# Advanced [Editor Tools]

Easy Custom Inspectors

By Raúl Martín ~ 05/2023

## Summary

Custom inspectors are essential for Unity developers as they allow for the creation of a user-friendly interface for GameObjects. This package provides the developer with some tools that make it easier to design your custom inspectors and reduce the amount of time spent on writing boilerplate code, enabling developers to focus on the creative aspects of game development.

This is achieved by using *Attributes* on top of your script fields to decide how the inspector will be painted. Among these tools are **Foldouts, Columns,** and **Buttons,** which are essential for every game developer who enjoys an organized and functional workspace.

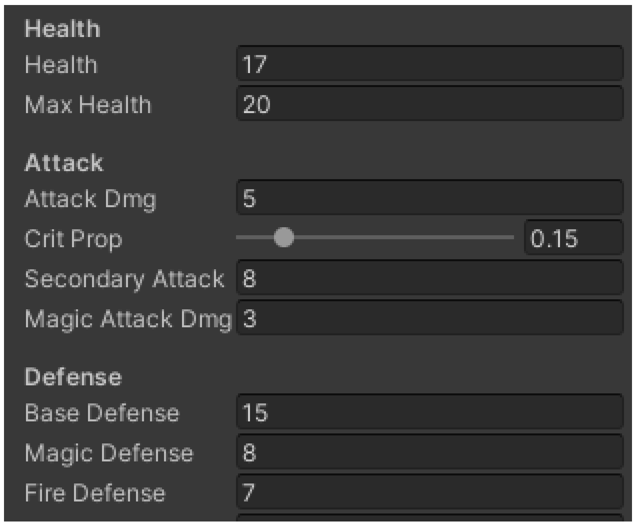
## Table of Contents

**1. Introduction** **1**
- 1.1. List of Tools 2
    - Layout Attributes 2
    - Foldouts - Layout 2
    - Columns - Layout 3
    - Buttons 4
    - Others 5
- 1.2. Combining attributes 5
- 1.3. QOL details 7

**2. Limitations** **8**
**3. Future Work** **8**

# 1. Introduction

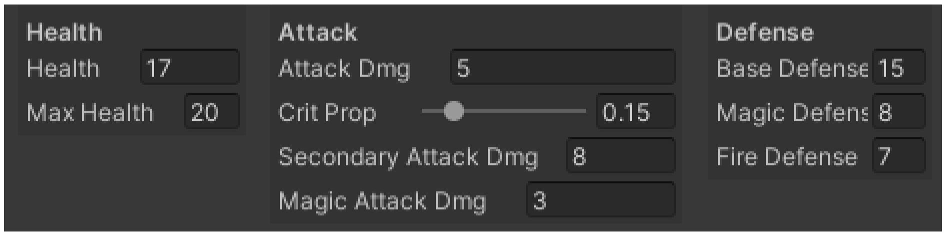
This package provides layouting and functional attributes to customize your inspectors in an easy and flexible manner, without the need to create separate script editor files in your project. These tools have been designed to expand Unity's default property attribute list. In case you don't want to read this documentation, a demo scene has been provided with examples of every available tool.



