CG FRAMEWORK – USER MANUAL

Content

System

ObjectTransform

Object Pooling Manager

- **Files:** CGFObjectPoolingManager.cs
- **Path:** "CGF/Systems/ObjectTransform"

Manager that allows to the associated gameobject to manage the object pooling.

Instantiate On Awake – Instance all objects on Awake.

Object To Pool – Pooleable objects list

- **Object To Pool** – Pooleable object.
- **Amount** – Total amount of instances.
- **Dynamic** – Allows to exceed the total amount of instances if needed.

Use

Add the CGFObjectPoolingManager to an empty gameobject in the scene and add all objects to pool to the poolable object list and configure their settings.

Call the method InstantiatePoolObject to instance a poolable object from the poolable object list.

```
CGFObjectPoolingManager.Instance.InstantiatePoolObject(elementToSpawn.prefab, spawnPo
sition, Quaternion.identity) as GameObject;
```

In case the object must be destroyed, it must be disabled to return to the pool.

```
gameObject.setActive(false);
```

Call the method AddObject to add more objects to the pool in run time.

```
AddObject(GameObject objectToAdd, int amount, bool dynamic)
```

Editor

Editor Utilities

- **Files:** CGFEditorUtilities.cs
- **Path:** "CGF/Editor"

Editor Utilities is a set of methods and utilities to build inspector scripts fast and easily.

Compatible property types:

- Int
  - Generic
  - Positive
  - Negative
  - Slider
- Float
  - Generic
  - Positive
  - Negative
  - Ceil
  - Floor
  - Rounded
  - Slider
  - Slider Ranged
- Long
  - Generic
  - Positive
  - Negative
- Double

    - Generic
    - Positive
    - Negative
    - Ceil
    - Floor
    - Rounded
- Boolean
- Vector 2
    - Generic
    - Positive
    - Negative
- Vector 3
    - Generic
    - Positive
    - Negative
- Vector 4
    - Generic
    - Positive
    - Negative
- Color
- Enumeration
- Rect
- Text
- Animation curve
- Object <T>
- Fold Out
- List
    - Values
    - Classes
- Tags (From Tags and Layers settings)
- Layers (From Tags and Layers settings)
- Sorting Layers (From Tags and Layers settings)
- Inputs (From Input manager settings)
- Layer Mask

Use

To create a custom inspector you should create an editor script with this attributes:

- [CustomEditor(typeof(SampleScript))] – Assign the type of the class that the editor script will represent in the inspector, in this case: SampleScript.cs.
- [CanEditMultipleObjects] – Allow support multi-object editing.
- Must inherit from “UnityEditor.Editor”.

Step 1

Create “SerializedProperty” variables in the editor script (SampleScriptEditor) that reference the variables of the target Class.

```
[CustomEditor(typeof(SampleScript))]
[CanEditMultipleObjects]
public class SampleScriptEditor : UnityEditor.Editor
{
    private SerializedProperty _int;

}
```

Step 2

Inside “OnEnable()” method to initialize the variables of “SerializedProperty” type. Adding as a parameter of “FindProperty(string)” method the name of the referenced variable in the target script.

```
[CustomEditor(typeof(SampleScript))]
[CanEditMultipleObjects]
public class SampleScriptEditor : UnityEditor.Editor
{
    private SerializedProperty _int;

    void OnEnable()
    {
         _int = serializedObject.FindProperty("integer");
    }
}
```

In this case, the variable “integer” from SampleScript it’s being linked to the “_int” variable.

Step 3

The method “OnInspectorGUI()” creates the new inspector, allows draw and place the properties in inspector panel. This method works like the “Update()” method, and it must be overrided to work correctly.

Accessing to the static class “EditorGUILayout” you can create any field in the inspector. In this case, the method “IntField()” will show our “_int” variable and will set “Integer” as its name.

```
[CustomEditor(typeof(SampleScript))]
[CanEditMultipleObjects]
public class SampleScriptEditor : UnityEditor.Editor
{
    private SerializedProperty _int;
```

```
    void OnEnable()
    {
         _int = serializedObject.FindProperty("integer");
    }

    public override void OnInspectorGUI()
    {
        _int = EditorGUILayout.IntField("Integer", _int);
    }
}
```

