“Editor Avanzado V0.1” de datos para Unity (Español).

Sumario

1.Introducción. ¿Para que sirve?

La idea de crear este Editor de datos para unity nació debido a la necesidad de reducir a largo plazo el trabajo que suponía crear ventanas para editar datos para cada una de las entidades que creaba.

Para crear los editores tardaba al rededor de 1h, entre hacer que todo se viese bien, colocar las variables y testear todo, y si la entidad era compleja, o tenía muchos parámetros, el tiempo se podía ir a las 2 o 3hs. Así que la creación de un editor general que permitiese crear ventanas adaptadas a las necesidades de la entidad era necesario a largo plazo para evitar perder mucho tiempo en la creación de ventanas.

Además de eso, un editor para cada entidad, significaba hacer 5 ficheros, 1 para cada una de las vistas (crear, modificar y ver), el listado y la base de datos. Con este editor, **no será necesario** crear más ficheros de los ya existentes, **a excepción de la base de datos.**

Ahora, con tan solo configurar los datos que quieres que se muestren en una propiedad “Editor” en cada una de las entidades, se auto genera una ventana con los datos para modificar la entidad.

Sólo funciona con los tipos básicos, objetos de Unity, arrays y diccionarios, pero tengo pensado en que se puedan **crear tablas y secciones personalizadas.**

2.Cómo Funciona

2.1.Editor

Para hacer el editor posible, he hecho uso de Reflection de C#. Toma todos las propiedades definidas en la **propiedad “Editor”** de cada entidad.

El editor avanzado se compone de 4 ficheros en la carpeta Assets>Scripts>Windows>Core.

Cada fichero corresponde a:

- ListWindow: **Lista los elementos** de la base de datos.
- WindowAction: **Enum con las funciones** posibles a hacer desde el editor.
- Window: Se ejecuta al pulsar cualquier elemento de creación, modificación o vista, y se encarga de la **generación de la ventana** con el objetivo de cumplir su acción definida por WindowAction.
- WindowInterface: **Vista de la ventana**, después de gestionarse en “Window”.
- Display: Se usa en “WindowInterface” para **mostrar todos** los datos que **NO son** comunes y se adaptan a la entidad a editar. **Por ejemplo, no genera el botón “Confirm”** del editor, eso lo hace “WindowInterface”.
- AdvancedEditor: Donde está todo lo relacionado a la **creación de campos** del editor.

2.2.Base de datos.

Además de eso, en Assets>Scripts>Windows está la clase “Windows”, ahí se **definen las entidades que tendrán editores.**

Existe una clase “CoreDB<T>” en Assets>Scripts>Data>Database>Core que sirve para **crear bases de datos**, simplemente con crear un fichero “EntityDB.cs” con una clase del mismo nombre, que se pueda serializar en un fichero con la anotación CreateAssetMenu y que extienda de “CoreDB<Entity>”.

2.3.Otros

Existe también algunas clases que pueden ser de utilidad, como el **SerializableDictionary** (no hecho por mi) que permite la serialización de diccionarios, ya que esta clase de parámetros no se puede serializar. Tanto el diccionario base como el diccionario serializable son compatibles con el Editor, pero **no es recomendable** usar el **Diccionario base**, debido a que no se guardará en la BD.

3.Como usar el editor Avanzado

3.1.Crear una entidad y su correspondiente BD

Para usar el editor, se debe **crear una Entidad y una Base de datos**, para facilitar el uso y el desarrollo tanto del editor, como de la base de datos, se **usa** “BaseEntity.cs”, que contiene un ID y Nombre como parámetros; y “CoreDB<BaseEntity>.cs” que contiene todo lo necesario para gestionar datos de BaseEntity y **entidades que extiendan** de BaseEntity.

Para que la entidad pueda tener un editor, se debe crear una entidad T que **extienda** de BaseEntity, y una base de datos que extienda de CoreDB<T> con un c**onstructor que clone** y genere un elemento **vacío con ID**.

3.1.1.Entidad

Crea una clase cualquiera que haga de entidad, y extiéndela de BaseEntity. Deberás definir un constructor para **clonar**, es muy importante que pueda clonar, con las propiedades que manipulen los campos; y para **crear uno por default con un ID** . Además de eso, si lo va a usar para el editor, debe usar **una propiedad “Editor”**, pero de eso se hablará mas adelante en el documento.

