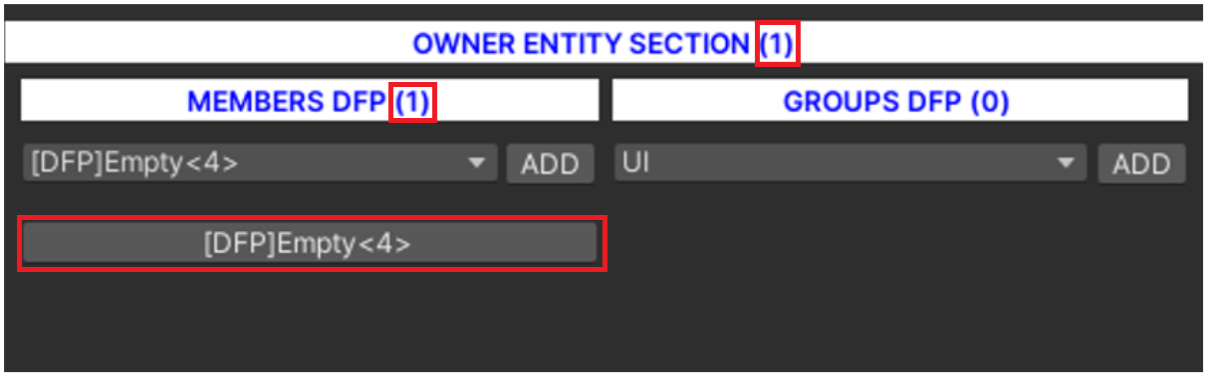


Ilustración 30 Selección de miembros

5.5.6 Sección Action DFP

Action se encarga de aplicar distintas acciones que se ejercen sobre la entidad objetivo cuando las condiciones de evaluación se han cumplido sobre la entidad propietaria. La entidad objetivo se define con los Miembros DFP y Grupos DFP que se añaden en la parte inferior de la sección. Las acciones se crean y configuran en forma de operadores, en la parte superior de la zona del Action (Ilustración 31).

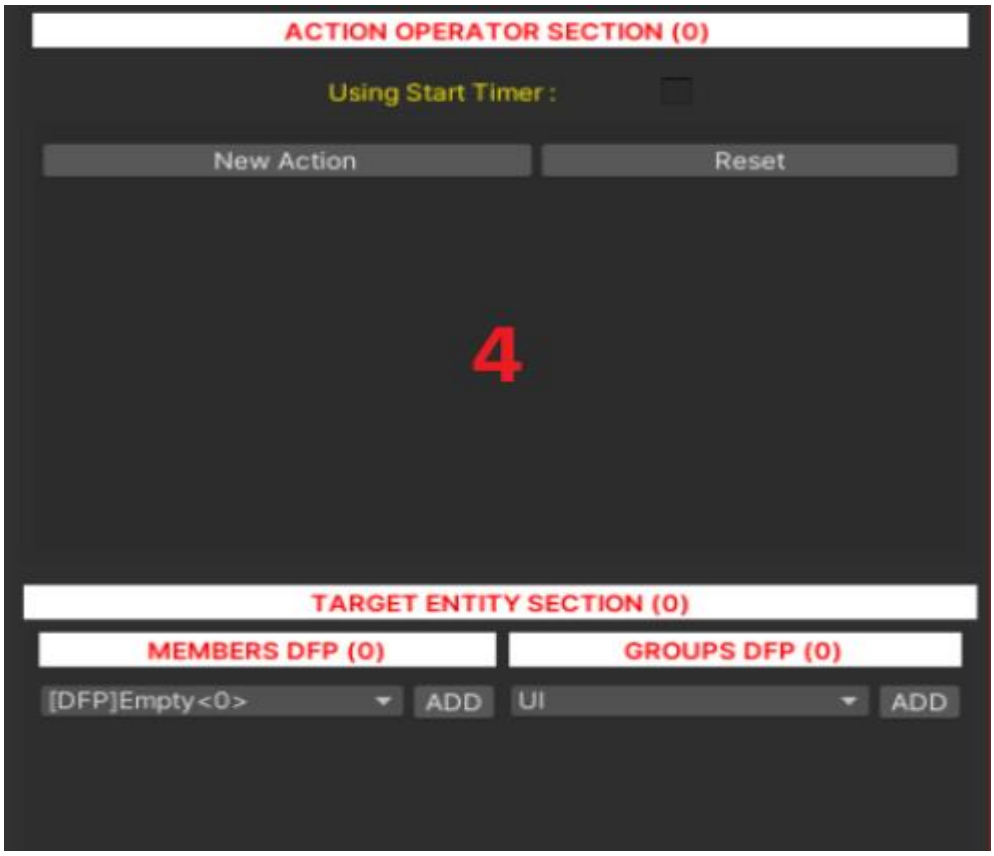


Ilustración 31 Sección Action

- New Action Operator: Añade, si es posible, un nuevo operador para configurar una nueva acción.
- Reset: Borra todos los operadores creados en el Action. No tiene confirmación.
- AddMember: Añade a la entidad objetivo el Miembro DFP que esté seleccionado en el DropDown de miembros. En esta lista aparecerán todos los Miembros DFP que contenga la escena.
- AddGroup: Añade a la Entidad objetivo el Grupo DFP que esté seleccionado en el DropDown de grupos. En esta lista aparecerán todos los Grupos DFP que contenga el Asset de gestión de grupos (Group Maker) que esté cargado.
- UsingStartTimer: Configura si se desea usar o no un timer global inicial.

Para todo lo demás, el uso de esta sección es similar a la sección central (3) del Trinket.

A medida que se vaya definiendo el Trinket (Operadores de Evaluación + Entidad Propietaria) y el Action (Operadores de Acción + Entidad Objetivo) el Evento DFP se irá haciendo más complejo (Ilustración 32):

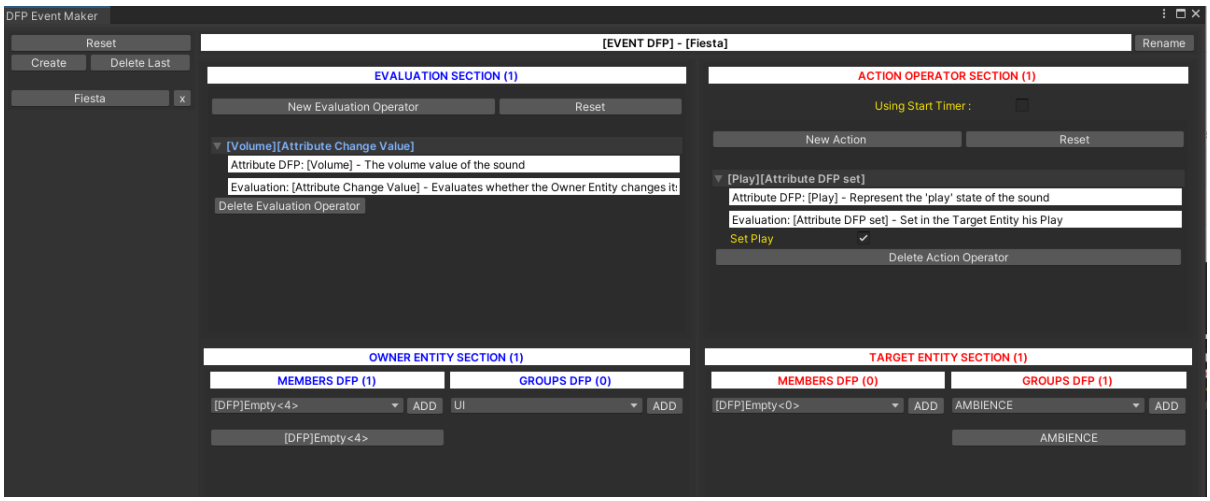


Ilustración 32 Evento DFP configurado

Capítulo 6 - Implementación

Esta sección pretende aclarar cómo funciona el plugin, por qué se han implementado las funcionalidades que tiene y para qué, y otras cuestiones de implementación, como, por ejemplo, las características y funciones en base al ámbito de trabajo.

6.1 Estructura general, directorios y funcionalidades

En términos de implementación el plugin está estructurado en cuatro directorios principales (Ilustración 33).

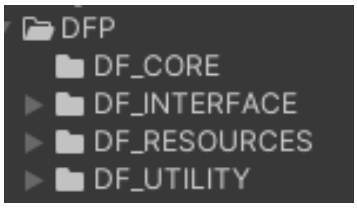


Ilustración 33 Directorios DFP

DF_CORE: Contiene el núcleo que gestiona todo lo relacionado con el plugin en tiempo de ejecución o RunTime.

DF_ INTERFACE: Cuenta con todos los scripts que dan soporte a las distintas interfaces: editores, editores extendidos, definición de serializables, etc.

DF_RESOURCES: Directorio que almacena los distintos recursos que usa el Plugin. Son directorios de recursos de uso RECOMENDADO, pero no obligatorio. El plugin instancia de forma predeterminada dentro de este directorio cualquier recurso auxiliar que necesite: Group Maker, Event Manager, etc.

DF_UTILITY: Directorio que almacena los scripts que definen y dan soporte a los distintos Actores o Herramientas del Plugin como Group Maker, State, Event Manager, etc.

Debido a la gran cantidad de código que ha requerido el desarrollo de este plugin, se ha implementado el uso de regiones (#region) en los scripts que ofrezcan una visión global de qué se puede encontrar dentro. Cada una de estas regiones contienen múltiples funciones que se pueden consultar en cada script del plugin y permiten obtener una visión global (Ilustración 34).

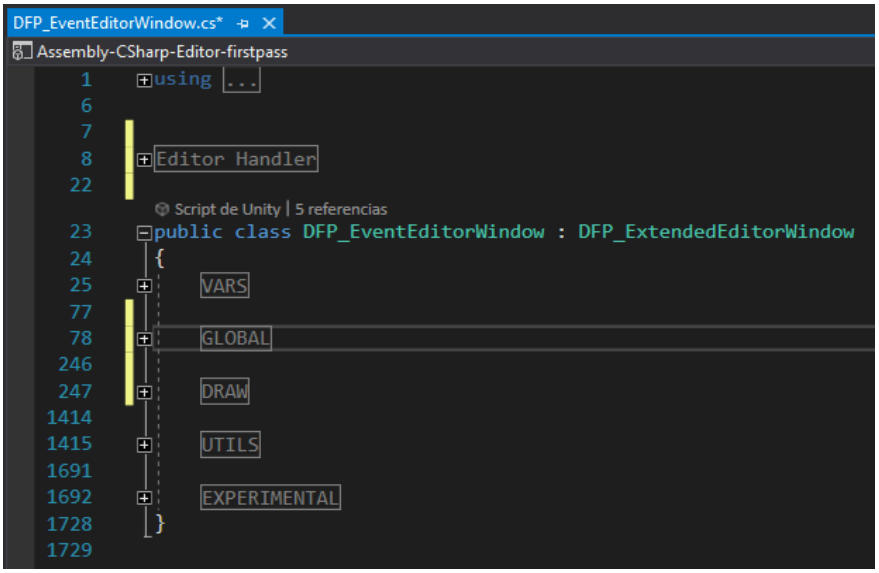


Ilustración 34 Regiones Editor Window

Para la correcta comprensión del Capítulo de Implementación, es necesario profundizar en algunos conceptos que se presentan en las siguientes secciones: "6.2 Ámbito de trabajo" y "6.3 Editores y ventanas de edición".

6.2 Ámbito de trabajo

Con "ámbito de trabajo" se hace referencia al tiempo y zona en la que interactúa cada una de las partes del plugin. En el uso de la plataforma de desarrollo de Unity se pueden diferenciar dos áreas de ejecución o de control por frames y ciclos de actualización: Editor Time y Run Time.

6.2.1 Editor Time y Run Time

Cuando se habla de EDITOR TIME, se hace referencia a todo lo relacionado al ciclo de ejecución del Editor (IDE) de Unity en tiempo de edición, entre los cuales se incluyen los ciclos de actualización con los que trabaja el plugin como OnGUI, OnInspectorGUI, OnEnable.. Está soportado por la API de "UnityEditor" y alimenta toda la parte de la interfaz de la plataforma de desarrollo. Aquí se construyen editores que ejecutan y extienden el propio editor de Unity.

Sin embargo, si se habla de RUN TIME, se hace referencia a todo lo relacionado al ciclo de ejecución de la escena, es decir, cuando la aplicación está en ejecución, por lo que incluye los ciclos de actualización como "Awake," "Start", "Update", etc con los que trabaja Unity. Está soportado por la API de "UnityEngine" y alimenta toda la parte del Backend en ejecución de la plataforma de desarrollo. Unity cataloga

como clases RunTime aquellas que heredan de la clase MonoBehavior y se ejecutan durante el lanzamiento de la escena o "gameplay".

En general las funcionalidades de las herramientas DFP de Editor Time están centradas en el desarrollo mientras que las herramientas DFP de Run Time están centradas en la gestión, administración y control de los componentes DFP en ejecución, además de la comunicación de estos.

Además, algunas herramientas cuentan con versiones para el otro ámbito. Por ejemplo, la herramienta Event Maker es la encargada de la creación de eventos en Tiempo de Edición o Editor Time mientras que Event Manager es la herramienta encargada del control de eventos en Run Time.

Esto está hecho de forma intencionada. La mayor parte de la carga del Plugin se centra en el ámbito de la edición, es decir, en trabajar en Editor Time. Trasladar toda esa carga de código a Run Time ralentizaría demasiado el procesamiento del plugin. Además podría provocar que se rompan fácilmente las interacciones en los ciclos de actualización de Unity por posibles llamadas a funciones que pertenecen al otro ámbito de trabajo.

Por ejemplo, no tiene sentido utilizar el Event Manager en Editor Time por el mero significado que tiene la herramienta. Esta está enfocada a dar soporte única y exclusivamente a Eventos DFP en Run Time, pretender darle uso en Editor Time sería alterar su propia naturaleza. Por otro lado, tampoco tendría sentido usar el Menú Principal DFP (DFP Main Window) en Run Time ya que cuenta con funciones de actualización que implican llamadas a funciones propias del sistema de editores de Unity, esto provocaría cambios de ámbito mientras se ejecuta la escena, lo cual dejaría la reproducción de la escena inservible tras corromperse.

Por estos motivos el plugin cuenta con una fase de "descarga" en el paso de Editor Time a Run Time. En esta fase de descarga hay determinados Actores DFP que tras pasar a Run Time eliminan todos sus componentes de edición, ya que en ejecución no son necesarios y resultaría inútil mantenerlos en memoria, por ejemplo, la herramienta DFP_ToolbarEditor para Miembros DFP o la herramienta de Creación de Eventos DFP.

Esta fase de descarga dificulta el tratamiento de datos y la coordinación de las herramientas pero agiliza el procesamiento del plugin y el coste en memoria en

tiempo de ejecución por lo que las herramientas o editores que tenían un uso exclusivo para Editor Time se "liberan".

Aunque existen otros editores con roles más secundarios, en la siguiente imágen se muestra la jerarquía principal de Editores DFP y Ventanas de Edición DFP (Ilustración 35).

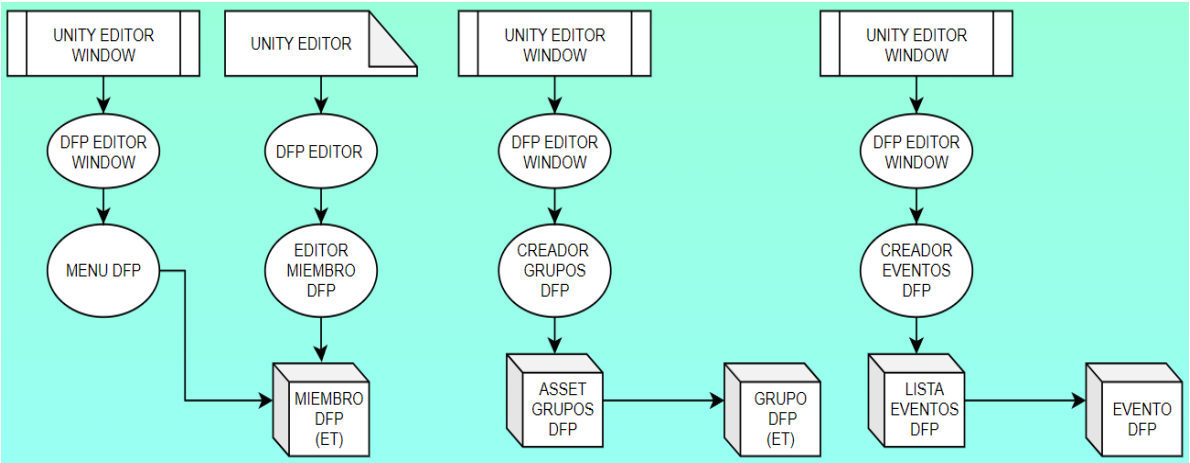


Ilustración 35 Jerarquía de editores y ventanas de edición DFP

En esta jerarquía podemos observar como las clases de Unity son la base de la red de herencia DFP, seccionada en Editores (DFP EDITOR) y Ventanas de Edición (DFP EDITOR WINDOW). Debajo de ellas se encuentran las principales Herramientas DFP como el Menú Principal DFP, el creador de Eventos DFP (Event Maker) o la herramienta que gestiona los Assets de Grupos DFP (Group Maker).

6.2.2 Herramientas Editor, Run e Híbridas

Las herramientas del Plugin se pueden clasificar en función al Ámbito de Trabajo que atienden. Además de disponer de herramientas y funcionalidades para Editor y Run Time, cuenta con herramientas híbridas, es decir que trabajan en los dos ámbitos de trabajo. Estas últimas se deben implementar con especial cautela ya que pueden provocar errores en los ciclos de actualización que podría corromper el editor o la misma plataforma de desarrollo.

Existen dos tipos de herramientas DFP que son híbridas dentro del plugin DFP. Esta distinción se debe a que no todas las herramientas cuentan con editores y esto hace que puedan ser utilizadas en Run Time, esta característica cataloga a las herramientas híbridas en:

Híbrida MonoScript: Se refiere a aquella herramienta que atiende a ambos ámbitos de trabajo de la plataforma de desarrollo y generalmente no cuenta con Editor DFP o Ventana de Edición DFP por lo que consta de un solo script.

Híbrida MultipleScript: Se refiere a aquellas herramientas que atienden a ambos ámbitos de trabajo de la plataforma de desarrollo y cuentan con al menos un Editor DFP o una Ventana de Edición DFP, es decir, cuentan al menos con un script que ofrece las funcionalidades de la herramienta en Editor Time, al menos un script que ejerce la función de Editor o Ventana de edición y que ofrece las funcionalidades de la interfaz visual y al menos, un script que ofrezca las funcionalidades de la herramienta en Run Time. Esto provoca que se necesite una fase de liberación de componentes o de puesta en pausa de la parte que atiende a Editor Time cuando la escena se ejecuta para evitar mantener en ejecución componentes innecesarios que puedan estar ocupando memoria o ejecutando ciclos de actualización innecesarios que consuman tiempo de procesamiento.

6.3 Editores y ventanas de edición

DFP no sería posible sin el soporte de las clases "Editor" (3) y "EditorWindow" (4), una grandísima parte de DFP son interfaces que se presentan a Unity como una extensión de sus propios editores.

Unity - Scripting API: Editor

"Editor: Derive from this base class to create a custom inspector or editor for your custom object." - Unity

Unity - Scripting API: Editor WIndow

"Editor Window: Use this class to create Editor windows that can either float independently or dock as tabs, similar to the default windows in the Unity Editor." - Unity.

Los editores se encargan de "Customizar" clases. A efectos prácticos esto quiere decir que como mínimo se necesita:

- Una clase base que sea el objeto de la "customización" o personalización.
- Una clase que "customice" o personalice la forma en la que se serializan y se muestran los datos en el editor.

Existen numerosas webs que ofrecen consejos o tips para abordar el desarrollo de la customización _(7). Aquí podemos ver como la barra de herramientas de los Miembros DFP es DFP_MemberInspectorToolBarEditor, esta customiza datos de tipo (typeof) DFP_MemberEditorTime (Ilustración 36).

```
[CustomEditor(typeof(DFP_MemberEditorTime))]
Script de Unity | 0 referencias
public class DFP_MemberInspectorToolBarEditor : DFP_ExtendedEditor
```

Ilustración 36 Customización de editor DFP

Cuando una clase es se utiliza para "customizar a otra" se presenta a Unity como una especie de plantilla con directrices que orquesta el editor que están customizando. Por ejemplo, aquí podemos observar la diferencia entre que una clase esté auditada por un Editor DFP o no.