**Health**
Health 17
Max Health 20

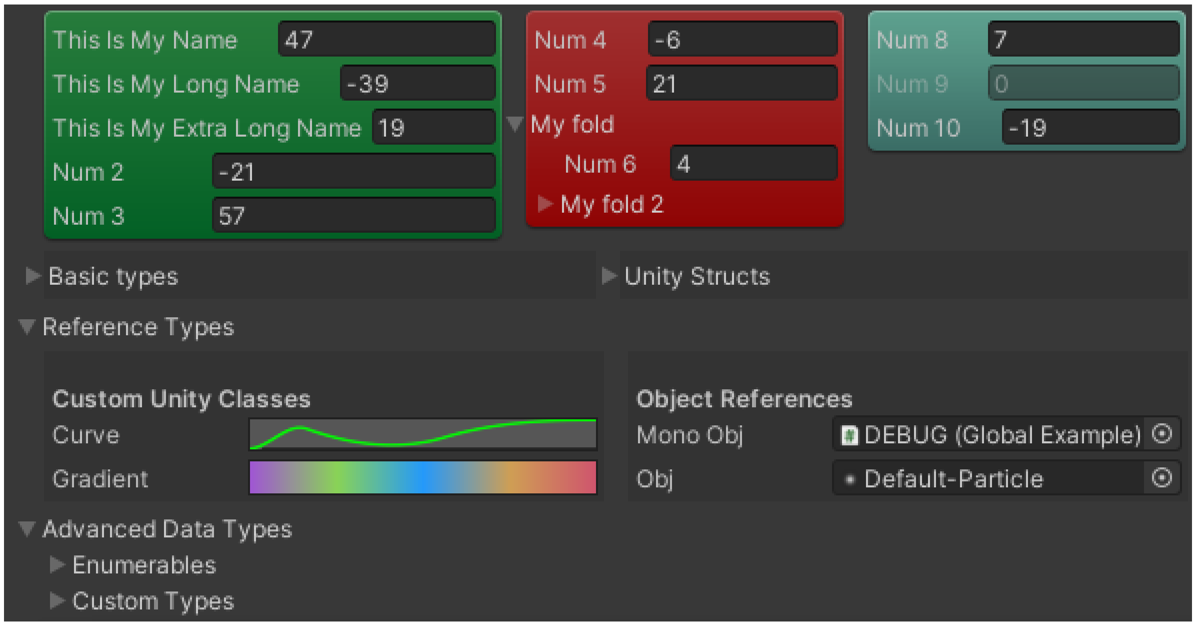
**Attack**
Attack Dmg 5
Crit Prop 0.15
Secondary Attack 8
Magic Attack Dmg 3

**Defense**
Base Defense 15
Magic Defense 8
Fire Defense 7



| **Health** | | **Attack** | | **Defense** | |
|---|---|---|---|---|---|
| Health | 17 | Attack Dmg | 5 | Base Defense | 15 |
| Max Health | 20 | Crit Prop | 0.15 | Magic Defens | 8 |
| | | Secondary Attack Dmg | 8 | Fire Defense | 7 |
| | | Magic Attack Dmg | 3 | | |

Simple columns layout



This Is My Name 47
This Is My Long Name -39
This Is My Extra Long Name 19
Num 2 -21
Num 3 57

Num 4 -6
Num 5 21
▼ My fold
Num 6 4
▶ My fold 2

Num 8 7
Num 9 0
Num 10 -19

▶ Basic types
▶ Unity Structs
▼ Reference Types

**Custom Unity Classes**
Curve
Gradient

**Object References**
Mono Obj DEBUG (Global Example)
Obj Default-Particle

▼ Advanced Data Types
▶ Enumerables
▶ Custom Types

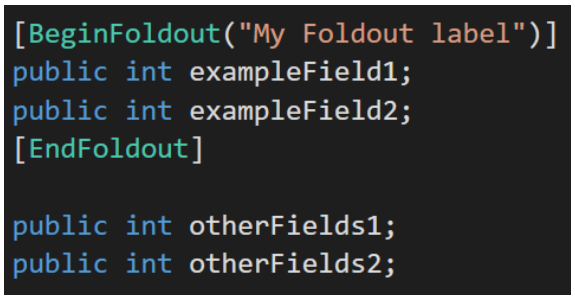
Advanced usage of the tool

## 1.1. List of Tools

### Layout Attributes

All layouting attributes follow the same principle of scopes, in which an attribute defines the beginning '[Begin...]' and a second attribute defines the end '[End...]'. If scopes are correctly defined, you can nest different types of layout attributes to create complex designs.