3.1.2.CoreDB – Base de datos

Al igual que la entidad, tu base de datos, debe estar en su propio fichero .cs y **extender** de CoreDB<Entidad>. Además, **debe tener la anotación** [CreateAssetMenu(menuName=”Database/Entities”)]. Esto servirá para poder crear desde Unity una base de datos a través de **click derecho Database>Entities.**

CoreDB tiene todo lo necesario para la **gestión básica de Datos**, si necesitase mas cosas para gestionar T entidades, puedes ampliar las funciones en el fichero y clase.

4.Uso del editor

4.1.Crear las ventanas para Editar

Para poder usar el editor, debe crear en Windows>Windows.cs una **clase (puede estar en el mismo fichero todos los editores).** Esta clase debe **extender** de ListWindow<Entity, CoreDB<Entity>> y tener la **anotación** [CustomEditor(typeof(CoreDB<T>))]. Esta parte es para crear las ventanas como tal, pero aparecerán vacías a excepción del ID, Name y el botón de confirmación.

4.2.Propiedad “Editor” en la Entidad

Finalmente, toda la configuración **para editar** los datos, se encuentran aquí.

Crea una nueva propiedad “Editor”, BaseEntity ya contiene una propiedad Editor, **sobrescriba** la en la entidad hija.

En esta propiedad se definirá lo que sería los datos a poder editar y su configuración. Editor será un atributo de tipo “EditorParams”. También en Utils.

EditorParams tiene un constructor que deberá usar en la propiedad: EditorParams(**parametters**, configurations, labels, columns, maxV, maxH, minV, minH, extensionOf), sólo “parametters” es obligatorio.

- Parametters: Array de strings, donde debes poner **los nombres de las propiedades** (pensado para las propiedades get-set), que se podrán editar en el editor, por orden de declaración en el array.
- Configurations: Diccionario<parámetro/propiedad, array de configuraciones>. Esta entrada sirve para asignarle al parámetro pasado **cómo clave** las **características** en el array. Por ejemplo, {“ID”, new[]{“readOnly”}} haría ID de sólo lectura para cualquier situación. Es recomendable usar los valores de “AdvancedEditor.Restrictions” para las restricciones, ya que, aunque se pueda poner manualmente con strings, puede escribirse mal y provocar errores difíciles de detectar. Como el Editor Avanzado hace uso de los valores de esa estructura para la comprobación, es más seguro usar los datos de ahí. También hay restricciones que hace uso de Tuplas, generalmente los relacionados a que necesiten datos (Elegir entre una lista o límites). De igual modo, usar los métodos de EditorParams para crear las Tuplas, ya que se necesitan tipos muy específicos y se ahorra en esfuerzo de crearlas.
- Labels: Diccionario<parámetro, leyenda>, como se **mostrará el nombre del parámetro** pasado como clave a la hora de mostrarlo en el editor. Si no tiene “Label”, se mostrará el nombre de la propiedad.
- Columns: ***No implementado aun***, servirá para **crear columnas** ignorando las áreas.
- Max/Min: Los **tamaños máximos y mínimos** en Vertical(V) u Horizontal(H) de la **ventana** que se creará.
- ExtensionOf: Se le **añaden los atributos** de un EditorParams pasado por este parámetro. Es muy **recomendable usar en esta parte base.Editors** para agregar todos los datos de

entidades padres.

Tras definir la propiedad, si se ha hecho bien, debería aparecer todo correcto en el editor y poder modificar los datos.

5.Anotaciones

- Hay un ejemplo “EditorExample” en el proyecto.
- Se puede usar varias funciones de AdvancedEditor **de forma independiente**, ya que los métodos son públicos. Funcionan correctamente, pero sólo se pueden usar aquellos que hacen algo en específico. Las descripciones vienen en los métodos.
- El mínimo en diccionarios no funciona en todos los casos. Sólo cuando el tamaño es superior al mínimo es cuando no deja bajar del número de elementos del valor mínimo.
- El Editor está en desarrollo y seguramente **necesite muchas mejoras**. Agradecería que todos los que usen el editor, **me notificase todos los errores y recomendaciones** que se le puedan ocurrir.
- Agradecería que **se acredite que he sido el creador** de este Editor en caso de usarlo. Hacer **una mención y poner algún contacto mío y/o Git**. Gracias.

6.Enlaces

- SerializableDictionary: https://discussions.unity.com/t/solved-how-to-serialize-dictionary-with-unity-serialization-system/71474/4