In the Editor Utilities you use the “CGFEditorUtilities.BuildInt()” method instead of “EditorGUILayout.IntField()” method. You can add a description as a method parameter to show a tooltip from inspector.

```
[CustomEditor(typeof(SampleScript))]
[CanEditMultipleObjects]
public class SampleScriptEditor : UnityEditor.Editor
{
    private SerializedProperty _int;

    void OnEnable()
    {
         _int = serializedObject.FindProperty("integer");
    }

    public override void OnInspectorGUI()
    {
        CGFEditorUtilities.BuildInt("Integer", "All integer values", _int);
    }
}
```

Step 4

In order to update the inspector fields and values you use the “Update()” method, and to save changes to the variables we must use “ApplyModifiedProperties()”.

```
[CustomEditor(typeof(SampleScript))]
[CanEditMultipleObjects]
public class SampleScriptEditor : UnityEditor.Editor
{
    private SerializedProperty _int;

    void OnEnable()
    {
```

```
        _int = serializedObject.FindProperty("integer");
    }

    public override void OnInspectorGUI()
    {
        serializedObject.Update();

        CGFEditorUtilities.BuildInt("Integer", "All integer values", _int);

        serializedObject.ApplyModifiedProperties();
    }
}
```

Reorderable Lists

Unity manages list information as arrays which is uncomfortable for user. To improve usability, lists have been created that allows sort list elements just dragging them and additional features have been added.

The method "BuildListCustom()" allows you to create reorderable lists of classes. With the "params string[] properties" parameter, assign all the class properties for each list element.

```
[CustomEditor(typeof(SampleScript))]
[CanEditMultipleObjects]
public class SampleScriptEditor : UnityEditor.Editor
{
    private SerializedProperty _listProperty;

    private ReorderableList _reorderableList;

    private int _newListSize;

    void OnEnable()
    {
        _listProperty = serializedObject.FindProperty("list");

        _reorderableList = new ReorderableList(serializedObject, _listProperty, true
, true, true, true);
    }

    public override void OnInspectorGUI()
    {
        serializedObject.Update();
```

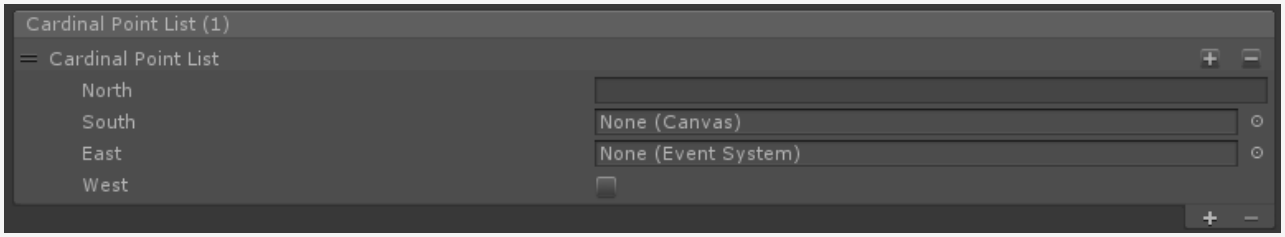
```
        _newListSize = CGFEditorUtilities.BuildListButtons(_listProperty, _newListSiz
e);

        _reorderableList = CGFEditorUtilities.BuildListCustom(_listProperty, _reorder
ableList, "List Name", bool vertical, int[] propertySpace, params string[] properties
);

        serializedObject.ApplyModifiedProperties();
    }
}
```

To make easier the add and remove elements actions from the list you have two buttons. The “BuildListButtons()” method adds these buttons.

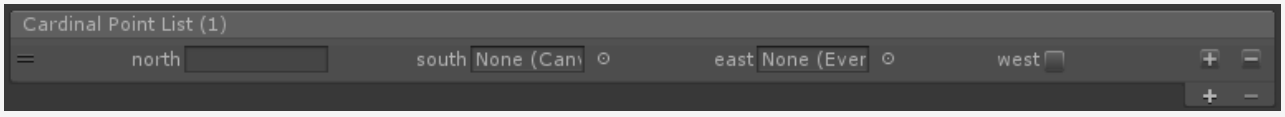
An additional button has been created that replaces the total number of elements in the list with a new total. This action is activated by matching the int value of the new quantity of elements to the “BuildListButtons()” method.

There are two reorderable list layouts, vertical (default) and horizontal. “bool vertical” parameter determine the layout to use.

In the vertical layout element fields appear one below the other.



In the horizontal layout element fields of elements appear next to each other.



You can specify width of each element field with “int[] propertySpace” parameter. You enter a value between 1 and 12, the total value of all the array elements must be 12.

Fold Out

Boolean value that manages if the objects inside the FoldOut must be shown or not.