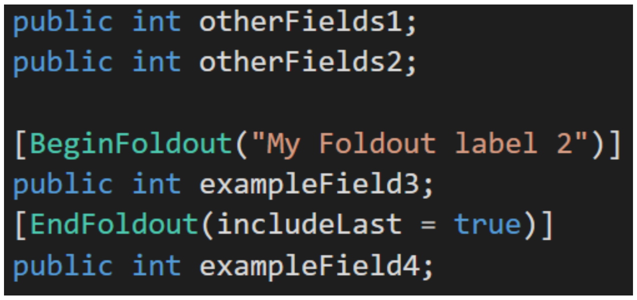
Attributes will always have to be on top of fields, so in case you want to include the last field of a script inside a scope, a flag is present inside every EndLayout attribute: '[End...(includeLast=true)]'. This flag decides whether or not to include the field it is sitting on in the scope.



```
[BeginFoldout("My Foldout label")]
public int exampleField1;
public int exampleField2;
[EndFoldout]

public int otherFields1;
public int otherFields2;
```

Normal layout attribute usage



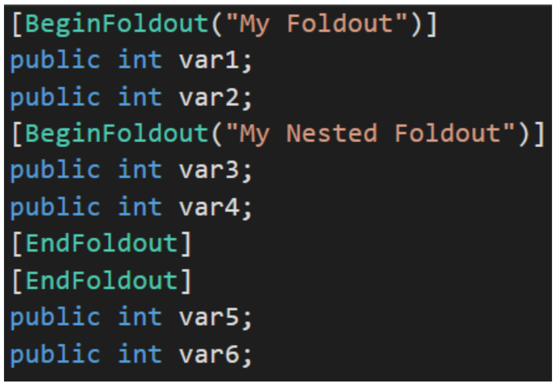
```
public int otherFields1;
public int otherFields2;

[BeginFoldout("My Foldout label 2")]
public int exampleField3;
[EndFoldout(includeLast = true)]
public int exampleField4;
```

Usage if field is last in the script

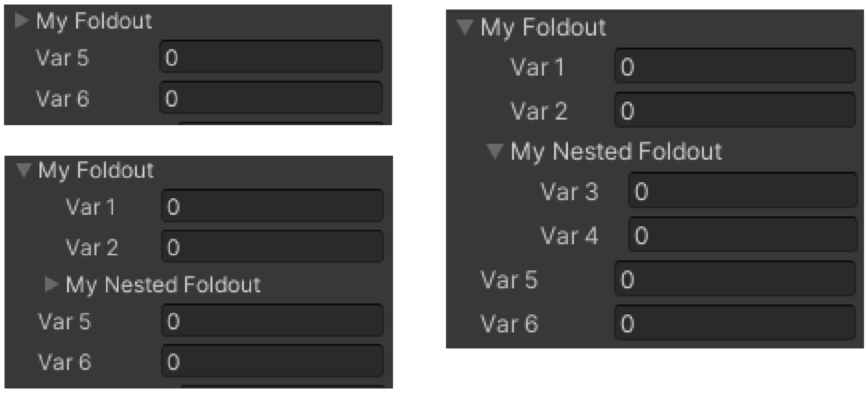
### Foldouts - Layout

Collapsable sections can come quite handy when you have a substantial number of fields in your inspector and do not require all of them to be visible at once. To define collapsible sections, use '[**BeginFoldout**(<label>)]' and '[**EndFoldout**]'.



```
[BeginFoldout("My Foldout")]
public int var1;
public int var2;
[BeginFoldout("My Nested Foldout")]
public int var3;
public int var4;
[EndFoldout]
[EndFoldout]
public int var5;
public int var6;
```