Visión de un Miembro DFP en Inspector sin su Editor (Ilustración 37).

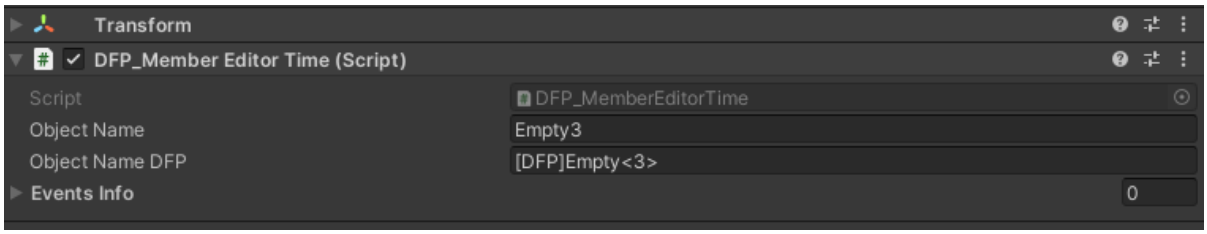


Ilustración 37 Miembros DFP sin su editor personalizado

Y aquí un Miembro DFP que cuenta con su Editor DFP que además añade secciones completamente nuevas que aparentemente no existen en la imagen superior, ya que son funcionalidades extras que añade el editor personalizado (Ilustración 38) (6).

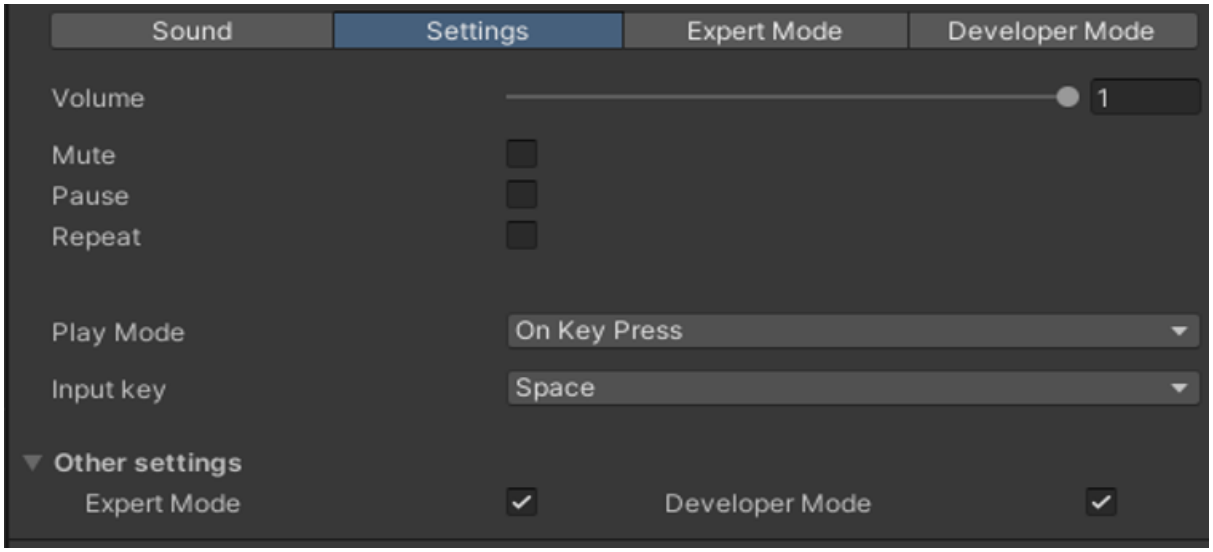


Ilustración 38 Vista de miembro DFP con editor personalizado

Se plantea una interacción muy interesante porque se ofrece libertad al desarrollador a diseñar una clase que tenga sus propias funcionalidades, relacionadas al tratamiento de los datos detrás de la interfaz y a otra clase, centrada en cómo esos datos se van a tratar y a visualizar en la interfaz visual. Esto lo puede convertir en un proceso tremendamente complejo por las capas que hay que tratar.

Por ejemplo, hay algunas herramientas DFP que cuentan con una clase base que está customizada por un Editor con sus propias funcionalidades. Ese editor cuenta con botones que inician Ventanas de Edición que extienden el Editor inicial por lo que hay que tener en cuenta varias capas de actualización.

Además, las clases Editor y Editor Window de Unity son clases que permiten el desarrollo customizado de editores y ventanas de edición gracias a estructuras auxiliares como Editores de Layouts _(8) o GuiLayouts_(9) para la inicialización de componentes de editores. Además, tienen la peculiaridad de que pueden trabajar de forma independiente o convertirse en una parte más de Unity, insertándose como pestañas o como ventanas conexas en barras de herramientas.

Tienen el grandísimo potencial de que comparten las directrices de actualización de las interfaces de Unity y además, al instanciar cualquiera de ellas pasa a ser parte del sistema de actualización de la plataforma.

Todo esto nos ofrece flexibilidad de desarrollo a la vez que garantizamos que el uso de nuestras interfaces sea rápido y obtengan la misma prioridad de

actualización que cualquier ventana de trabajo de Unity, aspecto muy importante en la sincronización de datos.

Unity nos facilita con este ejemplo entender cómo los editores se actualizan (Ilustración 39), aspecto que se detalla más adelante.

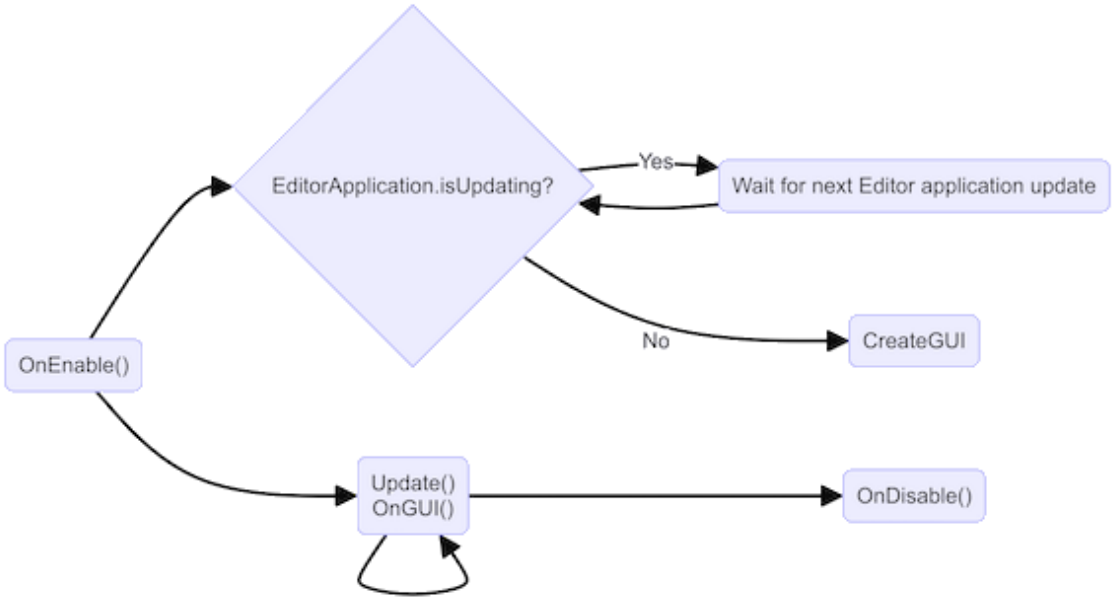


Ilustración 39 Ciclos de actualización editores de Unity

6.3.1 Editores y ventanas de edición extendidas

Como se ha comentado, las interfaces DFP se soportan en las clases Editor y EditorWindow de Unity. Tras hacer un análisis profundo, se decide implementar un sistema de herencia que ofrece una API con funcionalidades comunes a los editores y ventanas de edición (Ilustración 40) (10).

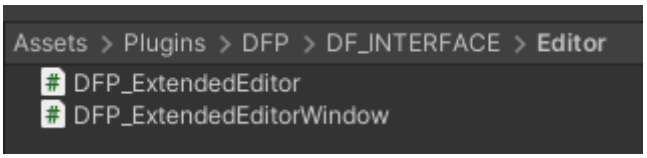


Ilustración 40 Interfaces DFP

Este sistema de herencias convierte a todas las clases que hereden de ellas en "Editores Extendidos" y "Ventanas de edición extendidas", más comúnmente conocidos como "Extended editors" y "Extended Editor Windows".

Con esto conseguimos que Unity entienda que todos los editores y ventanas de edición del plugin son parte del sistema de editor nativo. Esto agiliza los procesos

de actualización y además permite que puedan adaptarse y flexibilizarse como cualquiera de los editores de Unity.

DFP_ExtendedEditor: Ofrece soporte a la creación de editores que trabajan como componentes, es decir, editores de componentes o editores personalizados para el inspector.

DFP_ExtendedEditorWindow: Ofrece soporte a la creación de ventanas de edición de cualquier formato. Son utilizadas para crear menús, interfaces de edición, editores de modificación, etc (Ilustración 41).

```
    Script de Unity | 4 referencias
6   public class DFP_ExtendedEditorWindow : EditorWindow
7   {
8       private bool _editorGUIisEndHorizontal;
9       private bool _editorGUIisEndVertical;
10      private bool _editorGUIisEndChange;
11      private int _indentLevel;
12
13      protected bool isEditorWindowReady;
14      protected string selectedPropertyPath;
15      protected bool isDFPExtendedEditorWindowReady;
16
17      #region DFP BASE EDITOR
18      protected DFE_EICache _EICache;
19
20      protected SerializedObject _target_srlzOBJ;
21      protected SerializedProperty currentSrlzPPT;
22      protected SerializedProperty selectedSrlzPPT;
        18 referencias
23      protected bool IsEICacheReady()...
        1 referencia
29      private bool LoadOrCreateEICache()...
        3 referencias
67      protected void UpdateTargetSrlzOBJ()...
        5 referencias
74      protected void ApplyModifiedPropsTargetSrlzOBJ()...
79
80      UTILITY
223
224     EDITOR SHORTS
325
326     DRAW METHODS
436     #endregion
```

Ilustración 41 Vista regiones ExtendedEditorWindow

Los Editores Extendidos cuentan con distintas regiones de soporte al desarrollo de Editores:

Utility: Funcionalidades de utilizad general para la creación y desarrollo de futuros editores

Editor Shorts: Son funcionalidades de creación centradas en editores, como métodos de indentación, control de estilos, de estructuras que se repiten, etc.

Draw Methods: Funcionalidades de dibujado de interfaz, ofrecen soporte para facilitar el proceso de dibujado y de serialización de datos en los editores.

Al igual que los Editores Extendidos, las Ventanas de Edición Extendida cuentan con sus propias regiones de soporte al desarrollo de Ventanas de Edición.

Cuando uno de estos "Extended Editors" (5) o "Extended Editor Window" se especializa en una funcionalidad concreta, se reconoce cómo "versión personalizada", quedando su clasificación como:

- Editor Extendido Personalizado (Personalized Extended Editor)
- Ventana de Edición Extendida Personalizada (Personalized Extended Editor Window)

Por ejemplo, el editor de Miembros DFP es un Editor Extendido Personalizado y la herramienta de creación de Grupos es una Ventana de Edición Extendida Personalizada.

Además, Unity distingue las clases que trabajan en Editor Time de una forma especial y nos recomienda que usemos dichas clases dentro de directorios "Editor" ya que esto facilita a la plataforma identificar rápidamente las estructuras para el mapeo interno (assembly).

Ante esta situación se puede proceder de dos formas:

1.- "Editor" directorio general

Crear un directorio general y almacenar ahí todos los editores, esto nos permite controlar todos los editores desde un mismo directorio y ayuda a su

identificación, pero dificulta el acceso relacional a otros scripts, como, por ejemplo, la identificación de scripts que usan dichas clases, assets relacionados, etc.

2.-Añadir un directorio "Editor" en cada sitio que se requiera

Añadir un directorio "Editor" en cualquier directorio nos dificulta identificar los editores ya que se encuentran desperdigados por los distintos directorios, pero nos ayuda a relacionarlo con el contenido desarrollado a través de ellos.

Se analizaron los editores de algunos plugins, entre ellos los de Fmod y se decidió apostar por la segunda opción por las ventajas de desarrollo que ofrece ya que este aspecto no es relevante para el usuario final.

Crear un directorio "Editor" también permite trabajar a "intelly-sense" y tomar rápidamente acciones como la de mapear los editores y crear una "vista" de edición al abrir los scripts de nuestro proyecto en un IDE como Visual Studio (Ilustración 42).

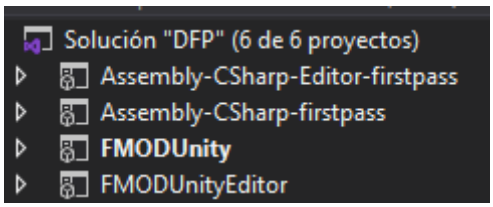


Ilustración 42 Vista resumida proyectos en VIsual

Este mapeo se consigue gracias a los directorios "Editor" creados en el proyecto (Ilustración 43, Ilustración 44 e Ilustración 45):

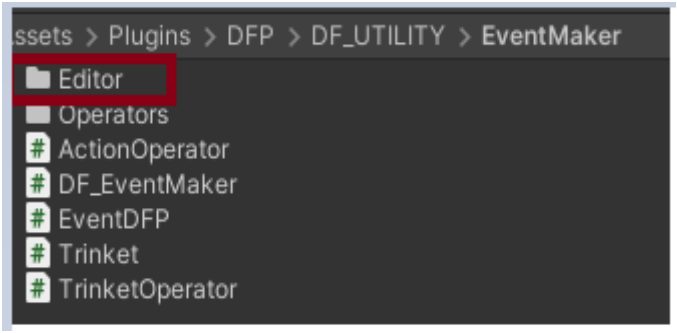


Ilustración 43 Directorio "Editor" para creación de editores

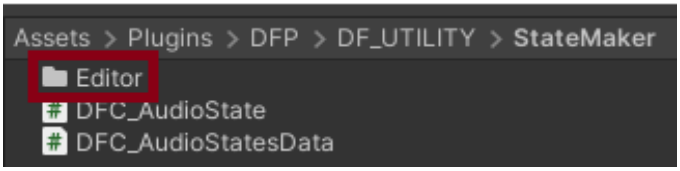


Ilustración 44 Directorio "Editor" para creación de editores

Ilustración 45 Directorio "Editor" para creación de editores

Gracias al mapeo y a la vista de edición se ofrece al usuario una vista general de desarrollo de todos los editores. Además, se convierten los editores en scripts "assembly" independientes (Ilustración 46).

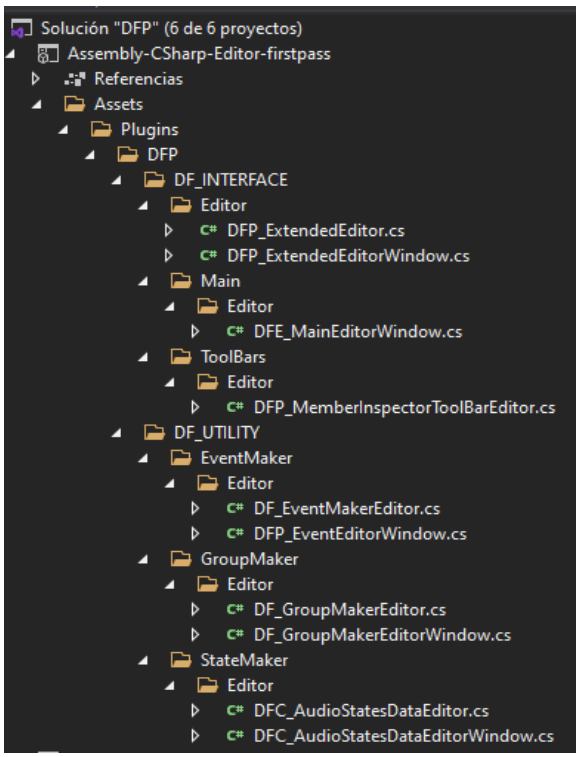


Ilustración 46 Vista Assembly Editores DFP

Esto ofrece un mayor rendimiento en la compilación de nuestras escenas, aislando el contenido de los editores.

Los "Assembly / firstpass" contienen todos los scripts compilados en "primer paso".

Entre otras cosas, esto significa que se puede controlar cuando Unity crea y compila los scripts desde nuestro editor, en primera instancia, se crea un enlace de actualización que hará que Unity entienda que queremos actualizar la compilación de nuestros scripts assets cuando se traiga de nuevo Unity a primer plano y exista al menos un cambio registrado.

A continuación, se tratan las características principales con las que cuentan los Editores y Ventanas DFP.

6.3.2 Editores y ventanas DFP. Flexibilidad

Uno de los conceptos fundamentales que se busca potenciar en el diseño del plugin es la sencillez de uso para que pueda usuarios de cualquier nivel puedan utilizarlo y su flexibilidad en el entorno de trabajo.

Cada persona tiene sus propios gustos y manías, cada uno tiene una predilección por cómo coloca su espacio de trabajo. Buscar una facilidad de uso sin tener en cuenta la comodidad en el entorno de trabajo sería como intentar ganar una maratón atándose voluntariamente los pies. Se ha enfatizado durante toda la documentación en la intencionalidad con la que los entornos de las herramientas del plugin han sido diseñados. La adaptabilidad del Menú Principal DFP, la disposición de los elementos de los Editores en general, cómo interactúan unos con otros, etc.

Gracias al sistema de herencia en los Editores y Ventanas de Edición Extendidas DFP, se adquieren funcionalidades que flexibilizan al usuario su adaptación al entorno de trabajo. Esto es una de las grandes razones de peso por las que se apuesta por el sistema de extensión de editores y ventanas y la centralización de las funcionalidades para estos.

Entre otras funcionalidades, las herramientas DFP se pueden comportar como pestañas incrustables en las interfaces de Unity. Si se intenta arrastrar la ventana desde la pestaña, se comportará como interfaz incrustable mientras que si se arrastra desde la barra trasera de la pestaña, se comportará como ventana independiente.

Además, al intentar arrastrar una ventana dentro de otra, podremos hacer uso del adaptador inteligente que nos dará soporte en la creación de nuevas disposiciones para nuestras interfaces.

En la siguiente imágen se puede apreciar como el Menú Principal DFP (Main Window) queda incrustado en el propio panel de Proyecto de Unity.

También pueden componerse ventanas de trabajo independientes y totalmente nuevas. Por ejemplo, en la siguiente imagen se muestra cómo varias de las herramientas se integran en una sola ventana. Se puede observar en el recuadro rojo que solo existe un control de Ventana (Ilustración 47)

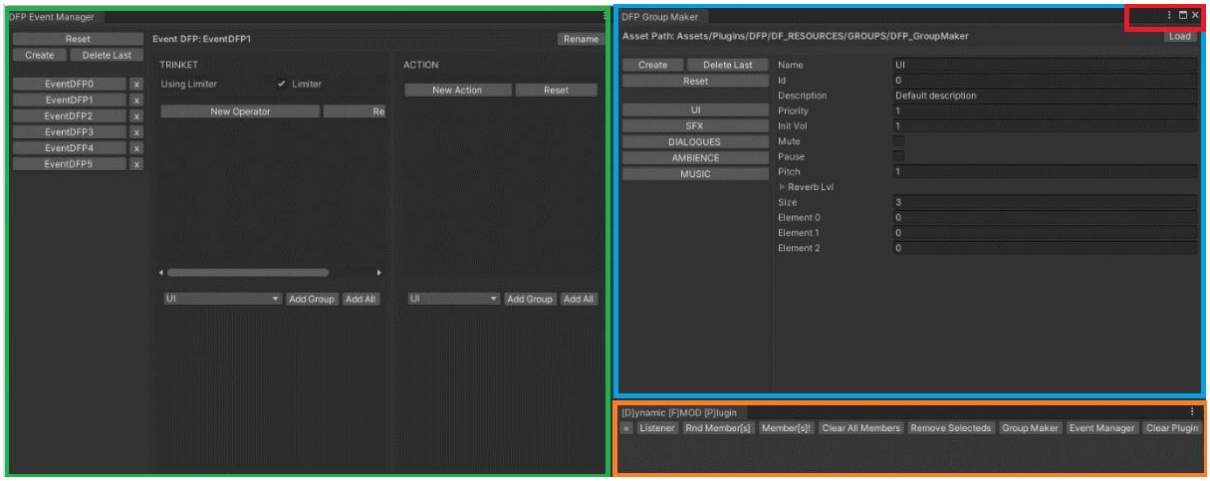


Ilustración 47 Integración de herramientas

También puede utilizarse la zona de inspector. En la siguiente ilustración (Ilustración 48) se puede observar al Inspector de Unity. Este muestra un objeto que cuenta con un componente Transform y además está sonorizado con el plugin por lo que cuenta con el script DFP_EditorMember. En la parte inferior del inspector se ha incrustado el Event Maker mientras que en la zona izquierda se ha puesto como herramienta conexa el Group Maker y el Menú Principal DFP (Main WIndow) se encuentra como parte de la sección de Proyecto del editor de Unity.