As the FoldOut boolean value in the inspector is selected, it changes its value to show or not the objects that contains.

Use

```
[CustomEditor(typeof(SampleScript))]
[CanEditMultipleObjects]
public class SampleScriptEditor : UnityEditor.Editor
{
    private bool _foldOut;

    private SerializedProperty _vector3;

    public void OnEnable()
    {
         _vector3 = serializedObject.FindProperty("vector3");
    }

    public override void OnInspectorGUI()
    {
        serializedObject.Update();

        _foldOut = CGFEditorUtilities.BuildFoldOut("Cube5", "Descr
iption", _foldOut);

        if(_foldOut)
        {
               CGFEditorUtilities.BuildVector3("Position", "Descri
ption", _vector3);
        }

        serializedObject.ApplyModifiedProperties();
    }
}
```

Component Tools

Adds shortcuts to actions related to the component.

Documentation – Shortcut to component documentation.

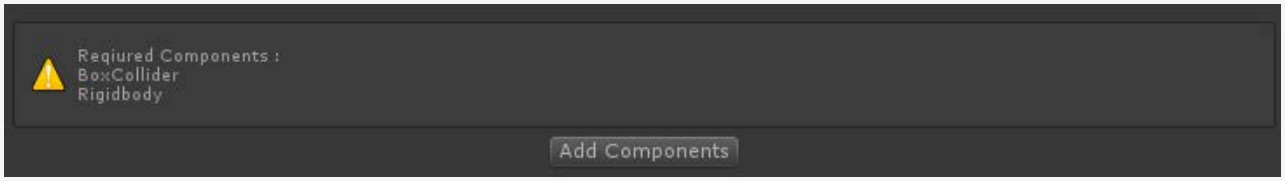
Remove – Shortcut to delete the component.

Use

```
CGFEditorUtilities.BuildComponentTools(string documentationURL, SerializedObject component)
```

Required Components

Warn what components are required gameobject and allows them to be added easily.



Add Components – Add the required components that are not present in the current gameobject.

Use

```
CGFEditorUtilities.BuildRequiredComponents(params Type[] components
```

```
CGFEditorUtilities.BuildRequiredComponents(typeof(BoxCollider), typeof(Rigidbody));
```

Manage Component Values

Copy, paste or paste as new component the values from a component.

Copied – Shows if copied values exists.

Copy – Copy the component values.

Paste – Paste the component values on the current component.

Paste as New – Add a new component to the gameobject with the copied values.

Use

```
CGFEditorUtilities.ManageComponentValues<T>()
```

Back Up Component Values

Allow create component back ups and restore this back ups saved in the project. Works in "Play" and "Edit" mode.

Backups – Dropdown menu with all available backups.

Backup – Create a back up of the component.

Restore – Replace the current component values for the values saved in the selected back up from dropdown menu.

Remove – Delete the selected backup from the dropdown menu.

Use

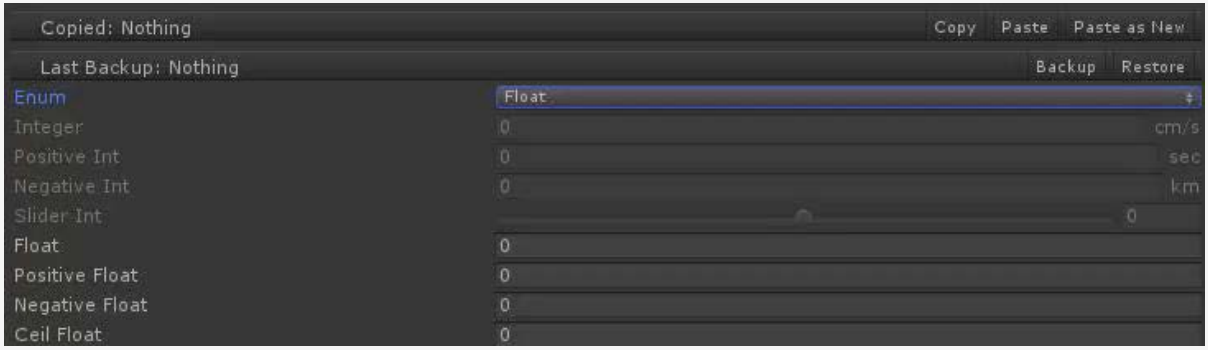
```
CGFEditorUtilities.BackUpComponentValues<SampleScript>(serializedObject)
```

The parameter of this method is the type <T> of the target scrip to back up and the "SerializedObject".

