using System;

using System.Collections.Generic;

using UnityEngine;

#if UNITY\_EDITOR

using UnityEditor;

#endif

namespace UnityStandardAssets.Utility

{

public class AutoMobileShaderSwitch : MonoBehaviour

{

[SerializeField] private ReplacementList m\_ReplacementList;

// Use this for initialization

private void OnEnable()

{

#if UNITY\_IPHONE || UNITY\_ANDROID || UNITY\_WP8 || UNITY\_BLACKBERRY

var renderers = FindObjectsOfType<Renderer>();

Debug.Log (renderers.Length+" renderers");

var oldMaterials = new List<Material>();

var newMaterials = new List<Material>();

int materialsReplaced = 0;

int materialInstancesReplaced = 0;

foreach(ReplacementDefinition replacementDef in m\_ReplacementList.items)

{

foreach(var r in renderers)

{

Material[] modifiedMaterials = null;

for(int n=0; n<r.sharedMaterials.Length; ++n)

{

var material = r.sharedMaterials[n];

if (material.shader == replacementDef.original)

{

if (modifiedMaterials == null)

{

modifiedMaterials = r.materials;

}

if (!oldMaterials.Contains(material))

{

oldMaterials.Add(material);

Material newMaterial = (Material)Instantiate(material);

newMaterial.shader = replacementDef.replacement;

newMaterials.Add(newMaterial);

++materialsReplaced;

}

Debug.Log ("replacing "+r.gameObject.name+" renderer "+n+" with "+newMaterials[oldMaterials.IndexOf(material)].name);

modifiedMaterials[n] = newMaterials[oldMaterials.IndexOf(material)];

++materialInstancesReplaced;

}

}

if (modifiedMaterials != null)

{

r.materials = modifiedMaterials;

}

}

}

Debug.Log (materialInstancesReplaced+" material instances replaced");

Debug.Log (materialsReplaced+" materials replaced");

for(int n=0; n<oldMaterials.Count; ++n)

{

Debug.Log (oldMaterials[n].name+" ("+oldMaterials[n].shader.name+")"+" replaced with "+newMaterials[n].name+" ("+newMaterials[n].shader.name+")");

}

#endif

}

[Serializable]

public class ReplacementDefinition

{

public Shader original = null;

public Shader replacement = null;

}

[Serializable]

public class ReplacementList

{

public ReplacementDefinition[] items = new ReplacementDefinition[0];

}

}

}

namespace UnityStandardAssets.Utility.Inspector

{

#if UNITY\_EDITOR

[CustomPropertyDrawer(typeof (AutoMobileShaderSwitch.ReplacementList))]

public class ReplacementListDrawer : PropertyDrawer

{

const float k\_LineHeight = 18;

const float k\_Spacing = 4;

public override void OnGUI(Rect position, SerializedProperty property, GUIContent label)

{

EditorGUI.BeginProperty(position, label, property);

float x = position.x;

float y = position.y;

float inspectorWidth = position.width;

// Don't make child fields be indented

var indent = EditorGUI.indentLevel;

EditorGUI.indentLevel = 0;

var items = property.FindPropertyRelative("items");

var titles = new string[] {"Original", "Replacement", ""};

var props = new string[] {"original", "replacement", "-"};

var widths = new float[] {.45f, .45f, .1f};

const float lineHeight = 18;

bool changedLength = false;

if (items.arraySize > 0)

{

for (int i = -1; i < items.arraySize; ++i)

{

var item = items.GetArrayElementAtIndex(i);

float rowX = x;

for (int n = 0; n < props.Length; ++n)

{

float w = widths[n]\*inspectorWidth;

// Calculate rects

Rect rect = new Rect(rowX, y, w, lineHeight);

rowX += w;

if (i == -1)

{

// draw title labels

EditorGUI.LabelField(rect, titles[n]);

}

else

{

if (props[n] == "-" || props[n] == "^" || props[n] == "v")

{

if (GUI.Button(rect, props[n]))

{

switch (props[n])

{

case "-":

items.DeleteArrayElementAtIndex(i);

items.DeleteArrayElementAtIndex(i);

changedLength = true;

break;

case "v":

if (i > 0)

{

items.MoveArrayElement(i, i + 1);

}

break;

case "^":

if (i < items.arraySize - 1)

{

items.MoveArrayElement(i, i - 1);

}

break;

}

}

}

else

{

SerializedProperty prop = item.FindPropertyRelative(props[n]);

EditorGUI.PropertyField(rect, prop, GUIContent.none);

}

}

}

y += lineHeight + k\_Spacing;

if (changedLength)

{

break;

}

}

}

// add button

var addButtonRect = new Rect((x + position.width) - widths[widths.Length - 1]\*inspectorWidth, y,

widths[widths.Length - 1]\*inspectorWidth, lineHeight);

if (GUI.Button(addButtonRect, "+"))

{

items.InsertArrayElementAtIndex(items.arraySize);

}

y += lineHeight + k\_Spacing;

// Set indent back to what it was

EditorGUI.indentLevel = indent;

EditorGUI.EndProperty();

}

public override float GetPropertyHeight(SerializedProperty property, GUIContent label)

{

SerializedProperty items = property.FindPropertyRelative("items");

float lineAndSpace = k\_LineHeight + k\_Spacing;

return 40 + (items.arraySize\*lineAndSpace) + lineAndSpace;

}

}

#endif

}