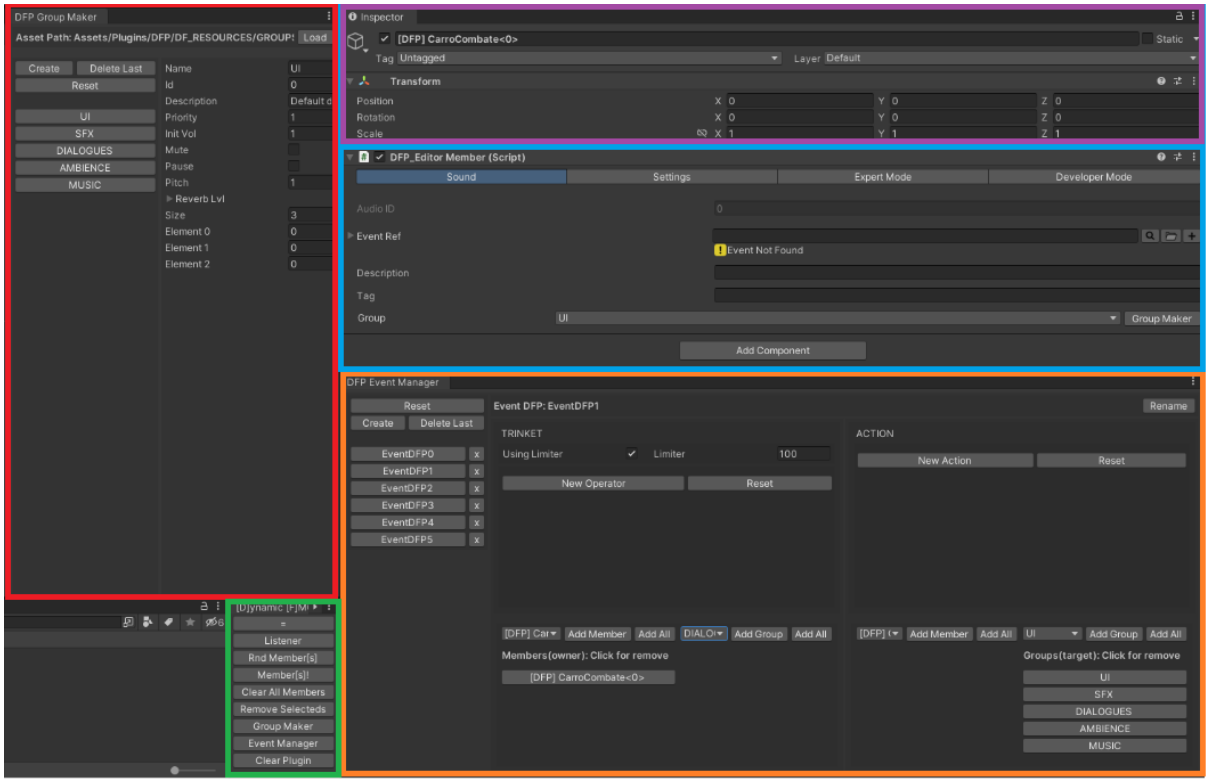


Ilustración 48 Composición herramientas inspector Unity

Queda en manos del usuario decidir cómo gestionar sus diferentes interfaces visuales y elegir dónde ubicarlas.

6.3.3 Editores y ventanas DFP. Tratamiento optimizado de datos.

DFP en esencia es un "constructor" (12). Mientras que el usuario le da uso y utiliza sus herramientas, crea nuevos conjuntos de datos que se deben ir almacenando.

En función del tipo de datos que se utilice, se tratan de una manera o de otra.

[System.Serializable] es un atributo de C# que se puede utilizar y aplicar a clases, estructuras o definición de campos para indicar que pueden ser serializados por el sistema de serialización (13) de Unity.

Esto transforma los datos para que puedan ser reubicados de una forma mucho más efectiva en forma de ficheros con formato de texto, xml, json, ficheros de tipo .assets o estructuras de tratamiento de datos serializados para los editores de Unity, por ejemplo (14).

Sin embargo, la serialización nativa del sistema de Unity nos restringe de forma inicial al uso de un conjunto de tipos de datos ligeramente reducidos, como: atributos básicos - int, float, etc-, arrays, clases de ISerializable y colecciones como listas o diccionarios (Ilustración 49).

```
[System.Serializable]
Script de Unity (2 referencias de recurso) | 41 referencias
public class DFP_MemberEditorTime: MonoBehaviour
{
    #region TOOLBAR-TAB: 1-1 "SOUND"
    [HideInInspector] [SerializeField] private FMOD.Studio.EventInstance _eventInstance;
    [HideInInspector] [SerializeField] private EventReference _eventRef;
    2 referencias
    public EventReference GetEventReference()
    {
        return _eventRef;
    }
    [HideInInspector] [SerializeField] private string _eventDescription;
    [HideInInspector] [SerializeField] private int _dfpMemberID;
    [HideInInspector] [SerializeField] private string _tag;
    [HideInInspector] public int _groupID;
    #endregion
```

Ilustración 49 Serialización MonoBehaviour

En DFP se hace uso de la serialización de todo tipo de datos, y también se usa la reubicación en ficheros .asset gracias al soporte de la clase "ScriptableObject" (15). Se convierte el fichero .asset en una especie de "fachada" y contenedor para el script que además se puede utilizar en la serialización de datos a través de Editores y Ventanas Extendidas. Unity nos dice:

The main use cases for ScriptableObject are:

- Saving and storing data an Editor session
- Saving data as an Asset in your Project to use at run time

Esto lo convierte en una herramienta perfecta para trabajar con DFP, que trabaja en ambos ámbitos de trabajo.

```
[CreateAssetMenu(fileName = "DFP_GroupMaker", menuName = "DFP/GroupMaker")]
Script de Unity | 13 referencias
public class DF_GroupMaker : ScriptableObject
```

Ilustración 50 ScriptableObject

Aunque inicialmente se trató de serializar en ficheros .asset todos los datos con los que trabajaba DFP, no tiene sentido en el contexto que se está tratando serializar en ficheros externos datos propios y únicos de una escena, como pueden ser los Miembros DFP y Eventos DFP.

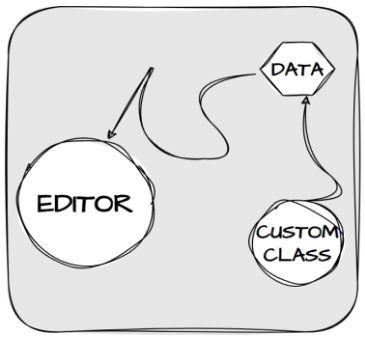


Ilustración 51 Serialización tipo 1

En función de cómo se quieran serializar los datos hay que darle un tratamiento u otro. En el plugin DFP se serializan los datos de tres maneras diferentes.

En la primera, la clase que se utiliza como objeto de la customización se trata como un objeto serializado y se manda empaquetado con todas sus propiedades, que deben ser de acceso público o contar con funciones que ofrezcan dicho acceso.

Otra de las formas que tiene el plugin de serializar los datos es mediante el acceso a propiedades serializadas a través de "tuberías" o "pipes" de edición. Se trata a los Editores como "Plantillas" que definen "huecos" de elementos y estos "huecos" se enlazan con elementos que se definen en forma de atributos, generalmente (Ilustración 53).

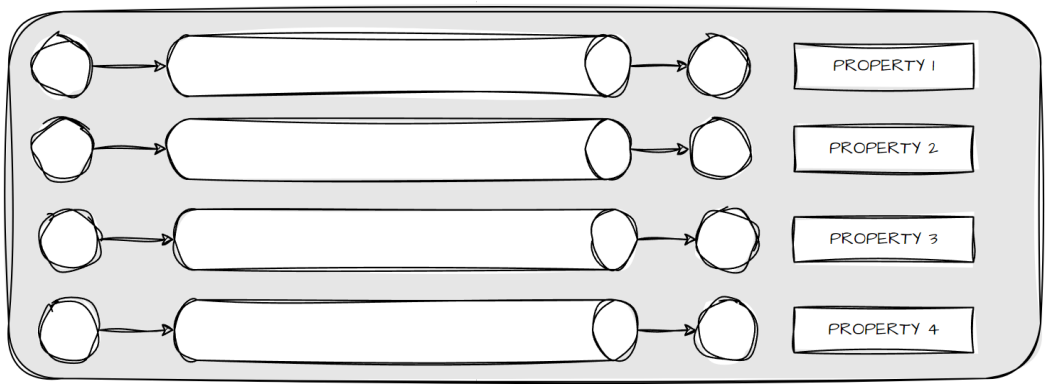


Ilustración 52 Serialización tipo 2

Se crea una tubería de enlace a cada una de las propiedades del objeto que se quieren serializar. Una propiedad puede representar un atributo, por ejemplo.

```
private void LoadSoundProperties()
{
    //_eventPath_srlzPPT = LoadPropertyByFind("_eventPath");
    _eventInstance_srlzPPT = LoadPropertyByFind("_eventInstance");
    _eventRef_srlzPPT = LoadPropertyByFind("_eventRef");
    _eventDescription_srlzPPT = LoadPropertyByFind("_eventDescription");
    _dfpMemberID_srlzPPT = LoadPropertyByFind("_dfpMemberID");
    _tag_srlzPPT = LoadPropertyByFind("_tag");
    _groupID_srlzPPT = LoadPropertyByFind("_groupID");
}
```

Ilustración 53 Enlaces de propiedades serializadas

Este tipo de conexiones son las más rápidas dentro de los editores de Unity pero está diseñada para programación de backend profundo ya que esas referencias que se cargan son los nombres textuales de las variables que se declaran y que se acceden a ellas a través del assembly de Unity. Es decir, si se renombra una variable, como eso se está referenciando a través de un "string", el valor del "string" no reconoce que se está cambiando la variable y se rompen las conexiones con las tuberías de serialización.

Este tipo de serialización de datos está muy bien para trasladar versiones ya completas de editores de manera que podemos agilizar los procesos de tratamientos de datos en la actualización global de los editores de Unity.

Por último se usa un tipo de serialización de datos que exporta estos datos a ficheros, en el caso del Plugin DFP, de tipo *.asset. . Son procesos de serialización controlados por script (16) que no tiene por qué ser un editor.

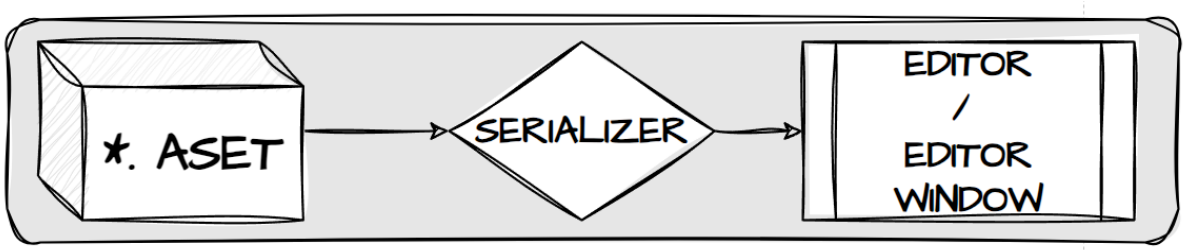


Ilustración 54 Serialización tipo 3

El editor se presenta como una herramienta que serializa e invierte el proceso de serialización de clases tipo ScriptableObjects. La mayoría de estos editores suelen comportarse como "pintores" de propiedades o "Property Drawers" (11).

6.3.4 Editores y ventanas DFP. Sincronización de datos.

Trabajando con ficheros .asset de tipo scriptable y haciendo uso del poder de los editores y ventanas extendidos se puede lograr una comunicación fluida entre Editor Time y Run Time. Al ser objetos serializados que pueden ser personalizados, se pueden manipular en Editor Time, a la vez que, por ser un contenedor único, se puede usar como almacén o gestor de datos configurables en Run Time.

Como se puede observar en la sección de Editores y Ventanas de Edición en DFP existen múltiples editores que manipulan información que está relacionada o compartida.

Esto puede provocar que el usuario vea información incoherente ya que todos los editores están preparados para que sean funcionales al mismo tiempo de desarrollo. Esto requiere una comunicación constante en cada ciclo de actualización, siempre que se registre algún cambio.

Los editores DFP están preparados para que cada cambio que se realice en la escena se actualice automáticamente en todos sus editores abiertos. Algunos de estos cambios son gestionados a través de los propios editores y otros a través de sistemas auxiliares como consultas a la Caché o los Managers.

Por ejemplo, si tenemos el Editor de Eventos DFP (Event Maker) abierto y hacemos uso de la Ventana Principal DFP (Main Window) para crear nuevos Miembros DFP, estos cambios se verán automáticamente reflejados en las listas de los usuarios de los eventos.

Ejemplo: Listado de Eventos DFP y Miembros DFP en el momento de la captura (Ilustración 55).

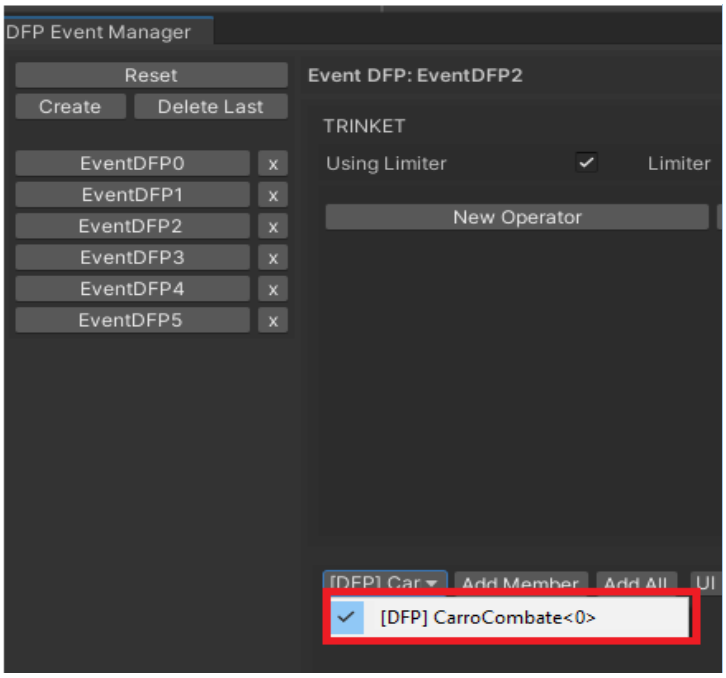


Ilustración 55 Listado de Eventos y Miembros DFP

Ejemplo: Objetos aleatorios creados en escena y convertidos en Miembros DFP (Ilustración 56).

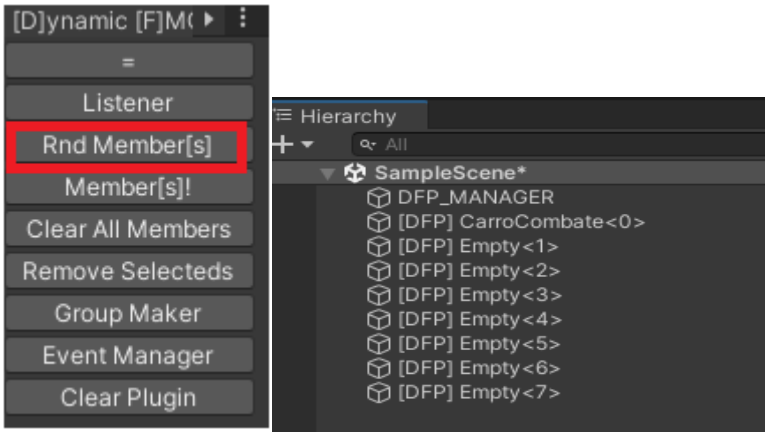


Ilustración 56 Objetos aleatorios

Ejemplo: La lista de los Miembros DFP en el listado de los eventos se actualiza automáticamente (Ilustración 57).

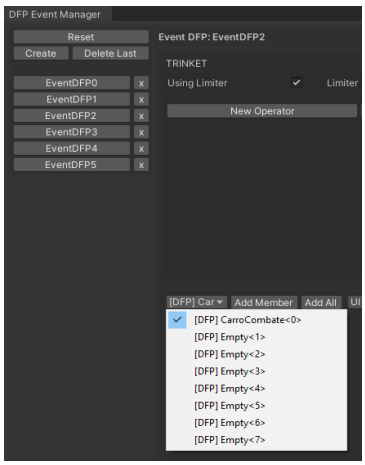


Ilustración 57 Listado de Miembros DFP actualizados

A su vez, los cambios producidos en cualquiera de los editores también provocan reacciones en los otros editores.

Si añadimos un Miembro DFP a un evento, automáticamente se actualiza su sección de Developer Mode, si se cambia el nombre de un Grupo DFP en Group Maker, se actualiza la sección de Sound de Miembro DFP, si se crea un Grupo DFP nuevo se actualiza el Manager DFP, si se borra un objeto que sea Miembro DFP, se actualizan todos los listados de los editores, etc (Ilustración 58).

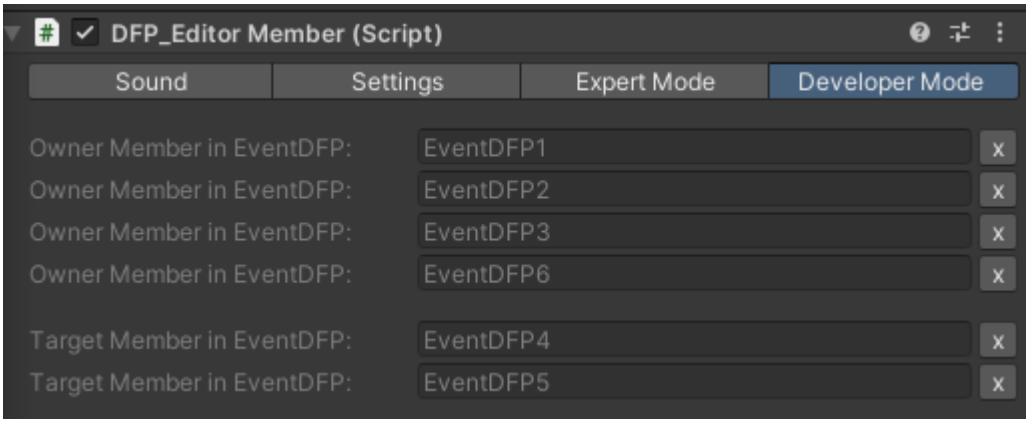


Ilustración 58 Developer Mode Miembros DFP

En definitiva, existe una sincronización total entre todos los datos implicados en los editores DFP.

Como ejemplo, aquí se puede observar, como los Editores y las Ventanas de Edición Extendidas se comunican para sincronizar la información en tiempo real de la información relativa a Eventos DFP (Ilustración 59).