Locking properties

Enumeration

Allows enable or disable fields from inspector according to a selected enumeration value.



Every property construction method has an overload with two additional parameters to add the enumeration and the enumeration index of values that enable or disable the field.

Use

In this example each filed method receives an "_enum" variable as an enumeration parameter and a int value as other parameter that designates wich values from "_enum" enables the field.

```
public override void OnInspectorGUI()
{
     CGFEditorUtilities.BuildEnum("Enum", "Enumaration of different values", _enum);


     CGFEditorUtilities.BuildInt("Integer", "All integer values", _int, "cm/s", _enum
, 1);

     CGFEditorUtilities.BuildIntPositive("Positive Int", "Only positive integer value
s", _positiveInt, "sec", _enum, 1);


     CGFEditorUtilities.BuildFloat("Float", "All float values", _float, _enum, 2);

     CGFEditorUtilities.BuildFloatPositive("Positive Float", "Only positive float val
ues", _positiveFloat, _enum, 2);
}
```

Boolean

Allows enable or disable fields from the inspector according to a selected boolean property.

Every property construction method has an overload with an additional parameter to add a boolean that will lock/unlock the property in the inspector.

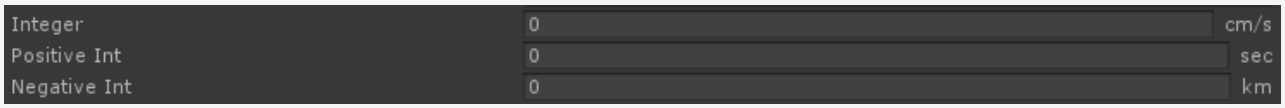
Use

In this example the property "_bool" will be the property locker, as we see in the code, every method adds as a parameter the the property "bool".

```
public override void OnInspectorGUI()
{
     CGFEditorUtilities.BuildBool("Boolean", "Locker Boolean", _bool);


     CGFEditorUtilities.BuildInt("Integer", "All integer values", _int, "cm/s", _bool
);

     CGFEditorUtilities.BuildIntPositive("Positive Int", "Only positive integer value
s", _positiveInt, "sec", _bool);


     CGFEditorUtilities.BuildFloat("Float", "All float values", _float, _bool);

     CGFEditorUtilities.BuildFloatPositive("Positive Float", "Only positive float val
ues", _positiveFloat, _bool);
}
```

Value Units

A unit can be added to the filds from inspector.



Every field construction method has an overload with an string parameter which will set the units.

Use

In this example you can see that the last parameter of each method is the units of each field.

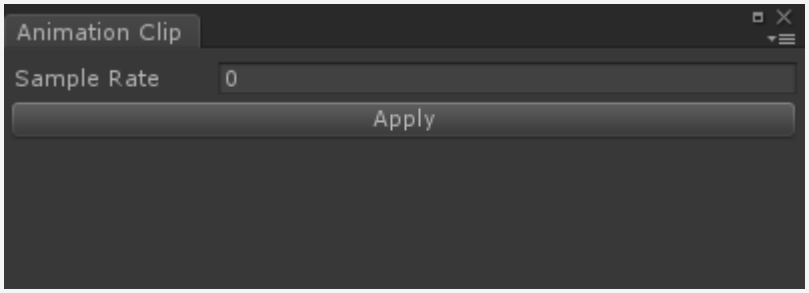
```
public override void OnInspectorGUI()
{
     CGFEditorUtilities.BuildInt("Integer", "All integer values", _int, "cm/s");

     CGFEditorUtilities.BuildIntPositive("Positive Int", "Only positive integer value
s", _positiveInt, "sec");

     CGFEditorUtilities.BuildIntNegative("Negative Int", "Only negative integer value
s", _negativeInt, "km");
}
```

Tools

Animation Clip Settings

- **Files:** CGFAnimationClipSettingsTool.cs
- **Path:** "CGF/Editor/Tools"
- **Menu:** "Window/Chloroplast Games Framework/Animation Clip Settings Tool"

Tool that allows change the sample rate from one or multiple animation clips.



- Sample Rate – New sample rate.

