```
using System;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

/// <summary>
/// IOAssigner is enabled in the editor scene after a custom circuit is validated to assign input/output orders and labels.
/// </summary>
public class IOAssigner : MonoBehaviour
{
    // Singleton state reference
    private static IOAssigner instance;

    /// <summary>
    /// The type of text <seealso cref="hoverText"/> should display.<br/><br/>
    /// <seealso cref="NONE"/>: prompts the user to hover on valid inputs/outputs.<br/>
    /// <seealso cref="INPUT"/>: currently hovered onto an input; display its potential order.<br/>
    /// <seealso cref="OUTPUT"/>: currently hovered onto an output; display its potential order.
    /// </summary>
    private enum TextMode { NONE, INPUT, OUTPUT }

    /// <summary>
    /// Exits the IOAssigner phase; the circuit must be validated again to reach this point.
    /// </summary>
    [SerializeField]
    KeyCode exitKey;

    /// <summary>
    /// The material applied to all empty inputs that are active and not being hovered on.
    /// </summary>
    [SerializeField]
    Material emptyInputMaterial;

    /// <summary>
    /// The material applied to all empty outputs that are active and not being hovered on.
    /// </summary>
    [SerializeField]
    Material emptyOutputMaterial;

    /// <summary>
    /// The material applied to an empty input/output that is currently
```

```
        hovered on.
42      /// </summary>
43      [SerializeField]
44      Material hoveredMaterial;
45
46      /// <summary>
47      /// Displays text determined by the current <seealso cref="TextMode"/
          >.
48      /// </summary>
49      [SerializeField]
50      TextMeshProUGUI hoverText;
51
52      /// <summary>
53      /// Whether the user has clicked on an empty input/output to bring
          about the label composition UI.
54      /// </summary>
55      private bool labelSelectionMode;
56
57      /// <summary>
58      /// Bypasses the movement system; prevents movement when composing a
          label.
59      /// </summary>
60      private bool movementOverride;
61
62      /// <summary>
63      /// The current selected input.
64      /// </summary>
65      private Circuit.Input currentInput;
66
67      /// <summary>
68      /// The list of empty inputs given to IOAssigner by <see
          cref="PreviewStructureManager"/>.<br/><br/>
69      /// These inputs are guaranteed to be valid, and the role of
          IOAssigner is to have the user label and order them to their liking.
70      /// </summary>
71      private List<Circuit.Input> emptyInputs;
72
73      /// <summary>
74      /// Contains all elements from <seealso cref="emptyInputs"/>, now
          reordered by the user.<br/><br/>
75      /// </summary>
76      private List<Circuit.Input> orderedInputs;
77
78      /// <summary>
79      /// The current selected output.
80      /// </summary>
81      private Circuit.Output currentOutput;
82
83      /// <summary>
```

```
84      /// The list of empty outputs given to IOAssigner by <see
          cref="PreviewStructureManager"/>.<br/><br/>
85      /// These outputs are guaranteed to be valid, and the role of
          IOAssigner is to have the user label and order them to their liking.
86      /// </summary>
87      private List<Circuit.Output> emptyOutputs;
88
89      /// <summary>
90      /// Contains all elements from <seealso cref="emptyOutputs"/>, now
          reordered by the user.<br/><br/>
91      /// </summary>
92      private List<Circuit.Output> orderedOutputs;
93
94      /// <summary>
95      /// The current valid GameObject hovered on by the user.<br/><br/>
96      /// This GameObject is guaranteed to either contain an <see
          cref="Circuit.Input"/> in <seealso cref="emptyInputs"/> or an <see
          cref="Circuit.Output"/> in <seealso cref="emptyOutputs"/>.
97      /// </summary>
98      private GameObject currentHover;
99
100     /// <summary>
101     /// Number of inputs and outputs have been successfully assigned and
          labeled by the user.
102     /// </summary>
103     private int inputCount, outputCount;
104
105     /// <summary>
106     /// The number of inputs and outputs that should be assigned for
          IOAssigner to complete its task.<br/><br/>
107     /// Both values initially are equal to their respective lengths of
          <seealso cref="emptyInputs"/> and <seealso cref="emptyOutputs"/>,
          but both lists have user-assigned elements removed.
108     /// </summary>
109     private int targetInputCount, targetOutputCount;
110
111     /// <summary>
112     /// Contain the respective labels for each user-assigned input and
          output.
113     /// </summary>
114     private List<string> inputLabels, outputLabels;
115
116     // Enforces a singleton state pattern and disables update calls.
117     private void Awake()
118     {
119         if (instance != null)
120         {
121             Destroy(this);
122             throw new Exception("IOAssigner instance already established;
```

```
                terminating.");
123         }
124
125         instance = this;
126         enabled = false; // After completing instance assignment, disable
              update calls until script is needed.
127     }
128
129     private void Update()
130     {
131         // Currently composing a label for an empty input or output.
132         if (labelSelectionMode)
133         {
134             // Exit conditions for quitting the labeling interface.
135             if (Input.GetKeyDown(exitKey) || Input.GetMouseButtonDown(1))
                  { CancelIOPress(); }
136
137             return;
138         }
139
140         // Exit conditions for quitting IOAssigner.