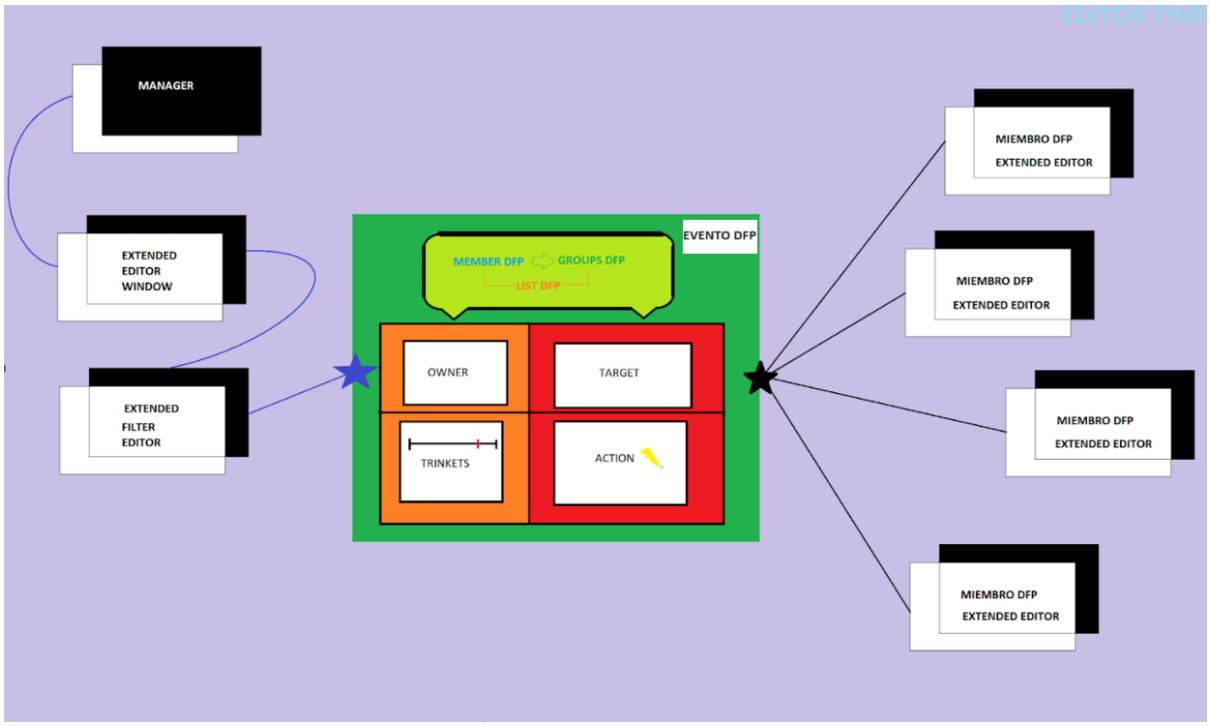


Ilustración 59 Comunicación entre ventanas y editores

6.3.5 Editores y ventanas DFP. Protección activa.

La Caché DFP es una herramienta del plugin que realiza un constante control sobre los módulos esenciales y los datos sensibles que gestiona el plugin.

El usuario en su desconocimiento podría corromper o borrar la caché y el proyecto quedaría inservible o al menos todo el contenido relacionado con el Plugin DFP.

Para evitar esto se crea un sistema de protección para los sistemas de protección activo que proteja a la Caché. Gracias a la red de herencia que se diseña para el sistema de Editores y Ventanas Extendidas DFP, se puede implementar una especie de vigilante "del vigilante" siguiendo este razonamiento lógico:

El uso del plugin DFP en Editor Time implica utilizar cualquiera de los Editores o Ventanas Extendidas DFP.

Cuando el plugin DFP interactúa en Run Time y gracias al sistema de Unity, se instancian copias de los datos en edición, por lo que podemos garantizar que el usuario no pueda corromperlos.

El total de las herramientas están construidas sobre Editores y Ventanas DFP.

Estas herramientas se actualizan de manera directa con los ciclos de actualización del editor de Unity.

Por lo tanto:

Si se protege la Caché desde los ciclos de actualización de la red herramientas DFP, y su vez, la Caché protege en todos los frames de actualización las herramientas del plugin, se garantiza que el mero uso del plugin haga que se proteja así mismo: Sistema de protección para los sistemas de protección.

De esta forma, si la Caché se ve afectada, el primero de los editores o ventanas DFP que se actualice detectará la incidencia y procederá a solventar el problema. Primero se avisará al usuario y le dará un tiempo prudencial para intentar arreglarlo, pasado este tiempo la red DFP intentará restaurar la Caché.

6.4 Actores DFP

El plugin, de forma general, se soporta en figuras o "actores" que interactúan en todas las herramientas (Ilustración 60), cooperando para sonorizar la escena. Se conoce como Actor DFP a cualquier "figura" que puede actuar de forma independiente y puede interactuar con otros actores DFP.

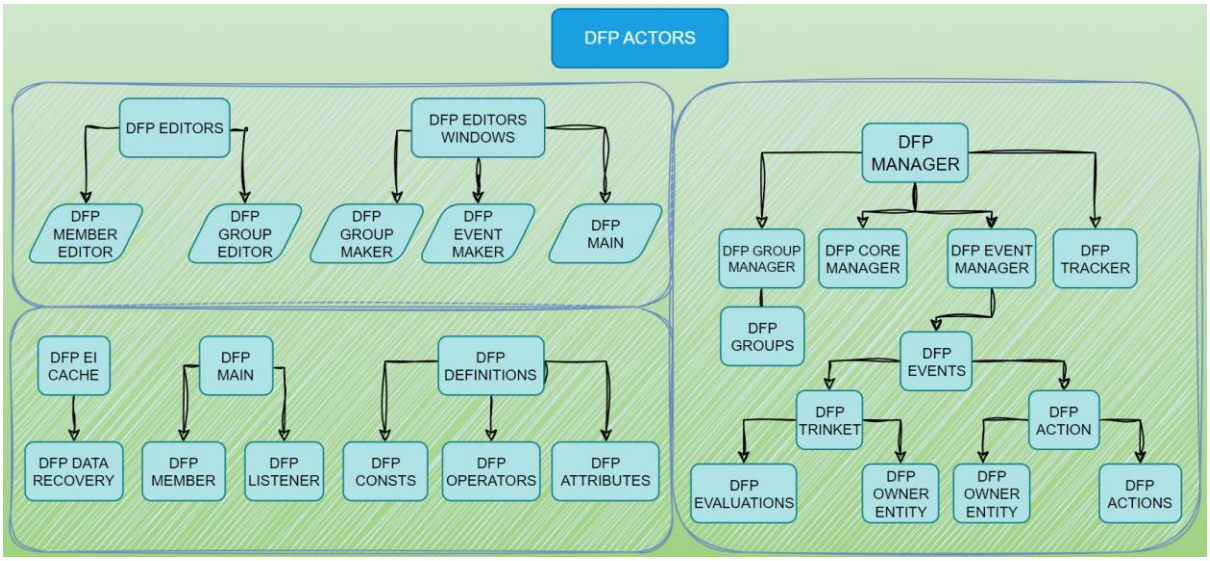


Ilustración 60 Actores DFP

Muchos de estos Actores DFP cuentan con editores propios y personalizados o ventanas de edición personalizadas. Algunos de ellos adquieren funcionalidades muy específicas por lo que se acaban convirtiendo en Herramientas DFP.

A continuación, hablaremos de estos Actores DFP, de cómo interactúan, cómo se comunican y de cómo se comportan en los distintos ámbitos de trabajo.

Para facilitar el proceso de comprensión al lector, se facilita en la siguiente imágen (Ilustración 61) un esquema global sobre cómo se comunican los distintos Actores DFP en los diferentes ámbitos de trabajo, Editor Time y Run Time.

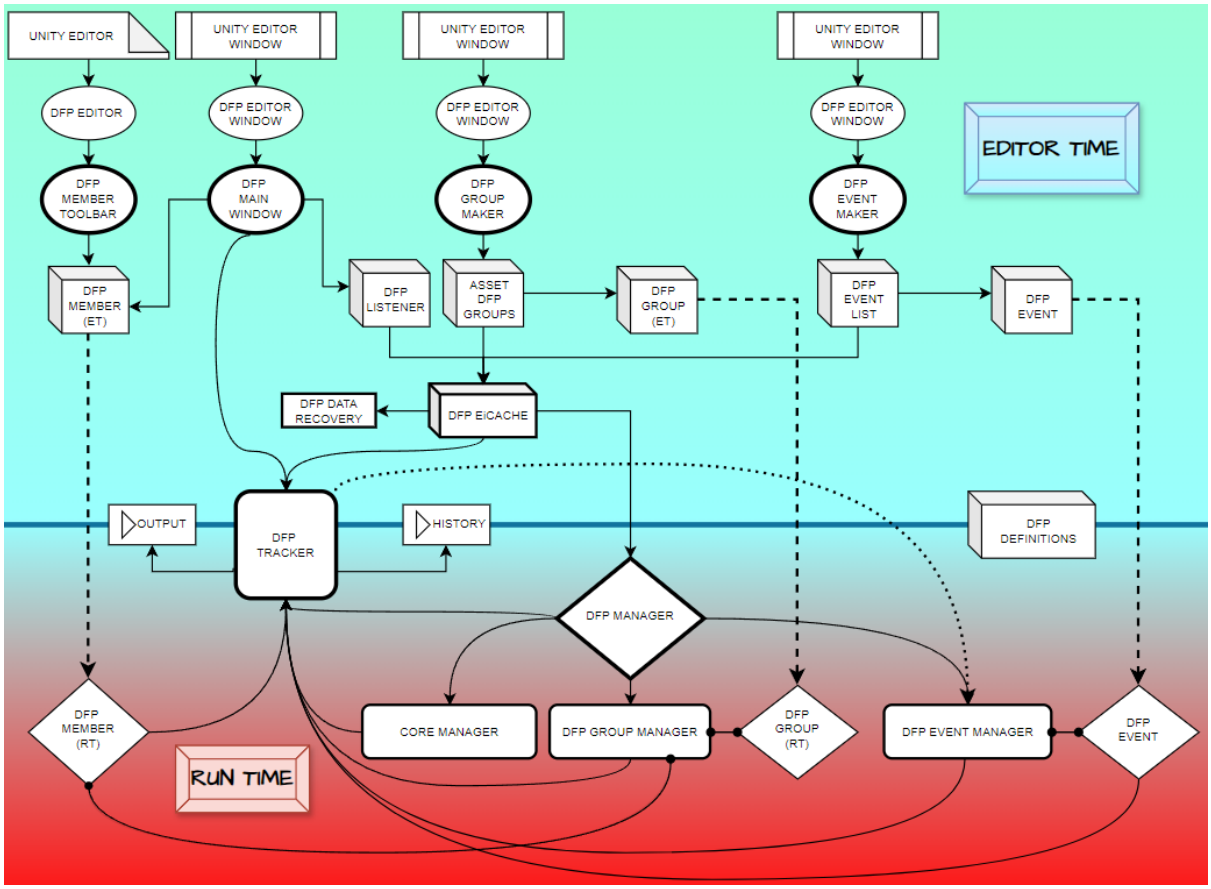


Ilustración 61 Comunicación Actores DFP

En este esquema también se puede apreciar cuáles de las herramientas atienden a un ámbito doble de trabajo, convirtiéndose en híbridas.

6.4.1 Miembro DFP

En términos generales, un Miembro DFP es un objeto que está sonorizado con el plugin y que puede interactuar con el total de las herramientas de este. En ejecución representan los diferentes puntos de emisión de audio(similar Fmod Emitter (24)) que se utilizan para calcular el conjunto global de la sonorización 3D orientada al Oyente DFP.

Los Miembros DFP cuentan con una versión diferente para cada ámbito de trabajo.

En Editor Time actúa el Miembro de Editor Time, se trata de un objeto serializable que se usa para alimentar el editor de Miembros DFP. Se comporta como una plantilla que se rellena y se encarga de inicializar los datos del Miembro de Run Time en el cambio de ámbito de trabajo, tras esto, se liberará del objeto sonorizado

para agilizar la carga y el tratamiento del objeto en tiempo de ejecución, participando en el proceso de agilizar el ciclo de actualización del plugin y reduciendo así el impacto del uso del plugin en el sistema.

En Run Time entra en acción el Miembro de Run Time el cual cambia completamente su rol. Pasa de ser un ente pasivo, a tener autonomía, ciclos de actualizaciones propios y cuenta con API's de tratamiento de Atributos de Fmod. Además, el Miembro DFP de Run Time tiene la obligación de informar de cada proceso asociado a un evento que haga, como por ejemplo, cambiar su volúmen o su pitch.

6.4.2 Grupo DFP

Un Grupo DFP es un objeto que se define para agrupar Miembros DFP y poder controlar de forma grupal las acciones que ofrece el plugin sobre los distintos Miembros DFP.

Inicialmente los Grupos DFP no existían. La API de bajo nivel de Fmod y la de Fmod Studio incorporan las clases Fmod Channel, Fmod Channel Group [20] y Fmod Bus [28] como estructuras para dar este soporte.

Tras encontrarme con infinidad de problemas en la integración de estos componentes y bugs en ejecución me puse en contacto con la misma empresa Fmod para ver si me podían ayudar con la solución a mi diseño de arquitectura ya que no entendía lo que me estaba ocurriendo. Las API's de estas clases cuentan con funciones que invitan a pensar que se pueden utilizar en cualquier ámbito de trabajo, ya que te permiten programar fragmentos de código como el que se muestra a continuación(Ilustración 62):

```
if (_fmodSys.createChannelGroup("SFX", out _chgMUSIC) == FMOD.RESULT.OK)
{
    _musicInstance = FMODUnity.RuntimeManager.CreateInstance(musicEventRef.Path);
    FMODUnity.RuntimeManager.AttachInstanceToGameObject(_musicInstance, GetComponent<Transform>(), GetComponent<Rigidbody>());

    if (_musicInstance.isValid())
    {
        _musicInstance.start();
        //-> how to add instance to channel group ????
    }
}
```

Ilustración 62 Fragmento experimental FMOD

Cuyas variables son(Ilustración 63):

```
private FMOD.System _fmodSys;
private FMOD.ChannelGroup _chgMUSIC;
private FMOD.Studio.EventInstance _musicInstance;
private bool _mute;
```

Ilustración 63 Variables fragmento experimental FMOD

Como se puede observar, la variable "_chgMUSIC" es de tipo ChannelGroup, el cual se puede inicializar con la llamada al sistema de Fmod "createChannelGroup" mientras que "_musicInstance" es una Instancia de Evento de Fmod, la cual se supone que está preparado para trabajar con los canales.

La pega técnica es que ChannelGroup pertenece a Core API mientras que Event Fmod es de la Studip API que a su vez se alimenta de la primera. Los buses por otro lado ofrecían el agrupamiento de Instancias, pero sin embargo no contaban con la API de gestión de Channel Group, básicamente se presentan como contenedores con funcionalidades muy limitadas.

Entonces, ¿cómo agrupar y manipular dichas instancias en ejecución?

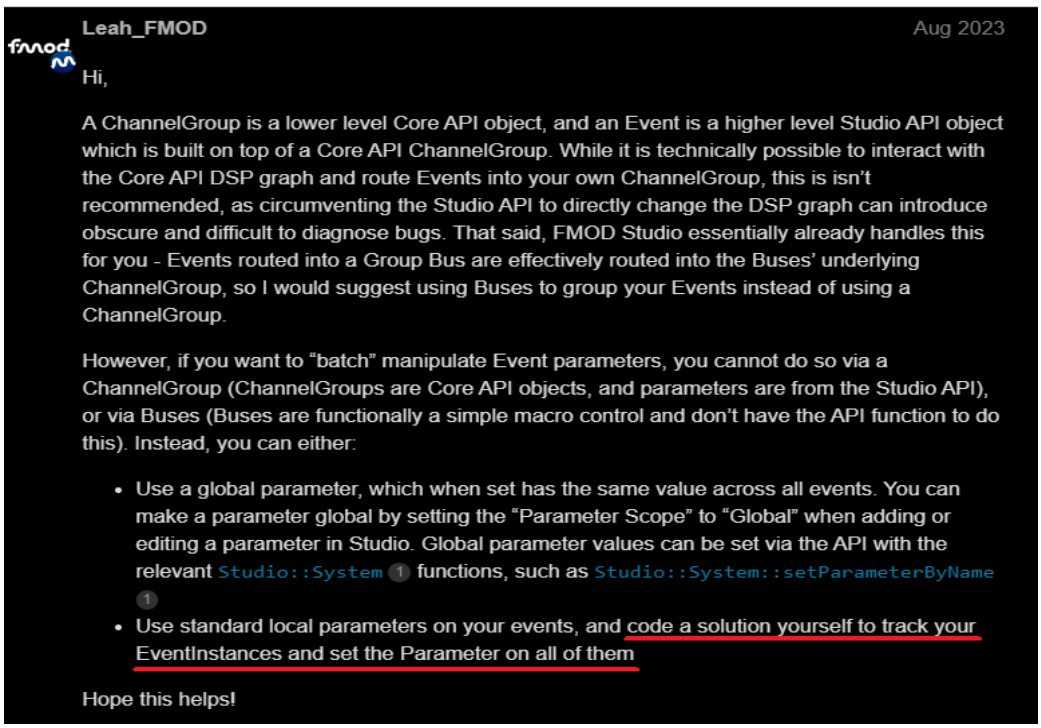


Ilustración 64 Respuesta administrador FMOD

"Code a solution yourself to track your EventInstances and set the Parameter on all of them" - Leah_Fmod Administrator – (27)

Si los Eventos Fmod (25) son estructuras que se definen en Tiempo de Edición o Editor Time, se cierra la puerta al tratamiento en vivo en tiempo de ejecución ya que acciones tan simples como la de añadir un Evento FMod a un Canal (FmodChannel) o moverlo de Grupo (Channel Group / Bus) desde script (26) son imposibles de hacer.

Ante las evidencias, se crea la figura de Grupo DFP como herramienta para la agrupación y control de Miembros DFP.

Con los Grupos DFP además se incorpora la funcionalidad de mover Miembros DFP de un grupo a otro, personalmente esto es algo que buscaba incorporar porque como desarrollador me he encontrado varias veces con la necesidad de poder hacer esto. Por ejemplo, juegos 2D que cuentan con objetos que contienen varios sprites o cargadores de sprites para poder renderizar diferentes modelos del objeto sin tener que reinstanciar el objeto para ahorrar tiempo de procesamiento. Esta misma acción se debería poder aplicar a los sonidos. Si un objeto puede cambiar su composición visual sin tener que reinstanciar el objeto, ¿por qué tener que hacerlo para cambiar su composición sonora? Según la política de Eventos Fmod debemos tener instancias en objetos auxiliares para poder cargarlas en otros objetos o tener objetos asociados a instancias determinadas e instanciar los objetos cuando queramos cambiar de instancia.

Aprovechando que había que añadir esta figura como nuevo Actor DFP, se analizó la posibilidad de aislar los datos de los Grupos DFP de la escena para posibilitar la reutilización de estos en otras escenas o proyectos. Este concepto trae asociada la creación de Group Maker, una especie de "Grupo de Grupos DFP" con funcionalidades para la gestión de "Grupos DFP". Cuenta con menú de acceso rápido y sus datos se serializan en un fichero .asset que se puede copiar, editar y utilizar en otras escenas.

Los Grupos DFP cuentan con varias herramientas a su disposición:

- Editor de Grupos nos permite modificar los parámetros de los grupos DFP.
- El Creador de grupos (Group maker) es una herramienta que permite la creación y administración de grupos (29).

- El Administrador de grupos se encarga de definir los grupos en tiempo de edición y del control de los grupos en ejecución.

6.4.3 Atributo DFP

Los Atributos DFP son una parte esencial que define los "atributos" con los que se trabaja en el plugin.

Un Atributo DFP no representa un atributo usual al que se está acostumbrado en la programación orientada a objetos. Es una especie de contenedor de identificadores o una especie de símbolo no resuelto que en algún momento de la ejecución se espera que se definan, como sería un atributo en lenguajes interpretados como JavaScript, pero adaptado para C#. Se pretende dar la posibilidad de crear un representante en forma de enumerado que no tiene ninguna diferencia con otro más allá del identificador pero que son definidos normalmente para comportamientos totalmente diferentes.

Es una forma de poder dar soporte a la creación de cualquier tipo de elementos que se puedan imaginar como atributos, por ejemplo, un atributo puede ser el Volúmen o la Prioridad, pero también lo puede ser "Play, uno como representante de una acción y otro como representante de un tipo de atributos propio de otra API.

Se definen en el fichero DFP_Definitions.cs (Ilustración 65) y cuenta con diccionarios para trabajar con los distintos operadores y funciones que dan soporte al plugin para todo tipo de traducciones y consultas.