Code sample



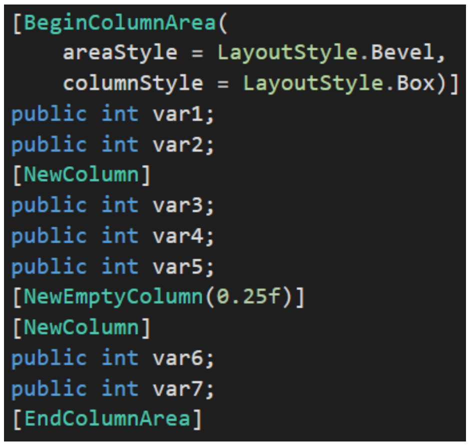
Expansion levels

## Columns - Layout

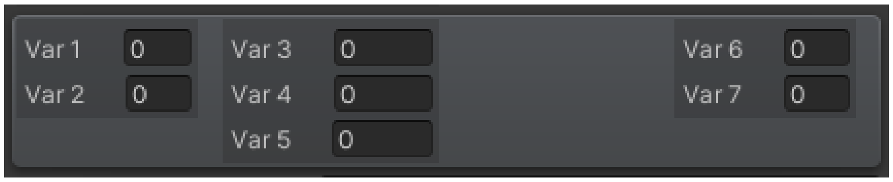
Arranging your fields in columns can provide a more organized workspace to group your fields into different categories. To define a column area, use '[**BeginColumnArea**]' and '[**EndColumnArea**]'. Every field inside this area will belong to a column. To define the beginning of a new column, use '[**NewColumn**]' inside your column area. To define empty columns as spacing, use '[**NewEmptyColumn**]'.

Columns can be configured with an automatic width (by default) or with a specific one using the property *'columnWidth'*, which defines the percentage of the inspector's width this column will be using. The columns in a column area that do not define a width will occupy the remaining available space.

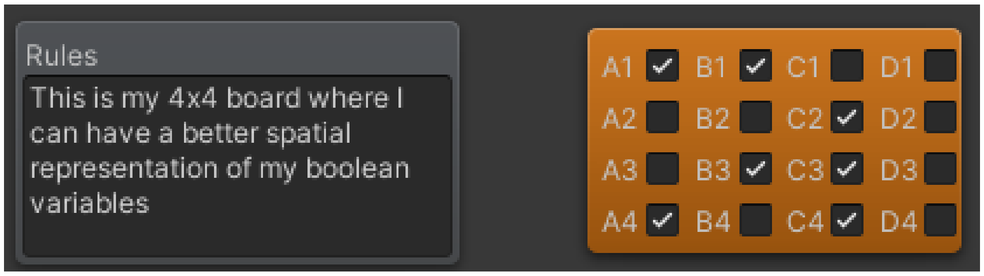
Columns can have a specific container style. These styles can be applied to columns or column areas via the properties *'columnStyle'* and *'areaStyle'* , respectively. A column style applied in the attribute '[BeginColumnArea]' will affect every column in its group. All available styles are listed in the enum **LayoutStyle**.



```
[BeginColumnArea(
    areaStyle = LayoutStyle.Bevel,
    columnStyle = LayoutStyle.Box)]
public int var1;
public int var2;
[NewColumn]
public int var3;
public int var4;
public int var5;
[NewEmptyColumn(0.25f)]
[NewColumn]
public int var6;
public int var7;
[EndColumnArea]
```



Basic usage of columns

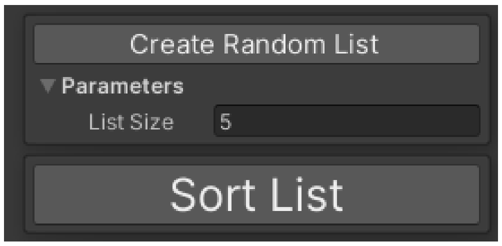

Rules
This is my 4x4 board where I can have a better spatial representation of my boolean variables
A1 ✓ B1 ✓ C1 D1
A2 B2 C2 ✓ D2
A3 B3 ✓ C3 ✓ D3
A4 ✓ B4 C4 ✓ D4