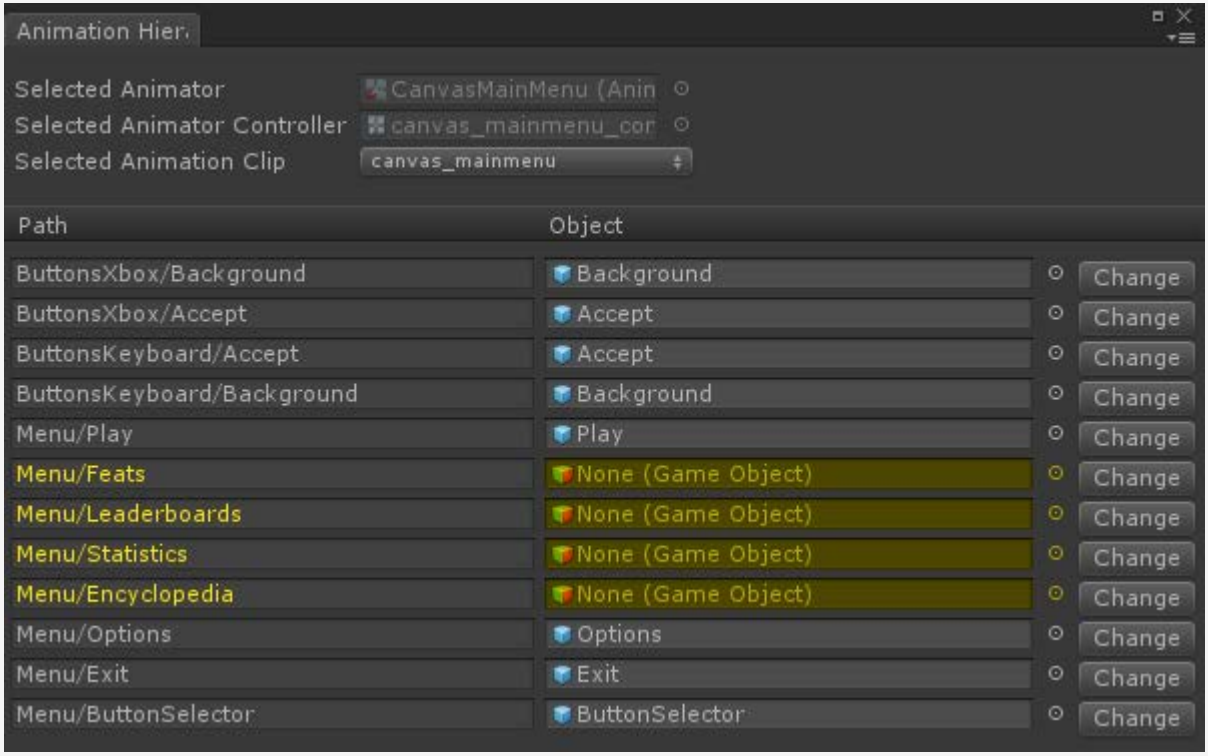
Use

Select a Animation Clip or multiple Animation Clips to change their settings.

Animation Hiearchy Editor

- **Files:** CGFAnimationHierarchyEditorTool.cs
- **Path:** "CGF/Editor/Tools"
- **Menu:** "Window/Chloroplast Games Framework/Animation Hierarchy Editor Tool"

Tool that allows change the path of hierarchy from a animation from an animation clip.



- Selected Animator – Selected Animator.
- Selected Animator Controller – Selected Animator Controller.
- Selected Animation Clip – Selected Animation Clip or Animation clips of the selected gameobject or Animator Controller.
- Path – Hierarchy path of the animated gameobject.
- Object – Animated gameobject or prefab.

Use

Select a gameobject or prefab with a Animator or Animator Controller, a Animator Controller or a Animation Clip.

Then you can change the hierarchy path of the animation changing the string (Path) and press the Change button or changing the animated gameobject or prefab (Object).

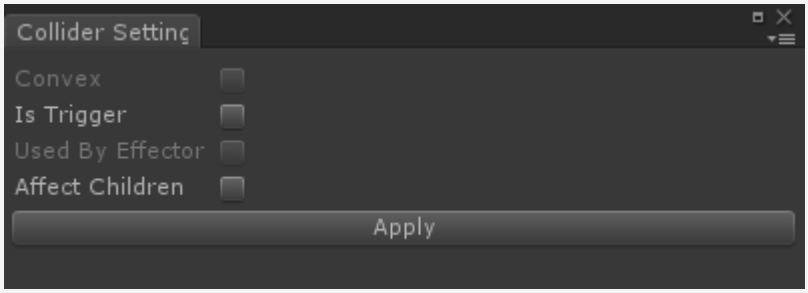
If the selected gameobject or Animator Controller have multiple Animation Clips you can switch the current Animation Clip using the Selected Animation Clip drop down menu.