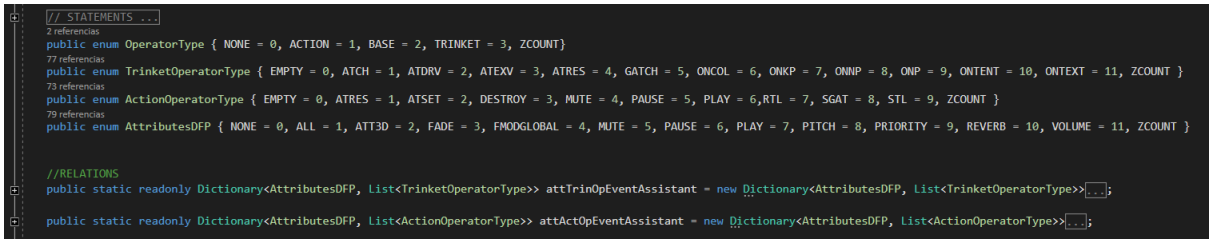


Ilustración 65 Declaraciones constantes DFP Definitions

Estos Atributos DFP se utilizan, entre otras cosas, para definir Operadores de Evaluación y Operadores de Acción en Eventos DFP, concretamente definen el objetivo de la evaluación o de la acción dentro de los eventos.

Para poder hacer una evaluación, se necesita algo qué evaluar y para poder ejercer una acción sobre algo, se necesita un objetivo: Atributos DFP.

6.4.4 Entidades DFP

Las Entidades DFP representan un conjunto de Miembros DFP y Grupos DFP que se relacionan para recibir un tratamiento especial ya que son utilizadas en la creación de Eventos DFP. Existen dos tipos de Entidades:

- Entidad Propietaria
- Entidad Objetivo.

La Entidad Propietaria agrupa a los sujetos de evaluación. Se evalúan unas condiciones, que, en caso de que se cumplan, provocará una serie de sucesos o acciones definidas para la Entidad Objetivo.

La Entidad Objetivo agrupa a los sujetos receptores de las acciones definidas en el Evento DFP, en caso de que la evaluación a la Entidad Propietaria se cumpla.

Es necesario entender el concepto de aplicación de evaluaciones o acciones en Run Time a las Entidades, en función de los actores que forman parte de ellas. Se trata de responder a preguntas cómo: ¿Cómo se evalúan las condiciones? ¿Cómo se aplican las acciones? ¿Se hace igual a Miembros que a Grupos? ¿Todos los grupos son iguales? ¿Y los miembros de unos grupos u otros qué diferencias tienen entre ellos? … Para responder a estas preguntas se diseña un sistema de configuración de evaluaciones y aplicaciones.

6.4.4.1 Aplicación a Miembro DFP como Entidad DFP

La evaluación o aplicación de un Evento DFP a un Miembro DFP se aplicará de forma instantánea o retardada por un temporizador. El comportamiento que adoptan varía en función de la Entidad a la que forman parte.

Si el Miembro DFP es parte de la Entidad Propietaria, se evaluará el TRINKET en dicho miembro como una parte más de la Entidad. Cada uno de los Miembros DFP que formen parte de la Entidad Propietaria será evaluado de forma equitativa y cada uno de ellos podrá ser responsable de que se cumplan las evaluaciones.

Si el Miembro DFP es parte de la Entidad Objetivo, se le aplicarán el total de acciones que formen parte de ACTION cuando el disparador del evento salte. A cada uno de los Miembros DFP que formen parte de una Entidad Objetivo se le aplicarán las acciones de forma equitativa.

6.4.4.2 Aplicación a Grupo DFP como Entidad DFP

La evaluación o aplicación de un Evento DFP a un Grupo DFP se aplicará de forma instantánea o retardada por un temporizador. El comportamiento que adoptan varía en función de la Entidad a la que forman parte.

Si el Grupo DFP es parte de la Entidad Propietaria, se evaluará el TRINKET en cada uno de los Miembros DFP que forman parte del Grupo DFP. Cada uno de los Miembros DFP que formen parte del Grupo DFP, que a su vez forme parte de una Entidad Propietaria, será evaluado de forma equitativa y cada uno de ellos podrá ser responsable de que se cumplan las evaluaciones.

Si el Grupo DFP es parte de la Entidad Objetivo, se le aplicarán el total de acciones que formen parte de ACTION cuando el disparador del evento salte. A cada uno de los Miembros DFP que formen parte del Grupo DFP que pertenezca a una Entidad Objetivo, se le aplicarán las acciones de forma equitativa.

6.4.5 Operador DFP

Los Operadores DFP surgen como necesidad del desarrollo. Ofrecen un abanico de posibilidades para la creación de eventos en forma de Operadores de Evaluación y Operadores de acción.

Los Operadores de Evaluación definen situaciones que se deben evaluar a los Miembros DFP y Grupos DFP. Estos operadores se utilizan en el actor Trinket DFP.

Los Operadores de Acción definen acciones que se pueden invocar sobre los Miembros DFP y Grupos DFP. Estos operadores se utilizan en el actor Action DFP.

Un Operador DFP se define con el formato:

- [ATRIBUTO DFP] [CÓDIGO OPERACIÓN]

El código de operación puede ser:

- Código de evaluación
- Código de acción

En función del tipo de código que se use para definir el operador lo convierte en:

- Operador de Evaluación (Trinket Operator)
- Operador de Acción (Action Operator)

6.4.6 Evento DFP

Como se comenta en los conceptos básicos, un Evento DFP representa la asociación de una serie de evaluaciones lógicas con una serie de operaciones de configuración sonora que se aplican a Miembros DFP y Grupos DFP.

En el ámbito de los Eventos DFP se pueden distinguir cuatro grandis figuras: Las evaluaciones, los receptores de esas evaluaciones, las acciones y los receptores de esas acciones. Los receptores de las evaluaciones o acciones son representados como Entidades mientras que los procesos son definidos en forma de operadores de evaluación o de acción.

Como ya se vió en el apartado de Entidades DFP, existen dos tipos de estas, Entidad Propietaria y Entidad Objetivo.

A la Entidad Propietaria se le evalúan unas condiciones, que, en caso de que se cumplan, provocarán una serie de sucesos o acciones definidas para la Entidad Objetivo.

La Entidad Objetivo será a quién se apliquen las acciones definidas en el Evento DFP en caso de que la evaluación a la Entidad Propietaria se cumpla.

El Evento DFP está compuesto por:

- Definición de las evaluaciones
- Definición de las acciones
- Definición de las distintas entidades que formarán parte del Evento DFP

Cada Evento DFP contiene dos gestores independientes además del controlador principal:

Trinket: encargado de la gestión de las evaluaciones y de la Entidad Propietaria a la que se evalúa

Action: encargado de la gestión de las acciones y de la Entidad Objetivo a la que se le aplican dichas acciones

Además, es necesario definir la forma en la que los Eventos DFP interactúan con los Miembros y Grupos DFP en Run Time. Un Evento DFP contiene entre otras cosas, evalúa una serie de condiciones u operadores y aplica acciones o configuraciones.

De alguna forma se debe relacionar el uso de los atributos de las clases con los operadores. Esta parte se cede al Event Manager, encargado de la gestión de Eventos DFP en ejecución.

6.4.7 Trinket DFP: operadores de evaluación y entidad propietaria

Trinket DFP forma parte de los Eventos DFP. Es el encargado de la gestión de las evaluaciones y de la Entidad Propietaria a la que se evalúa, es decir, definir qué evaluaciones se deben hacer y a quién dentro de un Evento DFP.

Las condiciones a evaluar se crean y configuran en forma de operadores, en la parte superior de la zona del Trinket.

La entidad propietaria se define con los Miembros DFP y Grupos DFP que se añaden en la parte inferior de la sección.

Por lo tanto, Trinket lo conforman:

- Operadores de evaluación para definir las evaluaciones.
- Miembros DFP para definir la Entidad Propietaria.
- Grupos DFP para definir la Eentidad Propietaria.

Un Operador de Evaluación se compone de:

- [Atributo DFP][Operación Evaluación]

Un TRINKET define un listado de operaciones que deben ser evaluadas en la entidad propietaria hasta que de forma parcial o todos se cumplan (en función del limitador). Al tratarse de instrucciones codificadas para permitir que el usuario "programe" a través de la interfaz, cada instrucción genérica debe tener un identificador. El identificador de operador se denomina Operador de Evaluación y define una secuencia de evaluación.

Cada Operador de Evaluación debe ser ÚNICO, es decir, no puede existir dos que ejecuten la misma secuencia de evaluación y deberá contener un identificador que podrá ser consultado - también será único -. Dicho operador tendrá formato de entero largo sin signo empezando desde el elemento 1 - reservando el 0 para el operador "VACÍO" - y serializando los operadores por orden alfabético de etiqueta de operador - salvo el 0 -.

En este apartado se podrían añadir infinitas posibilidades, para la versión inicial, se dispondrán algunos operadores generales y además, los atributos de

acceso público de los Miembros DFP - por propia naturaleza del atributo o por acceso delegado en funciones -, dispondrán de Operadores de Evaluación, siendo el total de estos:

- EMPTY
- ATCH = ATTRIBUTE CHANGE
- ATDRV = ATTRIBUTE DROPS VALUE
- ATEXV = ATTRIBUTE EXCEEDS VALUE
- ATRES = ATTRIBUTE RESTORE
- ONCOL = ON COLISION (NOT IMPLEMENTED)
- ONNP = ON NEW PLAY
- ONKP = ON KEY PRESS
- ONP = ON PLAY
- ONTENT = ON TRIGGER ENTER (NOT IMPLEMENTED)
- ONTEXT = ON TRIGGER EXIT (NOT IMPLEMENTED)
- GATCH = FMOD GLOBAL ATTRIBUTE CHANGE (NOT IMPLEMENTED)

Algunos de ellos no están implementados, pero se dejan como muestrario simple de las diferentes posibilidades que se podrían añadir como evaluaciones. Además el usuario podrá definir a través del Event Maker el porcentaje de Operadores de Evaluación que deben cumplirse para que el disparador se active, este valor se define como TRINKET_LIMITER. De esta forma, si el Trinket cuenta con 10 secuencias de evaluación y el usuario define TRINKET_LIMITER A 20%, al cumplirse 2 Operadores de Evaluación cualquiera, el disparador se activa.

Como un Trinket puede estar compuesto por secuencias de evaluación largas (múltiples Trinket Operators), se debe evitar las consultas abusivas, por esto se implementa el siguiente sistema de control y actualización de Trinkets:

- Vector de Operadores: Es de tipo "Trinket Operator" y contiene una posición por cada Trinket Operator que contenga el TRINKET.

- Vector de estado de operadores: contiene una posición por cada Trinket Operator y almacenará el valor del estado del operador, pudiendo contener valores de estado como STARTED, EVALUATION_TRUE, EVALUATION_FALSE, PENDING, etc.

Tras finalizar la evaluación de un operador se reajusta si fuera necesario el valor de estado del operador en el "Vector de estado de operadores", esto dependerá

del gestor de Eventos. El resultado global se debe tener en cuenta por el uso del Limitador.

Tras la evaluación de todos los operadores se comprueba a través de un algoritmo el estado global del Trinket. Si se llega al "goal" o mínimo de Operadores de Evaluación que deben completarse en función del limitador, se inicia el temporizador si fuera necesario y se lanza el Action asociado al Evento DFP.

Estas acciones están coordinadas con el tratamiento global de los Eventos DFP que ofrece la herramienta Event Maker.

6.4.8 Action DFP: operadores de acción y entidad objetivo

Action DFP forma parte de los Eventos DFP. Es el encargado de la gestión de las acciones y de la Entidad Objetivo a la que se le aplican dichas acciones, es decir, definir qué acciones se deben hacer y a quién dentro de un evento DFP.

Action DFP se encarga de aplicar distintas acciones que se ejercen sobre la entidad objetivo cuando las condiciones de evaluación se han cumplido sobre la entidad propietaria y cuentan con su propio gestor para que se puedan construir de forma independiente.

La entidad objetivo se define con los Miembros DFP y Grupos DFP que se añaden en la parte inferior de la sección.

Las acciones se crean y configuran en forma de operadores, en la parte superior de la zona del Action.

Por lo tanto, Action lo conforman.

- Operadores de acción para definir las acciones.
- Miembros DFP para definir la Entidad Objetivo.
- Grupos DFP para definir la Entidad Objetivo.

Un Operador de Acción se compone de:

- [Atributo DFP][Operación de Acción]

En el entorno de ejecución, estos Operadores de Acción son como contenedores llenos de información con instrucciones sobre cómo proceder para aplicar las distintas acciones a la Entidad Objetivo.

La configuración sonora que se aplica en la entidad propietaria se denomina "ACTION" y está constituida por una o varias instrucciones de configuración sonora que se aplican cuando el TRINKET asociado al evento DFP se dispara.

Al tratarse de instrucciones codificadas para permitir que el usuario "programe" a través de la interfaz, cada configuración genérica debe tener un identificador.

Cada Operador de Acción debe ser ÚNICO, es decir, no puede existir dos Operadores de Acción que ejecuten la misma secuencia de configuración.

En este apartado se podrían añadir infinitas posibilidades y no es el objetivo final de este proyecto. Para la versión inicial, se dispondrán algunos operadores generales y además, los atributos de acceso público de los Miembros DFP - por propia naturaleza del atributo o por acceso delegado en funciones -, dispondrán de Action Operators, siendo el total de estas:

- EMPTY
- ATRES = ATTRIBUTE RESTORE
- ATSET= ATTRIBUTE SET
- DES = DESTROY
- M = MUTE
- P = PAUSE
- STL = SAVE TIME LINE POSITION
- RTL = RESTORE TIME LINE POSITION
- SGAT = SET FMOD GLOBAL ATTRIBUTE (NOT IMPLEMENTED)

Action contará con temporizadores para gestionar el orden y los tiempos que tarda cada Operador de Acción en ejecutarse:

- Starter Timer: Temporizador global que configura el tiempo global que tarda en iniciarse las acciones

- Action Operator Timer: Temporizador individual que indica el tiempo que tarda cada acción en ejecutarse

La conjunción de los distintos elementos de los Eventos DFP clasifican los tipos de Eventos DFP que se pueden crear en:

- Eventos Simples: Eventos que cuentan con al menos 1 evaluación, 1 objeto de evaluación, 1 acción y 1 objeto de acción.

- Eventos Multi: Eventos que escalan las versiones de Eventos Simples añadiendo múltiples evaluaciones, acciones o Miembros/Grupos DFP en las entidades.
- Eventos Limitados: Eventos que se limitan porcentualmente en evaluación permitiendo el acceso a un "OR" lógico un tanto especial.
- Eventos Temporizados: Eventos que se temporizan a través de sus acciones para coordinar las distintas configuraciones de atributos que se ejercen sobre la Entidad Objetivo.
- Eventos Encadenados: Eventos que sirven de lanzamiento para otros eventos. Esto es posible porque la salida (output) de un Evento DFP en forma de acciones lógicas se pueden utilizar como entrada para evaluaciones en los Eventos DFP, pudiendo encadenar hasta N-1 niveles, siendo N el número total de Eventos distintos que contenga el gestor de eventos.

6.5 Herramientas DFP

6.5.1 Menú principal

El menú principal es la herramienta más humilde. A pesar de su apariencia es una de las herramientas más versátiles y poderosas del plugin.

Se encarga de auto instalar y configurar el plugin, de levantar inicialmente los pilares. También cuenta con sus propias funcionalidades que podemos ver como la gestión de la conversión a Miembros DFP de los objetos u otras como limpiar la escena de todos los datos relacionados con el plugin.

Cuando el usuario abra la ventana principal, gracias a la implementación de editores extendidos (ver sección: Editores y Ventanas de Edición), el plugin hará un primer escaneo en busca de la Caché (Ilustración 66).

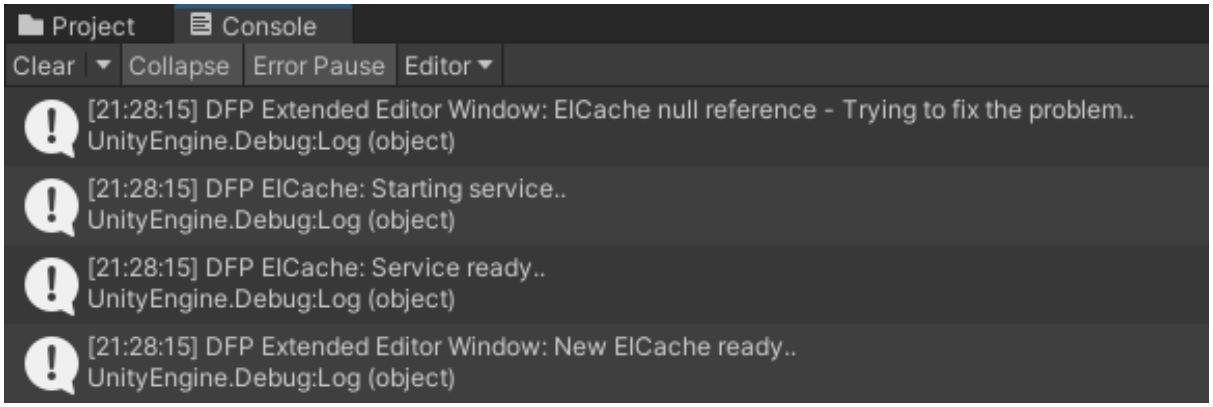


Ilustración 66 Inicialización de módulos

Tras levantar la Caché, ésta hará un escaneo de toda la escena en busca de los componentes que necesita el plugin para trabajar.

Cuando esto ocurre y de forma temporal el plugin da instrucciones al usuario que puede ejecutar para acelerar el proceso.

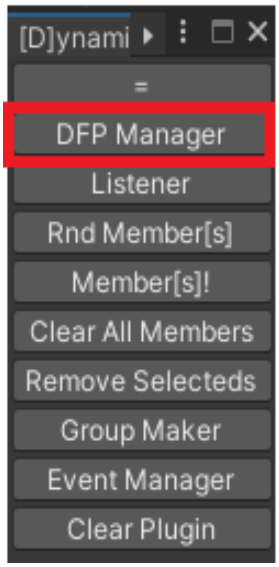


Ilustración 68 Ventana principal incidencia

Esto es posible gracias a que temporalmente y hasta que se resuelva la incidencia, el plugin nos habilita en la ventana principal de trabajo DFP un nuevo botón.

Este botón obliga al plugin a instanciar todos los managers necesarios para que reinicie la instancia, pero pierde la oportunidad de que la caché intente recuperar los posibles datos perdidos.

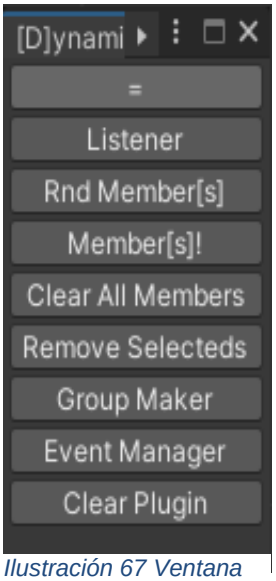


Ilustración 67 Ventana principal

Si no se ejecuta ninguna acción, la caché levantará todos los módulos necesarios que necesite para hacer el plugin funcional (Ilustración 69).

