The Unity Layers Controller is a simple helper class that helps with the management of layers on the coding side. It allows management of layers by name instead of making your own data matrices to manage layers as well as return the state of layers.

How to use:

1) Simply add the Layers.cs and ListExtensions.cs scripts to your project to start using the helper class.
2) Reference the helper class using Layers. For example, *Layers.SetLayer(gameObject, "testLayer")*.
    a) Make sure all layers referenced in the helper class is added in Unity; the helper class cannot handle adding layers at runtime; only setting them.

The class is static so no reference is needed to operate.

Changing Culling Mask

ChangeCullingMask(string layerName, bool visible)

Changes the culling mask on the main camera by toggling the layer layerName. The boolean visible notates the state; if true, the layer will be enabled on the camera. If false, the layer will be disabled.

Overflow: ChangeCullingMask(string layerName, bool visible, Camera camera)

Overflow to ChangeCullingMask(). Designate a camera to adjust instead of the main camera.

Checking Layer state

IsLayerRendered(int layer)

Returns state of the given layer on the main camera. This function does not take layer names and only takes integers.

Overflow: IsLayerRendered(int layer, Camera camera)

Return state of the given layer on the given camera. This function does not take layer names and only takes integers.

Setting layer of objects

SetLayer(GameObject obj, int newLayer)

Manually sets the layer of the given object to the given integer layer. This does not affect children.

Overflow: SetLayer(GameObject obj, string newLayer)

Manually sets the layer of the given object to the given layer by name. This does not affect children.

SetLayerRecursively(GameObject obj, int newLayer)

Sets the layer of the given object and all of its children to the given integer layer.

Overflows:

SetLayerRecursively(GameObject obj, string newLayer)

Sets the layer of the given object and all of its children to the given layer name.

SetLayerRecursively(GameObject obj, int newLayer, string ignoreObjectName)

Sets the layer of the given object and its children to the given integer layer. This function will ignore any objects with the name specified. (Instantiated objects may not apply without the " (clone)" suffix).

SetLayerRecursively(GameObject obj, string newLayer, string ignoreObjectName)

Sets the layer of the given object and its children to the given layer name. This function will ignore any objects with the name specified. (Instantiated objects may not apply without the " (clone)" suffix).

SetLayerRecursively(GameObject obj, int newLayer, string[] ignoreObjectName)

Sets the layer of the given object and its children to the given integer layer. This function will ignore any objects with names in the given array.

SetLayerRecursively(GameObject obj, string newLayer, string[] ignoreObjectName)

Sets the layer of the given object and its children to the given layer name. This function will ignore any objects with names in the given array.

SetLayerRecursively(GameObject obj, int newLayer, List ignoreObjectName)

Sets the layer of the given object and its children to the given integer layer. This function will ignore any objects with names in the given list.

SetLayerRecursively(GameObject obj, string newLayer, List ignoreObjectName)

Sets the layer of the given object and its children to the given layer name. This function will ignore any objects with names in the given list.

ListExtensions.cs

There are some extra functions that may prove useful in the ListExtensions helper class. Also made static.

List<T>**.Move(int oldIndex, int newIndex)**

An extension to the List<> class. This extension moves an object at *oldIndex* to *newIndex*. This function will override what is in the new index if there's something already there.

List<T>**.Replace(int index, T item)**

An extension to the List<> class that replaces the item in the list at *index* with the new *item*. If there is nothing at that index, it will insert it automatically.

Boolean array[]**.Contains<T>(T item)**

An extension to arrays that functions like List<>.Contains(). This gives you the same ability of List<>.Contains() for arrays as you would lists. This function does iterate through the array.

- Returns *true* if the item given is in the list. Will only return true for exact matches, so data differences in structs will be treated as separate objects.
- Returns *false* if the item does not exist in the list, if there are differences between the given item and the items in the list and none match, or there was an error.

Credits:

Programmed by Michael Cox.

Github: https://github.com/Arylos07

Twitter: https://twitter.com/tsarylos/

Reddit: https://www.reddit.com/user/Arylos07

Project Github: https://github.com/Arylos07/Unity-Layers-Controller

License: https://github.com/Arylos07/Unity-Layers-Controller/blob/master/LICENSE.md