When some property becomes yellow mens that the hierarchy path is incorrect because the referenced animated gameobject is missing.

Collider Settings

- **Files:** CGFColliderSettingsTool.cs
- **Path:** "CGF/Editor/Tools"
- **Menu:** "Window/Chloroplast Games Framework/Collider Settings Tool"

Tool that allows change the settings of a all colliders from a GameObject or prefab.



- Convex – Enable or disable the Convex property. Only for MeshCollider.
- Is Trigger – Enable or disable the Is Trigger property.
- Used By Effector – Enable or disable the Used By Effector property. Only for 2D Collider.
- Affect Children – Enable or disable the properties of the colliders of the childs from the selected gameobject.

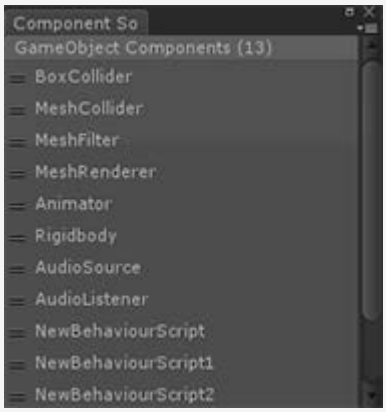
Use

Select a gameobject or prefab or multiple gameobjects and prefabs with a collider or multiple colliders of the same type to to change their collider properties.

Component Sorter

- **Files:** CGFComponentSorterTool.cs
- **Path:** "CGF/Editor/Tools"
- **Menu:** "Window/Chloroplast Games Framework/Component Sorter Tool"

Tool that allows sort the components from gameobject or prefab.



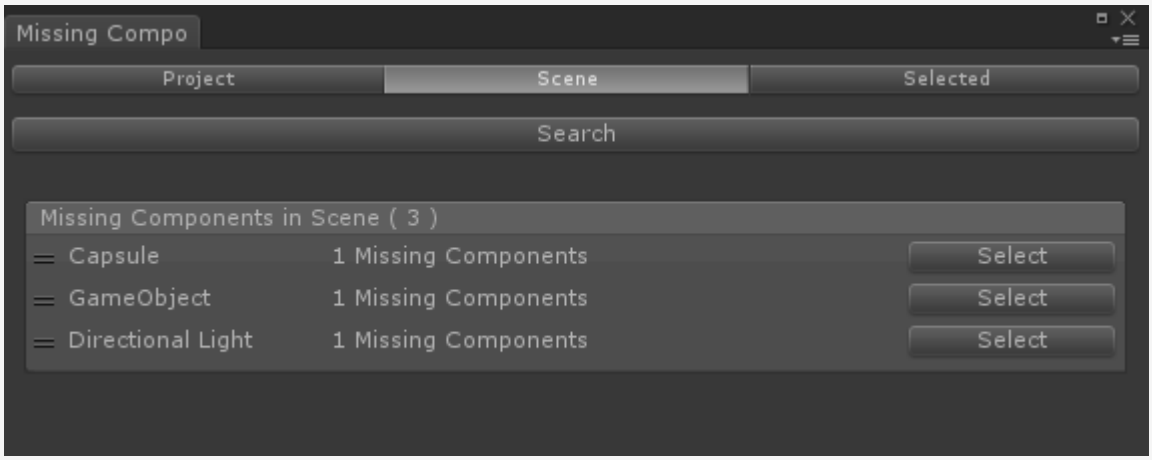
Use

Select a gameobject or prefab to sort (drag and drop) their components.

Missing Component Finder

- **Files:** CGFMissingComponentFinderTool.cs
- **Path:** "CGF/Editor/Tools"
- **Menu:** "Window/Chloroplast Games Framework/Missing Component Finder Tool"

Tool that allows search for missing components of gameobjects from a scene, project or selected gameobjects and folders.