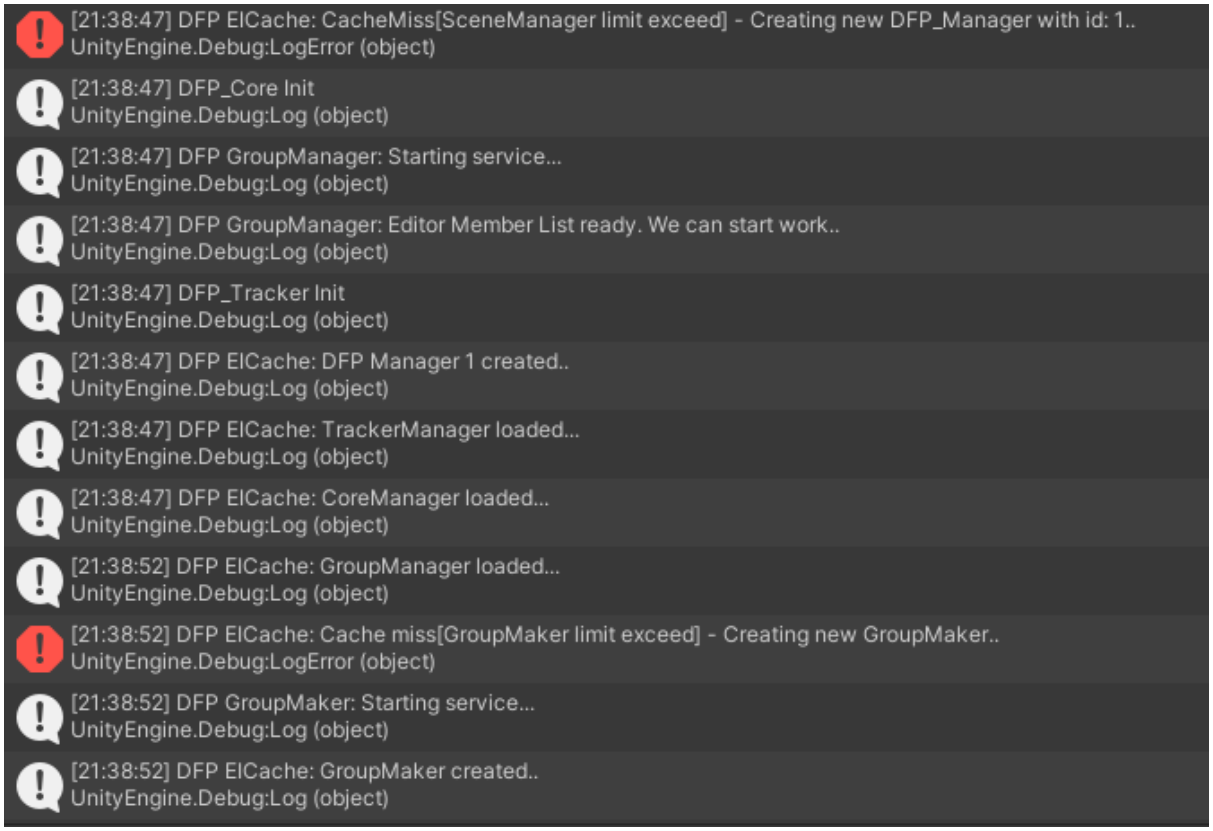


Ilustración 69 Restauración de módulos

Algunos de estos módulos son más importantes que otros debido a que están preparados para trabajar en Editor Time.

6.5.2 Member DFP Editor

El Editor de Miembros ofrece soporte para la configuración sonora y de datos de sonorización de los Miembros DFP. Cuenta con cuatro secciones de las cuales solo dos están inicialmente visibles y además de ofrecer otras funcionalidades como las del control de eventos en los que actúa el objeto sonorizado.

Durante la edición en Editor Time de los Miembros DFP entra en acción dos clases que dan soporte a la herramienta de Miembros DFP. Una actúa como interfaz de datos serializables y otra como editor y gestor de datos.

En términos de asistencia al usuario, esta herramienta ofrece facilidades para la gestión de los datos y la configuración relacionada con cada objeto que se sonoriza en la escena.

En términos de rendimiento, esta herramienta ayuda a reducir la carga de datos en Run Time aislando todo lo que está relacionado con la parte de edición y composición de la escena.

Cuando la escena se lanza, el plugin se encarga de borrar todo este tipo de componentes que son innecesarios en la parte de Run Time para aliviar la carga de procesamiento y memoria, aislar la información necesaria y alimentar todos los módulos necesarios.

En la sección "Manual de usuario" se muestra cómo utilizar la interfaz de esta herramienta.

6.5.3 Group Maker

Grupos DFP es un tanto peculiar ya que representa una herramienta reutilizable entre escenas. Esto requiere un soporte de datos en ficheros externos, que, en DFP se realiza en forma de ficheros .asset.

En un mismo proyecto sólo se puede utilizar un asset de Grupo DFP como gestor de grupos, pero se ofrece la posibilidad de crear múltiples assets de este tipo con la ventana de creación de acceso rápido.

Al seleccionar cualquiera de los asset de Grupo DFP se puede ver en el Inspector que aparece un único botón para editar el asset (Ilustración 70).

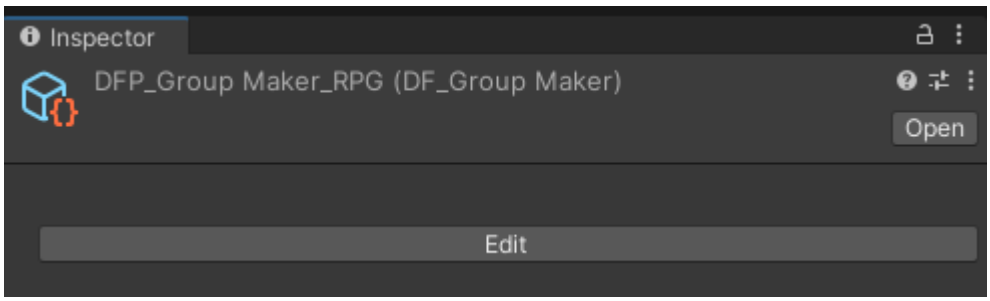


Ilustración 70 Inspector

Este botón nos cargará el asset en el Group Maker, herramienta que da soporte a la creación, modificación y borrado de datos relacionados con Grupos DFP (Ilustración 71).

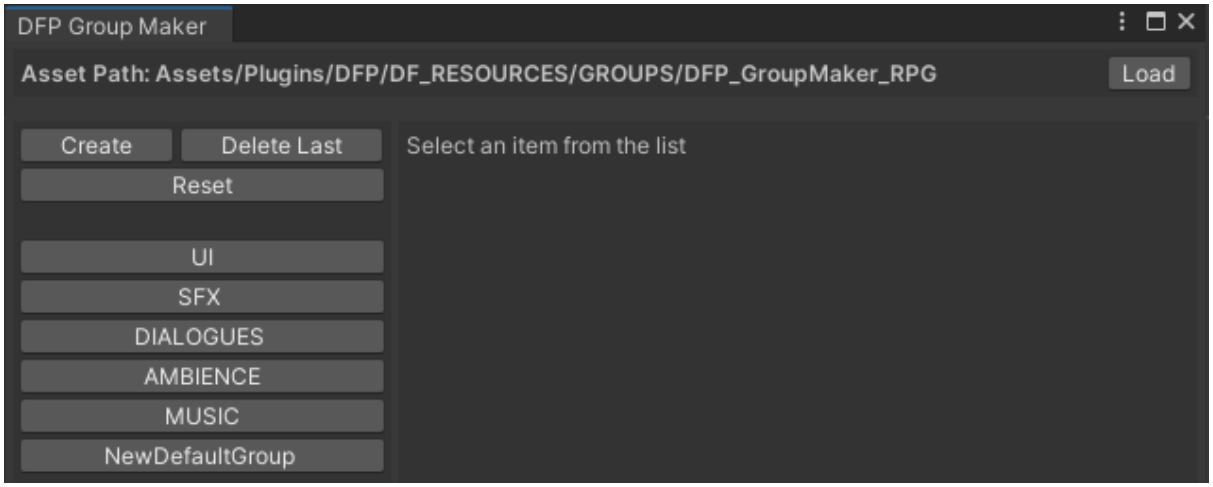


Ilustración 71 Group Maker

Se puede utilizar el botón Load para cargar el Asset de Grupos DFP que se está editando como asset de Grupo a utilizar en la escena.

La herramienta de grupos que se ofrece en el plugin DFP es un tanto peculiar. Mientras que los miembros son objetos de escena, los grupos son un conjunto de serialización de datos que se almacenan en un objeto de tipo Asset. Para evitar una dependencia directa desde los grupos a los miembros, son los miembros los que configuran a qué grupo pertenecen mientras que los grupos únicamente se configuran a sí mismos, es decir, todos los datos configurados en el editor de grupos pertenecen única y exclusivamente a dichos grupos. *Puedes aprender a usar esta herramienta en el* Tutorial - Herramienta: Group Maker

6.5.4 DFP Manager

DFP es una herramienta que se construye para el control y la gestión de funcionalidades determinadas del plugin en ambos ámbitos de trabajo. Dicha gestión se deja en mano de los Managers, gestores que se especializan en dar soporte a funciones específicas.

Core Manager gestiona la interacción principal con el Sistema de Fmod.

Group manager ofrece soporte para la edición y desarrollo de grupos y para su control en ejecución.

Event Manager se presenta como gestor para el control y la administración de Eventos DFP en ejecución.

Tracker es el encargado de la gestión de la comunicación y el sistema de depuración.

6.5.5 Core Manager

Core Manager es el gestor del núcleo del plugin en Run Time. Su funcionalidad es la de hacer de intermediario entre las API's de DFP y Fmod y la de dar soporte en tiempo de ejecución o Run Time.

Se encarga de recibir peticiones de las herramientas DFP y lanzarlas contra el sistema de Fmod, gestiona la creación de componentes y de los Miembros DFP en Run Time y el realiza otras funciones de control.

6.5.6 Group Manager

La figura del Group Manager es una herramienta híbrida un tanto peculiar, ya que es partícipe tanto en tiempo de edición como en ejecución, pero toma un rol mucho más activo e importante en el tiempo de ejecución. Mientras que la mayoría del tiempo de edición y desarrollo gana fuerza la herramienta de Group Maker para la gestión de Grupos DFP, Group Manager pasa a un segundo plano estando como observador. A fin de cuentas, un Grupo DFP en tiempo de edición es un actor pasivo.

Sin embargo, cuando la escena se lanza la figura del Gestor de Grupos cambia por completo.:

- Cada Grupo DFP tiene su propio objeto en ejecución que controla a sus diferentes Miembros DFP
- Puede tomar acciones a nivel individual o global los Miembros DFP
- Sirve como representante en la comunicación con el Tracker Manager para el control y ejecución de los Eventos DFP.

6.5.7 Tracker Manager

Una de las herramientas fundamentales para el desarrollo del plugin DFP es el Tracker Manager. En términos de implementación está diseñada para facilitar la comunicación entre las distintas estructuras y herramientas mientras que en términos de funcionalidad, se encarga de recoger y filtrar información de los distintos sistemas y actores que componen el plugin: mensajes de información del sistema, de errores, de ejecución de eventos importantes, de comunicación de Miembros DFP, etc.

El Editor del Tracker Manager nos permite depurar y filtrar información detallada a través de mensajes que se registran en la consola de Unity. Cuenta con estructuras para el tracking de mensajes de tipo DFP_GlobalMsg y con estándares de información para unificar los mensajes que se muestran al usuario.

Existen varios tipos de sistemas de mensajería:

- Global System Msg - Abarca todo tipo de mensajes relacionados con el sistema DFP para dar soporte al proceso de sonorización de la escena, incluye información de seguridad, información de creación o destrucción de estructuras importantes, información de incidencias, etc.

- Global Event Msg - Da soporte al seguimiento del funcionamiento del plugin en tiempo de ejecución con información como los escaneos a los operadores de evaluación, el registro de activación de operadores de acción, registros de ejecución de eventos, etc.

Con la siguiente imagen se pretende mostrar el ciclo de vida y el tratamiento de un DFP_GlobalMsg sin entrar en ámbitos de seguridad (Ilustración 72).

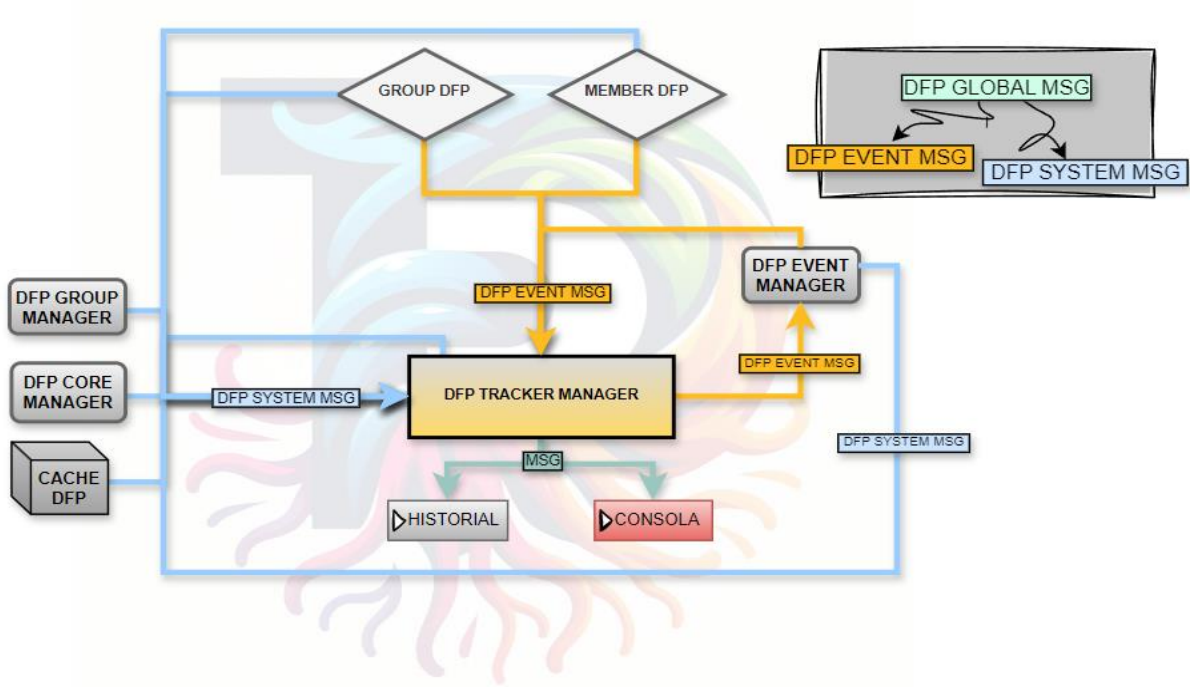


Ilustración 72 Ciclo de vida y tratamiento DFP Global MSG

El Tracker cuenta con estructuras para la formalización de los datos finales en forma de mensajes de texto que pueden ser construidos por ambos sistemas de mensajería. Se dividen en:

- Log
- Warnig
- Error.

La información final generada se lanza en la consola de Unity o se puede almacenar en estructuras auxiliares para su posterior tratamiento o evaluación. Esta información se puede filtrar o configurar a través del editor del Tracker.

Sencillez en los procesos: Para facilitar la posibilidad al usuario de filtrar toda la información en el proceso de sonorización, el Tracker Manager ofrece una vista rápida de campos configurables con los que realizar este proceso.

6.5.8 Sistema de depuración del Tracker Manager

Para ofrecer un tratamiento neutro a toda la mensajería del plugin se crea el sistema de comunicación "Global".

Para que los módulos DFP puedan hacer uso del sistema de comunicación se pone a su disposición la clase DFP_GlobalMsg que define un tipo de mensaje "global" y que se debe definir antes de ser mandado.

El sistema de comunicación entiende actualmente "dos idiomas":

- Sistema de comunicación "System": Comunicación general de los sistemas DFP

- Sistema de comunicación "Event": Comunicación general especializada en Eventos DFP

"System" cuenta con los tipos de mensajes:

- Normales: No conllevan impacto en el orden de ejecución ni altera las funcionalidades del plugin.

- Peligrosas: Informan de peligro en las funcionalidades

- Graves: Informan de gravedad en la integridad del plugin

Los mensajes de los eventos necesitan de un diseño más característico.

A cada función del Gestor de Grupos que afecte a los Atributos DFP de los Miembros DFP de forma directa, se le asigna un identificador único. De forma análoga, se realiza el mismo proceso con los que sean necesarios, aunque no sean atributos directos.

Adicionalmente el Gestor de Grupos contará con una tabla relacional que asocia los identificadores de funciones con los operadores de trinkets. A continuación, se muestra una tabla relacional de operadores de funciones de Manager con atributos de Miembros DFP.

Identificador Operador de Función	Operadores de Evaluación asociados
001 SetVolume	ATCH, ATDRV, ATEXV
002 RestoreVolume	ATCH, ATDRV, ATEXV, ATRES
003 SetMute	ATCH, ATDRV, ATEXV
004 SetPitch	ATCH, ATDRV, ATEXV
005 RestorePitch	ATCH, ATDRV, ATEXV, ATRES
006 SetPause	ATCH, ATDRV, ATEXV
007 SetAttributes3D	ATCH, ATDRV, ATEXV
008 RestoreAttributes3D	ATCH, ATDRV, ATEXV, ATRES
009 SetReverbLevel	ATCH, ATDRV, ATEXV
010 RestoreReverbLevel	ATCH, ATDRV, ATEXV, ATRES

Tabla 1 Relación funciones de atributos y operadores de evaluación

Como se puede observar, múltiples funciones diferentes pueden estar asociadas exactamente con la misma lista de operadores. Esto puede provocar que el Trinket tenga que consultar muchos datos por una simple notificación, pudiéndose dar situaciones de largas evaluaciones en cada frame. Para evitar esto, se asigna un identificador único a cada tipo de Atributo DFP que sea configurable y se añade como nuevo campo a la tabla:

- Volume: 001
- Mute: 002

- Pitch: 003
- Attributes3D: 004
- ReverbLevel: 005
- Pause: 006
- Miembro DFP: 007
- Grupo DFP: 008
- etc

Identificador Operador de Función	Identificador Atributo Sonoro	Trinket Operators asociados
001 SetVolume	001	ATCH, ATDRV, ATEXV
002 RestoreVolume	001	ATCH, ATDRV, ATEXV, ATRES
003 SetMute	002	ATCH, ATDRV, ATEXV
004 SetPitch	003	ATCH, ATDRV, ATEXV
005 RestorePitch	003	ATCH, ATDRV, ATEXV, ATRES
006 SetPause	006	ATCH, ATDRV, ATEXV
007 SetAttributes3D	004	ATCH, ATDRV, ATEXV
008 RestoreAttributes3D	004	ATCH, ATDRV, ATEXV, ATRES
009 SetReverbLevel	005	ATCH, ATDRV, ATEXV
010 RestoreReverbLevel	005	ATCH, ATDRV, ATEXV, ATRES

Tabla 2 Relación operadores de función, identificadores de atributo y operadores de evaluación asociados

Cuando un Miembro DFP ejecuta una instrucción que modifique alguno de sus atributos, y además pertenece a un OWNER, envía una notificación de operación al Gestor de Grupos con el formato:

- Int: Identificador único de Miembro DFP
- Int: Identificador único de Operación de función

El tratamiento de estas notificaciones se cede a la herramienta Event Manager.

6.5.9 Event Maker

Event Maker es la herramienta que da soporte a la creación de Eventos DFP en Editor Time. Su funcionalidad en este caso es la de ofrecer una interfaz intuitiva para que el usuario pueda construir Eventos DFP de forma sencilla.

Los eventos se componen de:

- Trinket: Gestor encargado de la Entidad Propietaria y los Operadores de Evaluación.
- Action: Gestor encargado de la Entidad Objetivo y los Operadores de Acción

Los Eventos DFP deben seguir ciertas reglas:

- Cada Evento DFP debe estar representado por una entidad independiente de otros Eventos DFPs.
- Los Eventos DFP de una escena deben estar gestionados por el mismo gestor (Event Maker).

Los Eventos DFP de una escena SÓLO pueden formar Entidades Propietarias y Entidades Objetivos con los Miembros DFP y Grupos DFP que formen parte de esa misma escena.

6.5.10 Event Manager

Cuando se inicia una escena entran en juego una serie de factores que son completamente distintos a los de Editor Time.

Supongamos que un el Trinket que define un Evento DFP determinado contiene un único operador de evaluación, [ATDRV] (atribute drop value), este se ha configurado a "0.2" y se ha seleccionado el Atributo DFP "Volúmen". El disparador del Trinket debe activarse cuando el volúmen la Entidad Propietaria baje de valor 0.2,

pero, ¿cómo sabe el Evento DFP que el volúmen se ha modificado? ¿debe estar preguntando constantemente por el volúmen? ¿Debe el Miembro DFP saber que tiene que informar a "alguien" cada vez que modifique su volúmen? ¿y si fuera el pitch? ¿o uno de los niveles de reverberaciones? ¿Cada cuánto y bajo qué condiciones se deben evaluar los operadores?, etc.

Se plantean una serie de interrogantes que puede provocar un gran impacto en el rendimiento del plugin en función de cómo se resuelva ya que se pretende que múltiples Eventos DFP puedan estar actuando de forma simultánea sin que implique un gran impacto en los tiempos de ejecución o en la memoria en uso de la aplicación.

Para afrontar esto se crea la figura del Event Manager, el cual toma control de los eventos en Run Time.