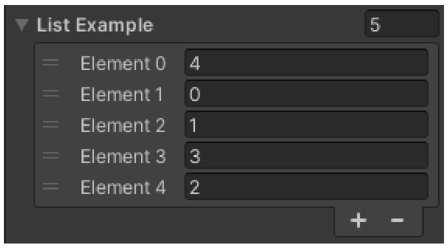
Advanced usage of columns

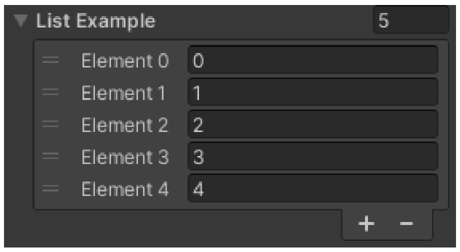
## Buttons

This is one of the key tools in this package and, without any doubt, the most powerful one. To create a button, simply add the attribute '[**Button**(<label>)]' on top of your instance method. You can also change the font size of the label with the property *fontSize*. Not all parameter types are supported at the moment. Make sure to follow the indications provided in the demo scene if this inconvenience occurs to you. Buttons can be useful to execute code without adding it to the main loop of the game or without using the attribute [ExecuteInEditMode] in your class. The demo scene provided in this package shows some examples:

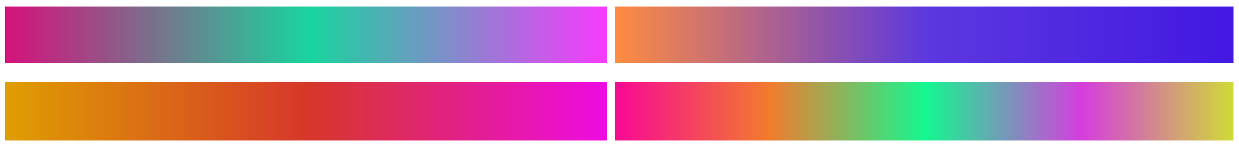
**Create a list of random numbers and sort it**
You can define methods to initialize your GameObjects on demand by following a set of parameters. In this example, the list size is a parameter of the button. An additional button is provided to sort the generated list.







**Generate a random gradient**
Another useful scenario is testing your methods. You can check the correct codification of your problem in an isolated manner before integrating it into the game. In this example, a method has been coded to generate a random gradient between 2 and 6 colors. Here are some execution results:

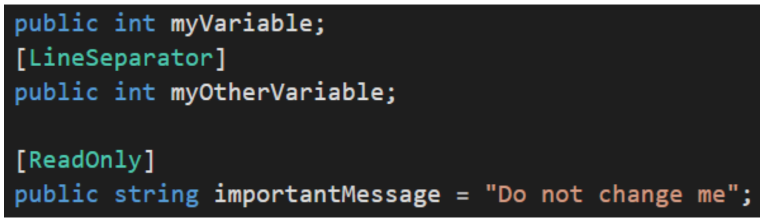


**Lerp through a list of colors**
Asynchronous methods are also supported. Be cautious with these methods, as there is no way to stop them once they have been started. This example modifies the value of a color variable by lerping through the colors provided in the color list parameter. To check the resulting effect, head to the demo scene and try it out!

### Others

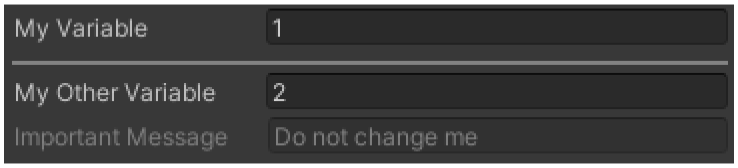
Instead of a simple [Space] to separate vertical sections, you can use the attribute '[**LineSeparator**]' to draw a line between two fields.