Use

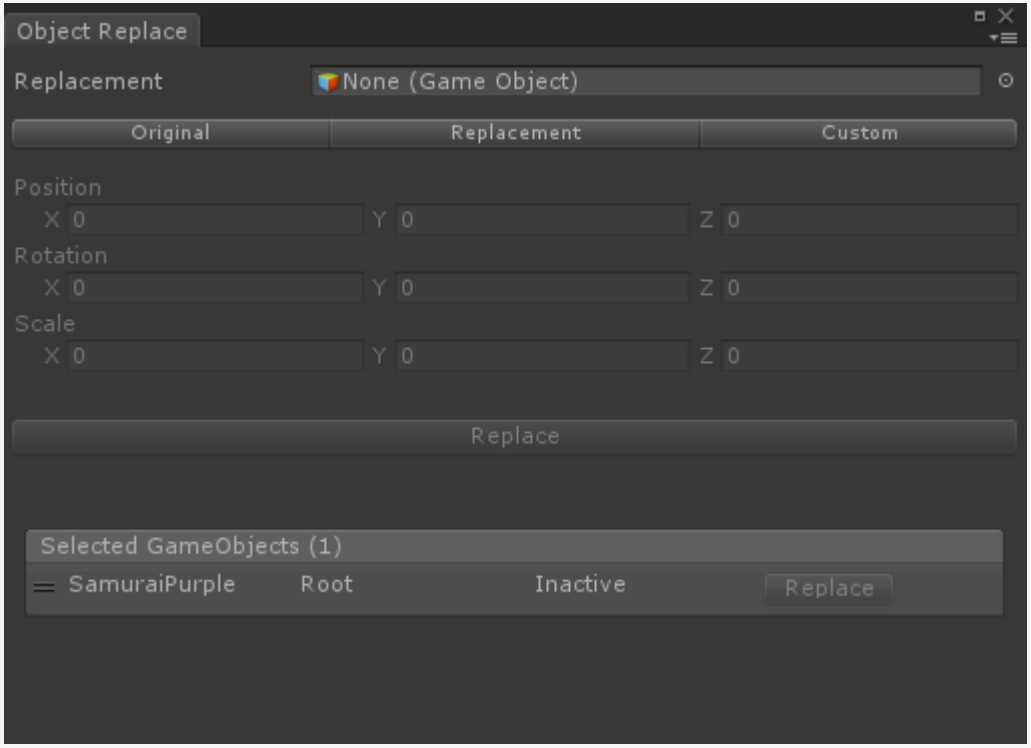
Three ways to search missing components from gameobjects.

- Project – Search missing components from prefabs and childrens inside project folder.
- Scene – Search missing components from gameobjects and childrens from the current scene.
- Selected – Search missing components from selected elements (gameobjects, prefabs or folders).

Object Replacer

- **Files:** CGFObjectReplacerTool.cs
- **Path:** "CGF/Editor/Tools"
- **Menu:** "Window/Chloroplast Games Framework/Object Replacer Tool"

Tool that allows replace a gameobject on the scene with other gameobject or a prefab.



- Replacement – Replacement gameobject or prefab that replaces the selected gameobject.
- Transform Replacement – Original, keep the transform values from original gameobject or prefab. Replacement, use the transform values from replacement gameobject or prefab. Custom, use the values from Position, Rotation and Scale properties.
- Position – New position. Only for Custom mode.
- Rotation – New rotatión. Only for Custom mode.
- Scale – New scale. Only for Custom mode.
- Selected GameObjects – Selected gameobject list.