Event Manager utiliza algoritmos de actualización que están diseñados para no abusar de las consultas. En vez de un sistema activo o uno de observación, se crea un sistema reactivo. Se mantiene a la espera y solo inicia acciones cuando hay algo que tratar y solo analiza sus estructuras si cuenta con información necesaria como para crear opciones factibles de evaluación.

Estos algoritmos de actualización tienen en cuenta el limitador de evaluación para las iteraciones de actualización y se basan en lo siguiente:

- No se harán comprobaciones de evaluaciones si no hay un mínimo indicio de que la evaluación global se pueda cumplir
- Por cada Evento el manager cuenta con un vector de evaluación.
- Los vectores de evaluación cuentan con un dígito de estado independiente por cada operador de evaluación.
- Cada vez que un actor DFP utiliza el sistema de mensajería de eventos, se mete el evento en cola y se comprueba si hay suficientes evaluaciones como para cumplir la evaluación global de alguno de sus eventos.
- Si existen notificaciones suficientes en cola como para poder evaluar un evento de forma global, se procesan todas las notificaciones en cola.
- El procesamiento de una notificación termina en la actualización de algún dígito de estado, de alguno de los vectores de evaluación.

- Al terminar de evaluar todas las notificaciones de la cola, se comprueba si algún evento cumple su evaluación global.
- Si es así, se procesa el Evento
- Si no, se recalcula el mínimo de operadores que se tienen que evaluar para que se pueda cumplir una evaluación global de un Evento cualquiera.

Para entender el concepto de Evaluación Global, supongamos que un evento cuenta con 4 operadores de evaluación y el limitador está al 75%. Esto quiere decir que si se cumplen 3 de las 4 evaluaciones se procederá a activar el módulo de Action con sus temporizadores. Su "evaluación global" satisfactoria sería de "3", porque necesita 3 operadores de evaluación para activarse.

A efectos prácticos, también quiere decir que si tenemos 2 notificaciones en cola, podemos garantizar que nunca se cumplirá la evaluación global del evento.

6.5.11 Caché y sistema de protección "Anti Patosos"

La caché es otra de las principales herramientas DFP y es autónoma. Realiza un constante control sobre los módulos esenciales, las herramientas y los datos sensibles que gestiona el plugin para funcionar. El usuario, en su desconocimiento o de forma intencionada puede corromper o borrar partes fundamentales del plugin lo que dejaría inservible la escena.

La caché cuenta con sistemas de protección. Parece "como que no están", pero realizan una cantidad importante de tareas de seguridad y administración del plugin para facilitar al usuario el uso del Plugin DFP.

Gestionan la protección de los módulos importantes del plugin, ofrecen cobertura para la resolución de incidencias leves y graves que pongan en peligro la integridad del plugin y consecuentemente la integridad de la escena del proyecto.

El usuario, en su desconocimiento o de forma intencionada puede corromper o borrar partes fundamentales del plugin, como podría ser el Manager principal de la escena. La caché detectará la incidencia y procederá en función de esta pero siempre seguirá ciertas directrices que le dé tiempo al usuario a hacer cosas:

- En primera instancia avisará al usuario
- Si el usuario no responde, intentará recuperar todos los datos posibles
- Si le es posible recuperar datos, intentará recuperar las estructuras dañadas y realimentar los datos que hubiera perdido
- Si no fuera posible recuperar estructuras dañadas, las instanciará de nuevo e intentará realimentar los datos que pueda.
- Si no fuera capaz de realimentar datos, reiniciará todos los sistemas de datos y reiniciará las herramientas como medida extrema.

Ilustración 73 Caché miss

Si no se ejecuta ninguna acción, la caché levantará todos los módulos necesarios que necesite para hacer el plugin funcional (Ilustración 74).

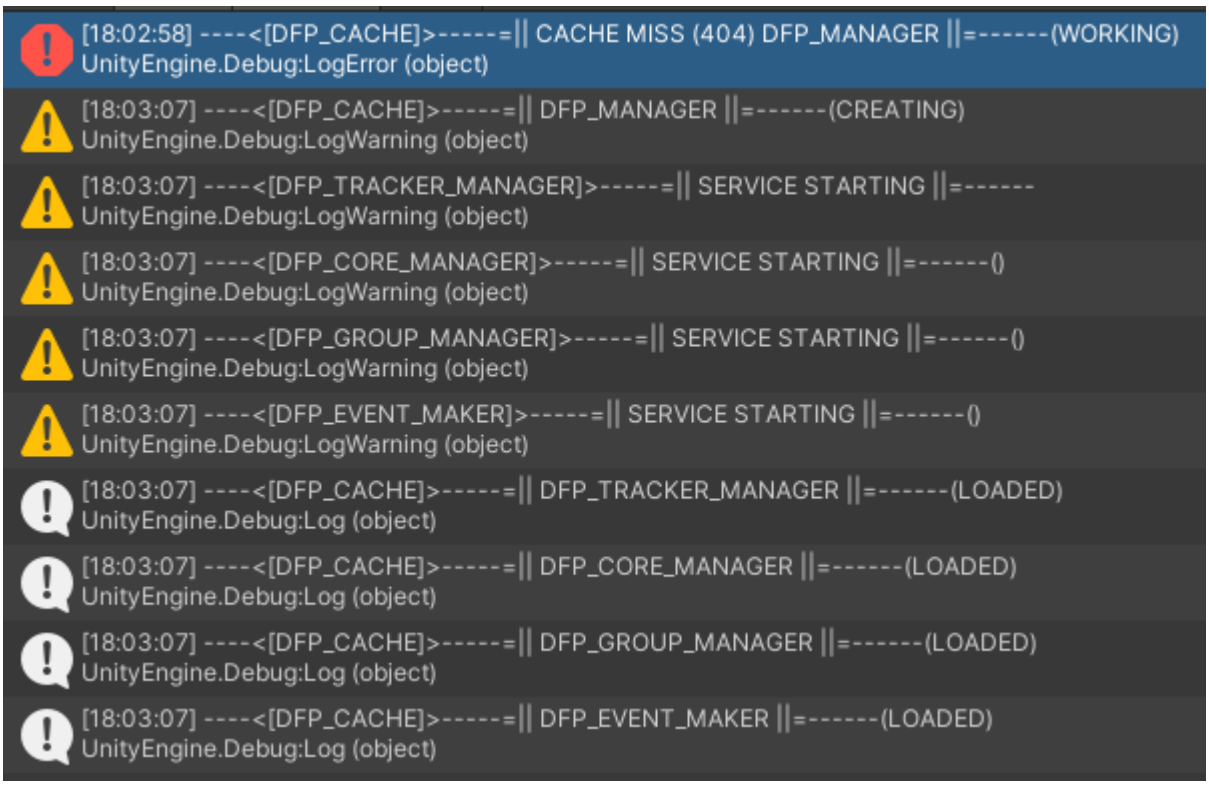


Ilustración 74 Protección caché

Algunos de estos módulos son más importantes que otros debido a que están preparados para trabajar en Editor Time, sin embargo, siempre intentará levantar el sistema de tracking para que el resto de las herramientas pueda seguir interactuando como si nada de esto estuviera ocurriendo.

Además, el sistema de protección realiza todo tipo de tareas de mantenimiento en vivo para que el plugin siga funcionando. Por ejemplo, un Miembro DFP forma parte de distintos Eventos DFP.

El usuario, por descuido o decisión propia decide borrar el Miembro DFP sin remover antes dicho status por la forma correcta desde el Editor de Miembros DFP.

Cuando un módulo de una herramienta se borra o se corrompe, se suele detener el uso de dicha herramienta. En DFP se apuesta por permitir al usuario que pueda cometer errores sin conocimiento de qué está ocurriendo y que el propio plugin sea capaz de resolverlos y de auto sanearse. Para ello se implementa el Sistema de protecciones "anti-patosos".

El plugin detectará la incidencia y la solucionará sin provocar errores ni pedir interacciones con el usuario. Entre otras cosas, actualizará todos los Eventos DFP, referencias rotas del antiguo Miembro DFP, limpiará los Grupos DFP, etc.

En conjunto, las protecciones se pueden englobar en:

- Módulos importantes: Protección para módulos como gestores del plugin.
- Soporte para editores: Protección para alimentar y proteger editores y ventanas ya que el plugin en gran parte se compone de interfaces.
- Datos sensibles: Protección ante acciones imprudentes o bruscos que no siguen los procesos habituales de uso del plugin como borrado de usuarios, de ficheros de soporte .asset, etc.
- Protección para los sistemas de protección: Protección activa de los sistemas que dan protección al plugin. Se hablará de esto a continuación.
- Otras protecciones.

Como se ha comentado en otra de las secciones, el talón de Aquiles del Plugin DFP es la Caché. Es el sistema que permite que todo funcione y a la vez, podría ser el único responsable de que todo deje de funcionar.

Para solventar esto, se instaura el sistema de protección auxiliar en la red de Editores y Ventanas de Edición DFP (sistemas de protección activo). Pero.. ¿qué pasa con los datos?

Si la caché pierde su configuración y es quién coordina todo, la información puede corromperse. Para evitar esto se crea un Actor DFP auxiliar [DFP] Recovery_Data(Ilustración 75).

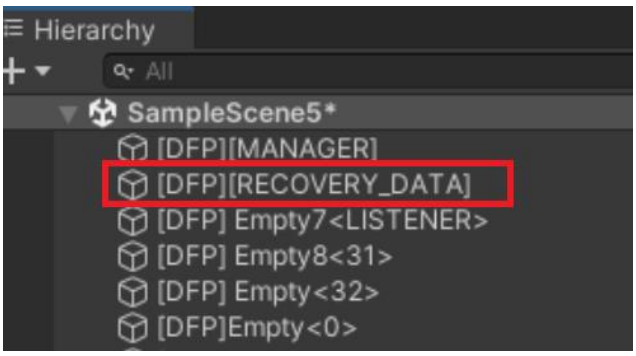


Ilustración 75 Objeto instanciado DFP Recovery Data

DFP_ RecoveryData es una estructura que utiliza la Caché para almacenar una copia segura de los datos (Ilustración 76).

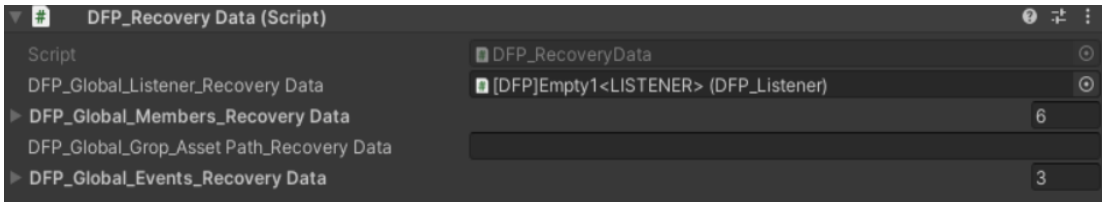


Ilustración 76 Estructura de datos DFP Recovery Data

El uso de copia de datos puede provocar incoherencia de datos. Para evitar incoherencia de datos ante pérdidas de módulos se implementan las siguientes políticas de actualización de datos en todas las herramientas DFP y la Caché:

1. El único Actor DFP que actualizará la copia de datos será la Caché. Existen sistemas de protección para los módulos y las herramientas, y a su vez las propias herramientas protegen al sistema de protección. No se necesita más de un administrador de los datos de respaldo y además podría dar lugar a incoherencia de instrucciones de actualización.

2. Siempre que se actualice un dato que afecte al Plugin, en cualquiera de sus herramientas, en cualquiera de sus módulos, se actualizará la copia de respaldo. No tiene sentido establecer un sistema de cola de actualización debido a que esto se reproduce en tiempo de edición principalmente por lo que prima la idea de realizar

la copia de seguridad lo antes posible. Si pasa 1 hora sin ningún cambio, no se realizará ninguna copia y si en un minuto se registran 1000 cambios, se actualizará los datos de respaldo 1000 veces, esa es la intención.

3. Jamás se volcarán datos de respaldo sobre Actores o Herramientas si no ha ocurrido antes alguna incidencia. No tiene sentido realizar un volcado de datos desde la copia de seguridad sin que esté justificado previamente por algún hecho, por lo tanto, jamás se hará.

4. Las copias de datos que se restauren vendrán precedidas por al menos una incidencia de carácter GRAVE. Las diferentes incidencias que trata el Plugin son:

- Normales: No conllevan impacto en el orden de ejecución ni altera las funcionalidades del plugin. Las incidencias normales nunca traen restauración de datos.
- Peligrosas: Informan de peligro en las funcionalidades, normalmente porque alguno de los módulos o herramientas aparentemente no está localizado o ha sido alterado. A veces simplemente ocurre por cambios en los tiempos de Ámbito de Trabajo, por ejemplo, al pasar de Editor Time a Run Time se instancian en ejecución los objetos de la escena, entre ellos los Managers, razón por la que la Caché, necesita refrescar sus referencias. Una incidencia peligrosa puede conllevar restauración de datos.
- Graves: Informan de gravedad en la integridad del plugin. Suele ser porque un módulo importante se ve alterado de su estado natural, por ejemplo, borrar el fichero .asset de la Caché. Estas incidencias pueden alterar gravemente el plugin y están preparadas para que la respuesta sea del plugin sea mucho más rápida.

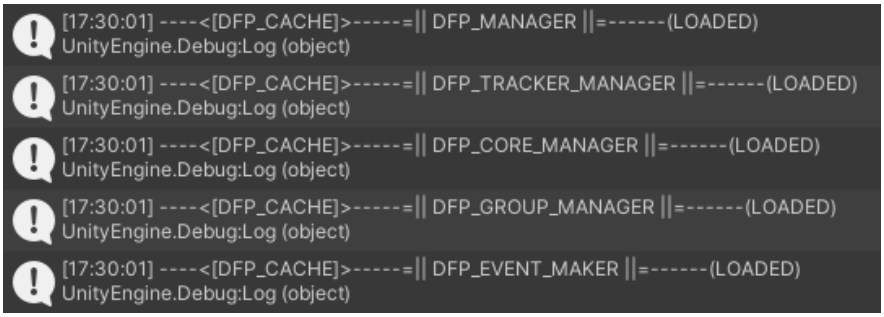


Ilustración 77 Informes Caché

Adicionalmente la Caché ofrece otro tipo de soporte como respaldo a herramientas que lo necesitan, por ejemplo, al Tracker Manager. Perder un informe de evaluación o acción puede ser determinante para el devenir final de los Eventos DFP. Se debe garantizar que bajo ningún concepto se pierde alguna traza de datos. Por ello se implementa el siguiente proceso de respaldo al Tracker (Ilustración 78):

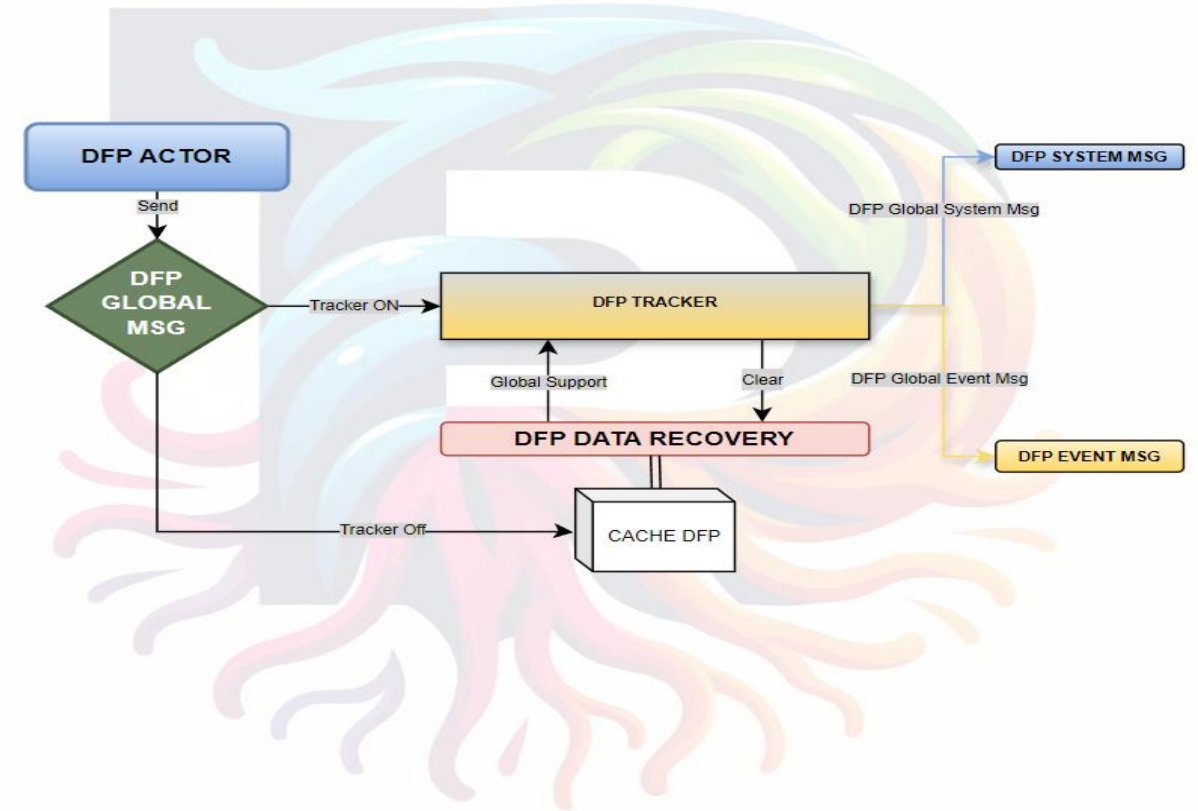


Ilustración 78 Global Support Caché

Una estructura auxiliar que ofrezca un respaldo para los tiempos de incidencia.

Capítulo 7 - Trabajo Futuro

Una vez terminado el desarrollo del plugin dentro del presente trabajo fin de grado, y considerando las posibilidades que puede llegar a ofrecer, se abre la posibilidad de implementar nuevas funcionalidades adicionales muy interesantes. A continuación, se muestra un listado de ellas clasificadas en función de lo que intentan tratar.

Funcionalidades Extras

1. Un recurso muy utilizado en los juegos modernos se basa en implementar instancias individuales a los jugadores basados en canales. Estas instancias pueden contener elementos comunes o ser individuales para cada jugador. Sería interesante que el plugin contara con la funcionalidad de permitir varios listener y que cada listener tenga su propio "sistema de audio" con el plugin, es decir, una especie de "instancia" privada para cada uno de los listener, de tal forma que cada uno de ellos pudiera trabajar con sus propios eventos, configuraciones, grupos, etc.

2. Implementar "sistema de audio" compartido que se pudiera compartir entre varios listeners, de tal forma que cada listener pudiera tener su propio sistema de audio y ambos contaran con uno compartido, esto eliminaría la duplicidad de eventos para distintos usuarios y aseguraría una coordinación en el sistema de lanzamiento de Eventos cooperativos.

3. Implementación de una herramienta que modifique las Interfaces para ajustar su temática, pudiendo incluso personalizar los colores de cada item de la interfaz de forma individual para mejorar la adaptabilidad al usuario, reforzando la accesibilidad a personas daltónicas.

4. Evento de multi eventos encadenados. Permitir la posibilidad de que un evento pueda definir eventos que se encadenan, es decir, un evento que actúa como padre y contiene eventos hijos que pueden ser lanzados por el padre en el orden de ejecución que desee y que además cuente con un

soporte de datos de entrada / salida que puedan usarse como input y como output de forma homogénea. Por ejemplo, la salida del Hljo 3 tiene que ser la entrada del Hijo1, por lo que al terminar el Hljo 3 este ejecuta el Hljo 1 y además cede el output del Hijo 3 como input al Hljo1.

5. Operadores de Evaluación y Operadores de Acción que llamen funciones como una especie de gestor de CallBacks o llamadas lambda para invocar de forma dinámica funciones a través de Eventos DFP.

6. Grupos de Grupos. Aunque existe el símil de "Grupo de Grupos" en el proyecto, no se creó con esa intencionalidad y consecuentemente no cuenta con la api necesaria para su gestión, sin embargo, sería un proceso relativamente sencillo de hacer y ampliaría las funcionalidades de los Grupos DFP.

Seguridad

1. Control de confirmación para herramientas como "Clear Plugin" y acciones irreversibles que implican la destrucción de datos de forma definitiva, como el borrado de Eventos DFP, Grupos DFP, Miembros DFP, etc.