If you want to have a variable visible in the inspector but do not want it to be modified by hand, you can use the '[**ReadOnly**]' attribute.



```
public int myVariable;
[LineSeparator]
public int myOtherVariable;

[ReadOnly]
public string importantMessage = "Do not change me";
```



| | |
|---|---|
| My Variable | 1 |
| My Other Variable | 2 |
| Important Message | Do not change me |

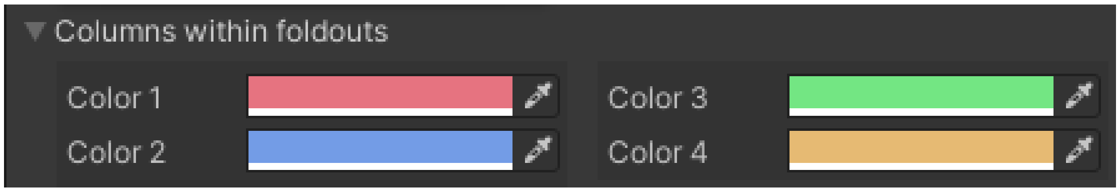
## 1.2. Combining attributes

You can combine every layout element with each other to define your own style. The following examples are present in the demo scene provided in the package. Head there to check how they were achieved.

**Columns inside foldouts**

You can hide entire column areas inside foldouts.

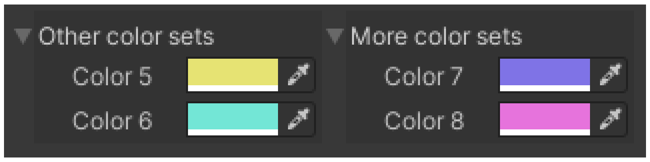
```
[BeginFoldout(<label>)]
[BeginColumnArea]
        <Column contents>
[NewColumn]
        <Column contents>
[EndColumnArea]
[EndFoldout]
```

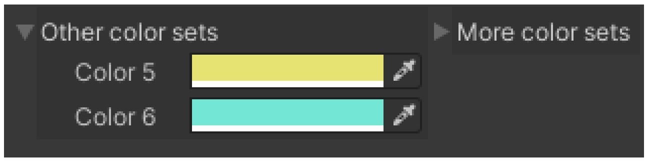


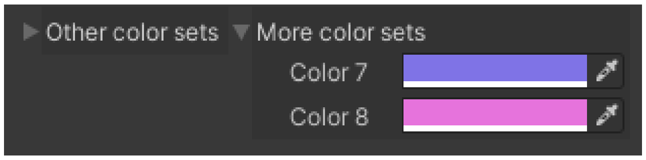
**Foldouts inside columns**

You can make columns collapsible by including all the contents of the column inside a foldout. This will be helpful as the width of the columns will be adjusted to the available space. If it is the only column expanded, its width will be much greater.

```
[BeginColumnArea]
        [BeginFoldout(<label>)]
        <Column contents>
        [EndFoldout]
[NewColumn]
        [BeginFoldout(<label>)]
        <Column contents>
        [EndFoldout]
[EndColumnArea]
```



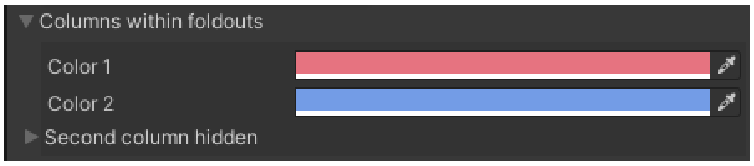




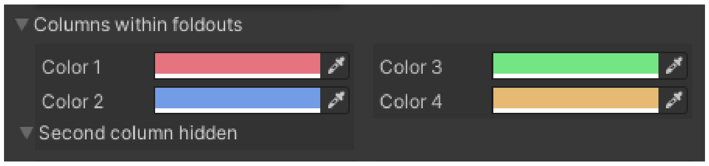
**Mixed nesting**

You can hide entire columns, which allows you to make column width more dynamic. This is done by defining the foldout before the new column is created. Make sure to close the foldout scope *before* the column area, or else the column area won't be correctly defined and the editor will paint garbage.

```
        [BeginColumnArea]
                <Column contents>
        [BeginFoldout(<label>)]
        [NewColumn]
                <Column contents>
        [EndFoldout]
        [EndColumnArea]
```