Use

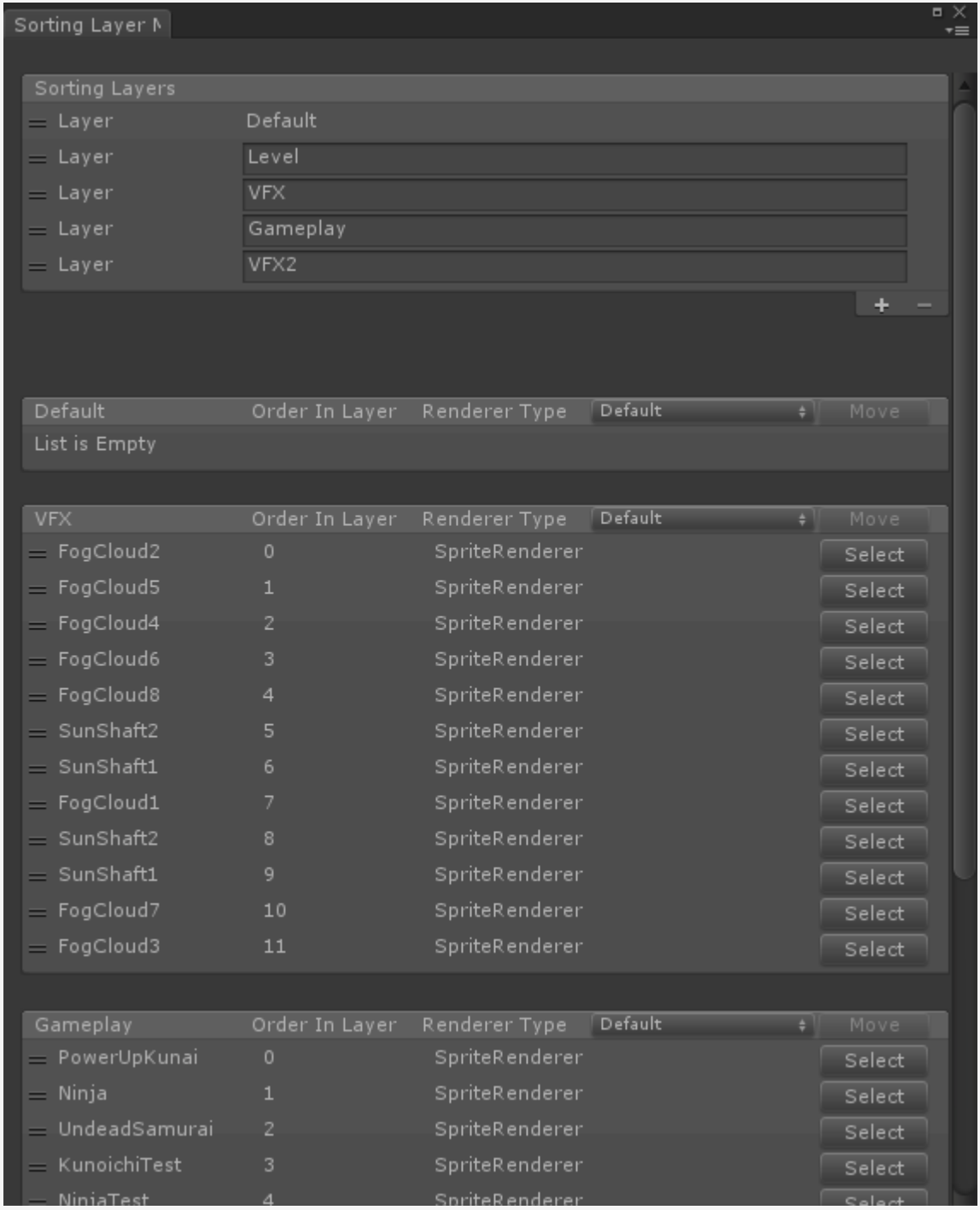
Drag a gameobject or prefaf into the Replacement property, configure the transform mode and press the Replace button.

You can replace each selected gameobject one by one.

Sorting Layer Manager

- **Files:** CGFSortingLayerManagerTool.cs
- **Path:** "CGF/Editor/Tools"
- **Menu:** "Window/Chloroplast Games Framework/Sorting Layer Manager Tool"

Tool that allows sort comfortably the sorting layers and order in layer of the elements from de current scene.



Use

With the top reorderable list you can sort (drag and drop), create and remove sorting layers.

With the bottom reorderable lists you can sort (drag and drop) the order in layer from each gameobject. Show all types of renderers (SpriteRenderer, ParticleSystem (ParticleRenderer), LineRenderer, TrailRenderer) less the MeshRenderer.

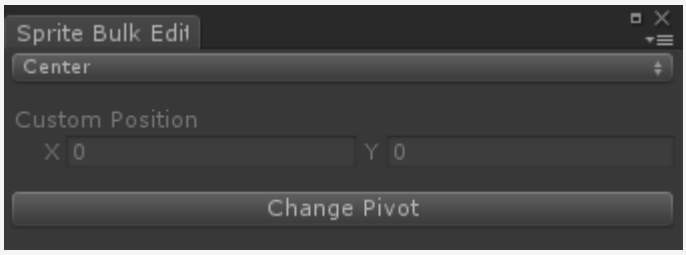
If you remove a sortign layers, all gameobjects from this sorting layer are moved to sorting layer “Default”.

Unity Sorting Layer Manager and Sorting Layer Manager Tool should not be used at the same time because it can cause errors.

Sprite Bulk Editor

- **Files:** CGFSpriteBulkEditorTool.cs
- **Path:** "CGF/Editor/Tools"
- **Menu:** "Window/Chloroplast Games Framework/Sprite Bulk Editor Tool"

Tool that allows modify the pivot of selected sprites.



- Dropdown menu – Position presets.
- Custom Position – Custom pivot position. Only if the selected option from the dropdown menu is Custom.

Use

Select a sprite or miltiple sprites.

If Sprite mode is set to Multiple, the changes affects all the sprite slices.