using System;

using System.Collections;

using UnityEngine;

#if UNITY\_EDITOR

using UnityEditor;

#endif

namespace UnityStandardAssets.Utility

{

public class TimedObjectActivator : MonoBehaviour

{

public enum Action

{

Activate,

Deactivate,

Destroy,

ReloadLevel,

Call,

}

[Serializable]

public class Entry

{

public GameObject target;

public Action action;

public float delay;

}

[Serializable]

public class Entries

{

public Entry[] entries;

}

public Entries entries = new Entries();

private void Awake()

{

foreach (Entry entry in entries.entries)

{

switch (entry.action)

{

case Action.Activate:

StartCoroutine(Activate(entry));

break;

case Action.Deactivate:

StartCoroutine(Deactivate(entry));

break;

case Action.Destroy:

Destroy(entry.target, entry.delay);

break;

case Action.ReloadLevel:

StartCoroutine(ReloadLevel(entry));

break;

}

}

}

private IEnumerator Activate(Entry entry)

{

yield return new WaitForSeconds(entry.delay);

entry.target.SetActive(true);

}

private IEnumerator Deactivate(Entry entry)

{

yield return new WaitForSeconds(entry.delay);

entry.target.SetActive(false);

}

private IEnumerator ReloadLevel(Entry entry)

{

yield return new WaitForSeconds(entry.delay);

Application.LoadLevel(Application.loadedLevel);

}

}

}

namespace UnityStandardAssets.Utility.Inspector

{

#if UNITY\_EDITOR

[CustomPropertyDrawer(typeof (TimedObjectActivator.Entries))]

public class EntriesDrawer : PropertyDrawer

{

private const float k\_LineHeight = 18;

private 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 width = position.width;

// Draw label

EditorGUI.PrefixLabel(position, GUIUtility.GetControlID(FocusType.Passive), label);

// Don't make child fields be indented

var indent = EditorGUI.indentLevel;

EditorGUI.indentLevel = 0;

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

if (entries.arraySize > 0)

{

float actionWidth = .25f\*width;

float targetWidth = .6f\*width;

float delayWidth = .1f\*width;

float buttonWidth = .05f\*width;

for (int i = 0; i < entries.arraySize; ++i)

{

y += k\_LineHeight + k\_Spacing;

var entry = entries.GetArrayElementAtIndex(i);

float rowX = x;

// Calculate rects

Rect actionRect = new Rect(rowX, y, actionWidth, k\_LineHeight);

rowX += actionWidth;

Rect targetRect = new Rect(rowX, y, targetWidth, k\_LineHeight);

rowX += targetWidth;

Rect delayRect = new Rect(rowX, y, delayWidth, k\_LineHeight);

rowX += delayWidth;

Rect buttonRect = new Rect(rowX, y, buttonWidth, k\_LineHeight);

rowX += buttonWidth;

// Draw fields - passs GUIContent.none to each so they are drawn without labels

if (entry.FindPropertyRelative("action").enumValueIndex !=

(int) TimedObjectActivator.Action.ReloadLevel)

{

EditorGUI.PropertyField(actionRect, entry.FindPropertyRelative("action"), GUIContent.none);

EditorGUI.PropertyField(targetRect, entry.FindPropertyRelative("target"), GUIContent.none);

}

else

{

actionRect.width = actionRect.width + targetRect.width;

EditorGUI.PropertyField(actionRect, entry.FindPropertyRelative("action"), GUIContent.none);

}

EditorGUI.PropertyField(delayRect, entry.FindPropertyRelative("delay"), GUIContent.none);

if (GUI.Button(buttonRect, "-"))

{

entries.DeleteArrayElementAtIndex(i);

break;

}

}

}

// add & sort buttons

y += k\_LineHeight + k\_Spacing;

var addButtonRect = new Rect(position.x + position.width - 120, y, 60, k\_LineHeight);

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

{

entries.InsertArrayElementAtIndex(entries.arraySize);

}

var sortButtonRect = new Rect(position.x + position.width - 60, y, 60, k\_LineHeight);

if (GUI.Button(sortButtonRect, "Sort"))

{

bool changed = true;

while (entries.arraySize > 1 && changed)

{

changed = false;

for (int i = 0; i < entries.arraySize - 1; ++i)

{

var e1 = entries.GetArrayElementAtIndex(i);

var e2 = entries.GetArrayElementAtIndex(i + 1);

if (e1.FindPropertyRelative("delay").floatValue > e2.FindPropertyRelative("delay").floatValue)

{

entries.MoveArrayElement(i + 1, i);

changed = true;

break;

}

}

}

}

// Set indent back to what it was

EditorGUI.indentLevel = indent;

//

EditorGUI.EndProperty();

}

public override float GetPropertyHeight(SerializedProperty property, GUIContent label)

{

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

float lineAndSpace = k\_LineHeight + k\_Spacing;

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

}

}

#endif

}