Second column collapsed



Second column expanded

## 1.3. QOL details

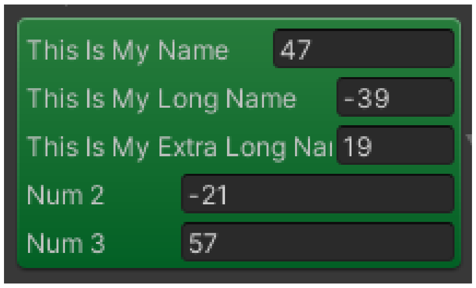
This section covers some details about some quality-of-life features that the tool incorporates and that might not be noticed at first glance. The list goes as follows:

The preferences of every attribute will be saved on domain reloads (when Unity recompiles scripts or when the application is launched). These preferences are saved per MonoBehaviour, so two GameObjects with the same script can have different saved data. These preferences include the status of your foldouts (expanded or collapsed) or the values set in your button parameters (hence the need for them to be serializable).

Every component has a special tracking system so that their values persist when you rename or reorder them (not both at the same time). This applies to:

- Foldouts: rename or relocate.
- Buttons: rename button, rename method, or reorder methods.
- Button parameters: rename or rearrange the parameters of a method.

A feature that Unity does not quite manage correctly is adjusting the width of labels. With the existence of columns, this issue became noticeable and has been fixed. Labels will occupy 35% of the width available, unless this space is not enough to display the whole label. In that case, the property field will shrink. When the property field becomes too small, the label will not be able to take up more space and will overflow.



Finally, in case you want to temporarily disable the tool, you can access the menu *Window > Advanced Editor Tools*, which will disable all the layout logic and the buttons. You can also reload the whole application from said menu, which will delete all the saved preferences for each MonoBehaviour, in case of a corruption of the saved data. If you find any errors, please report them to the email available in my profile on the Asset Store.

## 2. Limitations

This package currently has some limitations that might worsen the experience of creating your desired workspace. In future versions of the application, these problems will be tackled with high priority, as they are considered inconveniences.

Decorator drawers cannot be applied before the layout. This prevents the user from separating different column areas or foldouts, as the decorators will always be applied on top of the property fields. E.g., you cannot use [Space], [Header], or [LineSeparator] between two different column areas, as they will be applied inside the first column of the second column area.

Buttons are always painted at the bottom of the inspector. You cannot mix fields and buttons seamlessly or apply layouts to the buttons. E.g., you cannot hide all your buttons inside a foldout or arrange them in columns.

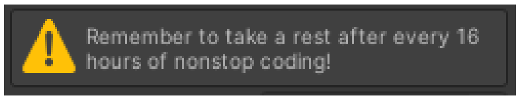
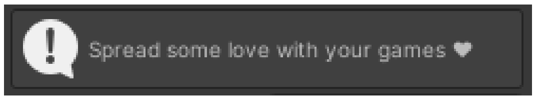
Buttons' parameters do not integrate undo and redo operations, and do not support class or struct types as parameters at the moment. These issues will be fixed in the following version with maximum priority.

## 3. Future Work

After fixing the existing limitations explained in Section 2, some additional features will be coming in future versions of this package. (Suggestions are also accepted!)

- The first and most essential one is to include serializable dictionaries without the need to write any custom inspector scripts.
- Buttons will accept references to script variables as input parameters instead of only constant values.
- It will be possible to display the fields of MonoBehavior and ScriptableObject properties instead of as object property fields.

**New Attributes**

- Custom Header (font size, bold, color, vertical spacing, etc.)
- Message field, to display constant messages to other members of the team who might want to use some tool you are developing.
- 
  Remember to take a rest after every 16 hours of nonstop coding!
  
  Spread some love with your games ♥
- Scrollbars, both horizontal and vertical I will check if this is comfortable to use or not before implementing it.
- Tabs layout. Choose which group of fields to display with the click of a button. Only one at a time!