2. Sistema de permisos para los eventos. Esta idea se deseaba implementar y se hizo un estudio sobre cómo implementar los permisos en el plugin, se trata de implementar un sistema de permisos que haga que las interacciones con el sistema de Eventos en ejecución tengan que ser autorizadas por un sistema de permisos. De esta forma, se evitaría que cualquier entidad pueda invocar cualquier función que pueda alterar el transcurso natural del procesamiento de los datos del plugin en tiempo de ejecución.

3. Respaldo de datos relacionados con la escena fuera de esta. Figuras como los ScriptableObjects están preparados para serializar y tratar datos de forma eficiente en el entorno de Unity pero no permiten almacenar contenido relacionado con la escena. Sin embargo, sería posible crear una entidad que serialice e invierta el proceso de serialización de objetos de escena con

identificadores alpha numéricos. Esto permitiría recuperar datos aún cuando la escena se corrompe y daría la posibilidad al desarrollador a recuperar parte de su trabajo.

Capítulo 8 - Conclusiones

Tras finalizar el proyecto se ha conseguido implementar un plugin que integra Fmod como motor de audio profesional por script (21) en Unity como plataforma de desarrollo.

El plugin cuenta con una serie de Actores DFP y Herramientas DFP que, en su conjunto, ofrecen al usuario una alternativa de sonorización de escenas. El plugin se comporta a efectos prácticos como un constructor. Permite que el usuario vaya construyendo conjuntos de datos que son objetos de tratamiento con el fin de garantizar la coherencia y la seguridad de los datos que genera el usuario en el uso cotidiano del plugin. Entre las herramientas del plugin se destaca Event Maker como herramienta que permite la programación visual de Eventos DFP.

Los Miembros y Grupos DFP cuentan con comportamientos novedosos que permiten interacciones como crearlos o modificarlos en ejecución.

El plugin es rápido en Editor Time o Tiempo de Edición gracias a sus editores incrustados como editores nativos en la interfaz de Unity mientras que el comportamiento de sus herramientas y la liberación de carga de edición hace que sea rápido en Run Time o Tiempo de Ejecución.

Además, el plugin cuenta con un Tracker configurable que facilita la depuración, filtrado y análisis del uso del software en cualquiera de los ámbitos de trabajo de la plataforma.

Capítulo 9 - Conclusions

After finishing the project, a plugin has been implemented that integrates Fmod as a professional audio engine for scripting (21) in Unity as a development platform.

The plugin has a series of DFP Actors and DFP Tools that, together, offer the user an alternative for sounding scenes. The plugin behaves, for practical purposes, as a constructor. It allows the user to build data sets that are objects of treatment in order to guarantee the coherence and security of the data that the user generates in the daily use of the plugin. Among the plugin's tools, Event Maker stands out as a tool that allows the visual programming of DFP Events.

DFP Members and Groups have novel behaviors that allow interactions such as creating or modifying them at runtime.

The plugin is fast in Editor Time thanks to its editors embedded as native editors in the Unity interface while the behavior of its tools and the release of editing load makes it fast in Run Time. In addition, the plugin features a configurable Tracker that facilitates debugging, filtering and analyzing the use of the software in any of the platform's work areas.

Capítulo 10 - Bibliografía

(01) How to write native Plugins [En línea]. - https://www.youtube.com/watch?v=Ya4DNfMH6jo&ab_channel=BoosterSpace.

(02) Starting Making a Custom Editor [En línea]. - https://www.youtube.com/watch?v=RInUu1_8aGw&ab_channel=Brackeys.

(03) Editor in Unity [En línea]. - https://docs.unity3d.com/ScriptReference/Editor.html.

(04) Unity Editor Window [En línea]. - https://docs.unity3d.com/ScriptReference/EditorWindow.html.

(05) Editores Personalizados Extendidos [En línea]. - https://edom18.medium.com/using-customeditor-extension-85b5db4e8994.

(06) Introduction to Custom Editor Window in Unity [En línea]. - https://www.youtube.com/watch?v=34736DHWzaI&list=PLPNV77N9nGhUV_q4tlXx0hz-pjhtcRl2w&index=11&ab_channel=TurboMakesGames.

(07) Tips para Unity Editor [En línea]. - https://www.youtube.com/watch?v=b7vY6id63OY&list=PLPNV77N9nGhUV_q4tlXx0hz-pjhtcRl2w&index=7&ab_channel=LostRelicGames.

(08) Layouts Editores [En línea]. - https://docs.unity3d.com/ScriptReference/EditorGUILayout.Popup.html.

(09) GUI Layouts [En línea]. - https://www.youtube.com/watch?v=tlKNk6jH1JY&list=PLPNV77N9nGhUV_q4tlXx0hz-pjhtcRl2w&index=12&ab_channel=TurboMakesGames.

(10) Unity 2013 Advanced Editor Scripting [En línea]. - https://www.youtube.com/watch?v=t-wShOv8c1E&list=PLPNV77N9nGhUV_q4tlXx0hz-pjhtcRl2w&index=27&ab_channel=Unity.

(11) Property Drawer [En línea]. - https://www.youtube.com/watch?v=9Z-alV6LZkg&list=PLPNV77N9nGhUV_q4tlXx0hz-pjhtcRl2w&index=19&ab_channel=KasperDevUnity.

(12) Generación de scripts y componentes a través de editores [En línea]. - https://discussions.unity.com/t/is-there-any-way-to-create-a-script-from-an-editor-window/850492.

(13) Serialización en Unity [En línea]. - https://docs.unity3d.com/es/530/Manual/script-Serialization.html.

(14) Introducción a la serialización y al tratamiendo de datos [En línea]. - https://www.gamedeveloper.com/business/introduction-to-unity-serialization-and-game-data.

(15) Scriptable Objects [En línea]. - https://www.kodeco.com/2826197-scriptableobject-tutorial-getting-started.

(16) Serialización a través de scripts [En línea]. - https://docs.unity.cn/560/Documentation/Manual/script-Serialization.html#:~:text=Unity%20uses%20serialization%20to%20load,as%20MonoBehaviour%20components%20and%20ScriptableObjects.

(17) Audio Integration workflow with Fmod & Unity [En línea]. - https://www.youtube.com/watch?v=5jYSmq9Xqb0&ab_channel=ProSoundEffects.

(18) Fmod Game Components [En línea]. - https://www.fmod.com/docs/2.01/unity/game-components.html#studio-parameter-trigger.

(19) Fmod Api [En línea]. - https://www.fmod.com/docs/2.00/api/welcome.html.

(20) Fmod Channel Group [En línea]. - https://www.fmod.com/docs/2.01/api/core-api-channelgroup.html.

(21) Ejemplos Scripting Fmod [En línea]. - https://www.fmod.com/docs/2.00/unity/examples-timeline-callbacks.html.

(22) Integración Fmod Event Emitter [En línea]. - https://mediasound.indiana.edu/gqaudio/fmod-unity-21.html.

(23) Sonido Fmod para videojuegos Toni Oliveros [En línea]. - https://www.youtube.com/watch?v=EUIvAhHyZtA&ab_channel=ElSVCanal.

(24) Fmod Event Emitters [En línea]. - https://mediasound.indiana.edu/gqaudio/fmod-unity-21.html.

(25) Fmod Event Instance [En línea]. - https://www.fmod.com/docs/2.02/api/studio-api-eventinstance.html#studio_eventinstance_setpitch.

(26) Manipulando Fmod Event Instance desde código [En línea]. - https://www.youtube.com/watch?v=wfJBIdlOD4Q&list=PLhGo7L_BAS83H5st1sLvDfwCyHzEZzHjY&index=10&ab_channel=ElSonidistaFantasma.

(27) Administración Fmod [En línea]. - https://qa.fmod.com/t/sound-control-through-channel-groups/20594.

(28) Fmod Bus [En línea]. - https://alessandrofama.com/tutorials/fmod/unity/mixer#declaring-an-fmod-bus.

(29) Audio System in Unity with Fmod - Structure, basic groups [En línea]. - https://www.youtube.com/watch?v=rcBHIOjZDpk&ab_channel=ShapedbyRainStudios .

Capítulo 11 - Apéndices

Apéndice A – Lecciones aprendidas

El motor de audio de Fmod se basa en C++, por ello se dedicó inicialmente una gran parte de la investigación en esta orientación y se intentó realizar una implementación como plugin nativo (1) para Unity en C++.

Cuando se habla de "Plugins Nativos para Unity" (1) se hace referencia a una biblioteca externa que se integra directamente en un proyecto de Unity para aportar funcionalidades que no existen de forma nativa en la plataforma de desarrollo, escrito en lenguajes de programación de bajo nivel y que implica gestionar aspectos tan importantes como la compatibilidad con las distintas versiones, sistemas operativos, drivers, controladores y mantener en foco los posibles problemas de rendimiento asociados.

Por desgracia Unity trabaja en primer plano con C# por lo que es necesario crear una capa de abstracción entre C++ y C#.

Se hizo un primer prototipo del plugin con una interfaz en C que hiciera de intermediario entre el plugin y Unity pero surgieron problemas de compatibilidad con la API de Fmod. Cuando se definían cabeceras en la interfaz del plugin con elementos relacionados directamente con el motor de Fmod, surgía la necesidad de crear también las interfaces para dichos elementos. A groso modo es lo que ofrece "Fmod for Unity" pero enfocado a C#, lenguaje con el que se suele trabajar en Unity.

El desarrollo de la interfaz en C estaba ocupando una gran parte del tiempo del proyecto por lo que se decidió descartar este prototipo y optar por la versión en C#, esta vez, incrustado como recurso asociado a un "Plugin basado en Assets" (*) en lugar de como "Plugin Nativo". (1)

(*) Entiéndase "Plugin basado en Assets" como una complemento o herramienta que se añade a un proyecto y que utiliza recursos activos en forma de Assets para funcionar, es decir, en lugar de crear un plugin personalizado utilizando únicamente programación de bajo nivel y partiendo desde cero, se pueden utilizar assets previamente creados por otros desarrolladores y cargados en el proyecto, en este caso se utiliza para el desarrollo el asset "Fmod for Unity" creado por la empresa Firelight Technologies y que es de libre acceso y uso, si el fin no es comercial.

Apéndice B – Guía de Instalación

Para que el plugin funcione correctamente es necesario realizar los siguientes pasos:

1. Descargar e instalar la plataforma de desarrollo Unity 2021.3.11.f1
2. Descargar la versión X del motor de audio "Fmod for Unity" e importarlo en el proyecto de Unity.
3. Descargar e instalar la versión Y de la versión Studio de Fmod
4. Descargar el plugin de filtrado dinámico DFP e importarlo en el proyecto de Unity.

Paso 1. Plataforma de desarrollo, Unity 2021.3.11.f1

Descarga Unity 2021.3.11.f1 y realiza una instalación correcta de la plataforma. Aunque se ha probado el plugin en diferentes versiones sin encontrar fallos aparentes, el desarrollo principal se hizo en esta versión por lo que se recomienda su uso.

Puedes descargar el instalador desde aquí.

Paso 2. Instalación e integración de "Fmod for Unity" en Unity

Fmod cuenta con dos secciones en su zona de trabajo: la sección de Studio y la sección principal o interfaz de programación de aplicaciones, conocida como "Core Api".

"Fmod for Unity" cuenta con todo el core del motor de Fmod adaptado a Unity. Es requisito tenerlo instalado en nuestra plataforma de desarrollo para poder utilizar el plugin. Podremos obtenerlo gratuitamente en la Asset Store de Unity.

Para descargar e importar Assets desde la Store de Unity hay que seguir los siguientes pasos:

2.1.1 Installing the Plugin

Follow these steps to install FMOD for Unity:

1. In your Unity project, open the Asset Store (select "Window > Asset Store" from the Unity menu bar) and search for "FMOD for Unity".
2. Select the FMOD for Unity asset and click the Download button.
3. Once the download completes, click the Import button.
4. Once the Import Unity Package window appears, leave all of the assets ticked and click the Import button.
5. We recommend that you **disable the Unity built-in audio** on all platforms to prevent it from conflicting with FMOD.
6. This completes the plugin installation. Now you can **set up access to your FMOD Studio content**.

Ilustración 79 Instrucciones de instalación

Si necesitas más información puedes encontrarla aquí.

Paso 3. Fmod Studio para creación y exportación de bancos de sonido a Unity

Como se explicará más adelante, sólo es necesario el core de Fmod para que el plugin funcione correctamente en Unity, sin embargo, se recomienda complementar el desarrollo con la versión Studio, ya que esto facilitará procesos como la creación de bancos de sonidos y su posterior exportación a Unity.

Descarga el paquete de Fmod Studio para PC aquí.

Paso 4. Plugin dinámico de audio DFP

Para instalarlo es necesario abrir proyecto en Unity e importar el plugin en el directorio "Plugins". Si este no existe puede crearse dentro de la carpeta "Assets" antes de importar.

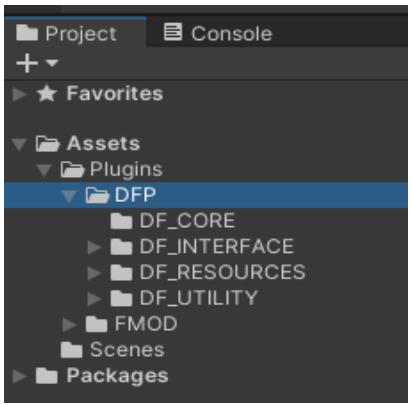


Ilustración 80 Directorios

También podemos hacerlo de forma manual copiando el contenido que se ha descargado del Plugin DFP, dentro del directorio del nuestro proyecto de Unity, sin olvidar hacerlo dentro de la carpeta "Plugins".

De cualquier forma, la ruta del plugin tras haberse importado debe ser:

"/Assets/Plugins/DFP.

Si se han realizado correctamente los pasos anteriores, el resultado debe quedar parecido a la imagen (Ilustración 80) en función de los plugins que se tengan instalados en un proyecto de Unity.

Apéndice C - Creación bancos de sonido FMOD

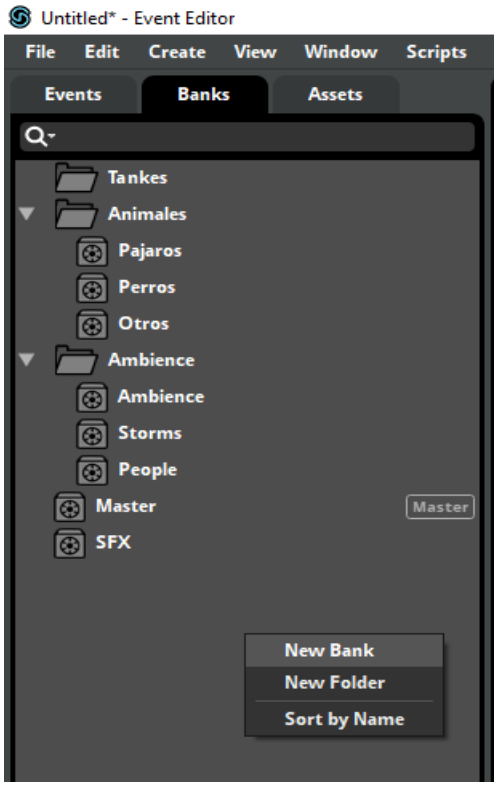


Ilustración 81 Event Editor FMOD

Cuando se trata de sonorizar una escena, necesitaremos sonidos con los que poder realizar dicho proceso. Según "FMOD", un banco es "*una colección de eventos y assets de tu proyecto de FMOD, formateado y comprimido para ser usado en tu juego*".

Hay varias formas de formar nuestros bancos de sonidos para trabajar en Unity. DFP trabaja con Fmod, el cual, como hemos dicho, cuenta con la versión de Studio.

La forma más sencilla de componer nuestros bancos de sonidos es utilizar Fmod Studio. Es la única parte del Plugin en la que se aconseja trabajar fuera del entorno de Unity con objeto de simplicidad del proceso. Para ello, abriremos un nuevo proyecto de Fmod Studio y en la pestaña "Banks", haciendo click derecho, aparecerá un menú que nos permitirá crear sencillamente nuestros bancos de sonido.

Desde la pestaña "File" podremos importar nuestros sonidos con un simple botón para añadirlos a nuestros bancos.

Apéndice D - Link Fmod Studio a proyecto Unity

Para trabajar con un proyecto de Fmod Studio en Unity debemos dirigirnos a la sección de configuración que se encuentra en la pestaña "FMOD > Setup Wizard" del menú superior de Unity (Ilustración 82).

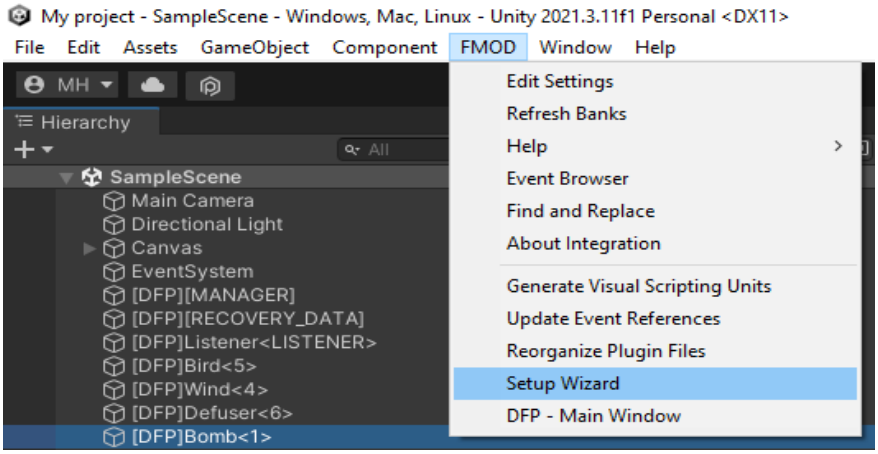


Ilustración 82 Setup Wizard para FMOD en Unity

Este acceso rápido nos abrirá la ventana de configuración de Fmod for Unity. Debemos dirigirnos a la sección que se indica en la imagen a continuación y elegir la ruta donde tengamos nuestro proyecto de Fmod Studio (Ilustración 83).

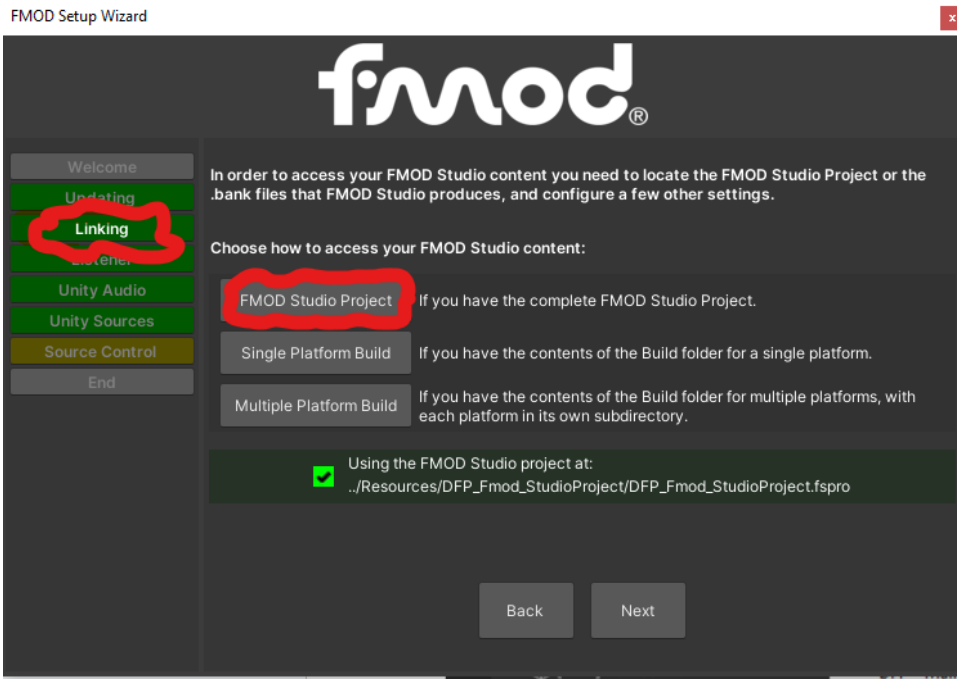


Ilustración 83 Ventana de configuración FMOD For Unity