141         if (Input.GetKeyDown(exitKey) || Input.GetMouseButtonDown(1))
              { Exit(); return; }
142
143         Ray ray = CameraMovement.Instance.PlayerCamera.ScreenPointToRay
              (Input.mousePosition);
144
145         // Checks to see if the raycast hit ANY input or output
146         if (Physics.Raycast(ray, out RaycastHit hitInfo) &&
              (hitInfo.transform.gameObject.layer == 9 ||
              hitInfo.transform.gameObject.layer == 10))
147         {
148             Circuit.Input input = null; Circuit.Output output = null;
149             GameObject hitObject = hitInfo.transform.gameObject;
150
151             // Is an input
152             if (hitObject.layer == 9)
153             {
154                 input =
                      hitObject.GetComponent<CircuitVisualizer.InputReference>
                      ().Input;
155
156                 // If not within emptyInputs, not valid.
157                 if (!emptyInputs.Contains(input)) return;
158             }
159
160             // Is an output
161             else
162             {
```

```
163                 output =
                      hitObject.GetComponent<CircuitVisualizer.OutputReference
                      >().Output;
164
165                 // If not within emptyOutputs, not valid.
166                 if (!emptyOutputs.Contains(output)) return;
167             }
168
169             // Updates the interface and input/output material if the
                  current hit object is not the hovered object from the last
                  frame.
170             if (currentHover != hitObject)
171             {
172                 // If the last hit object was something, restore its
                      default material.
173                 if (currentHover != null)
                      currentHover.GetComponent<MeshRenderer>().material =
                      currentHover.layer == 9 ? emptyInputMaterial :
                      emptyOutputMaterial;
174
175                 // Update material and set text based on GameObject layer
                      (i.e. input or output)
176                 hitObject.GetComponent<MeshRenderer>().material =
                      hoveredMaterial;
177                 SetHoverText(hitObject.layer == 9 ? TextMode.INPUT :
                      TextMode.OUTPUT);
178
179                 // Store as current hover object
180                 currentHover = hitObject;
181             }
182
183             // Also begins the labeling process if LMB is pressed
184             if (Input.GetMouseButtonDown(0)) OnIOPress(input, output);
185         }
186
187         // Restores to default values if the raycast was unsuccessful.
188         else if (currentHover != null)
189         {
190             currentHover.GetComponent<MeshRenderer>().material =
                  currentHover.layer == 9 ? emptyInputMaterial :
                  emptyOutputMaterial;
191             currentHover = null;
192             SetHoverText(TextMode.NONE);
193         }
194     }
195
196     /// <summary>
197     /// Enables IOAssigner and starts the labeling/ordering process.
198     /// </summary>
```

```
199     /// <param name="emptyInputs">The empty inputs of the prospective
          custom circuit.</param>
200     /// <param name="emptyOutputs">The empty outputs of the prospective
          custom circuit.</param>
201     public void Initialize(List<Circuit.Input> emptyInputs,
          List<Circuit.Output> emptyOutputs)
202     {
203         labelSelectionMode = false; movementOverride = true;
204         inputCount = outputCount = 0;
205         targetInputCount = emptyInputs.Count; targetOutputCount =
              emptyOutputs.Count;
206         this.emptyInputs = emptyInputs; this.emptyOutputs = emptyOutputs;
207         orderedInputs = new List<Circuit.Input>(); orderedOutputs = new
              List<Circuit.Output>();
208         inputLabels = new List<string>(); outputLabels = new List<string>
              ();
209         hoverText.gameObject.SetActive(true);
210
211         // Switches the materials of all empty inputs and outputs to
              highlight them.
212         foreach (Circuit.Input input in emptyInputs)
              input.Transform.GetComponent<MeshRenderer>().material =
              emptyInputMaterial;
213
214         foreach (Circuit.Output output in emptyOutputs)
              output.Transform.GetComponent<MeshRenderer>().material =
              emptyOutputMaterial;
215
216         SetHoverText(TextMode.NONE);
217         Update();
218         enabled = true; // Enables frame-by-frame update calls from Unity.
219     }
220
221     /// <summary>
222     /// Disables IOAssigner and exists the labeling/ordering process.
223     /// </summary>
224     private void Exit()
225     {
226         movementOverride = false;
227
228         // If there are any empty inputs and/or outputs left, restore
              their default material.
229         foreach (Circuit.Input input in emptyInputs)
              input.Transform.GetComponent<MeshRenderer>().material =
              CircuitVisualizer.Instance.InputMaterial;
230
231         foreach (Circuit.Output output in emptyOutputs)
              output.Transform.GetComponent<MeshRenderer>().material =
              CircuitVisualizer.Instance.OutputMaterial;
```

```
232
233         hoverText.gameObject.SetActive(false);
234         enabled = false; // Disables frame-by-frame update calls from Unity.
235         TaskbarManager.Instance.CloseMenu();
236     }
237
238     /// <summary>
239     /// Cancels the label selection process for the current input or output.
240     /// </summary>
241     public void CancelIOPress()
242     {
243         movementOverride = true; labelSelectionMode = false;
244         TaskbarManager.Instance.CloseMenu();
245         TaskbarManager.Instance.NullState();
246     }
247
248     /// <summary>
249     /// Begins the label selection process for the current input or output.<br/><br/>
250     /// While there is both an input and output parameter, one of them will always be null.
251     /// </summary>
252     /// <param name="input">The input to label.</param>
253     /// <param name="output">The output to label.</param>
254     private void OnIOPress(Circuit.Input input, Circuit.Output output)
255     {
256         movementOverride = false; labelSelectionMode = true;
257         currentInput = input; currentOutput = output;
258         TaskbarManager.Instance.CloseMenu();
259         TaskbarManager.Instance.OpenLabelMenu(input != null);
260     }
261
262     /// <summary>
263     /// Successfully completes the label selection process for the current input or output.
264     /// </summary>
265     /// <param name="inputField">The text box to extract the label name from.</param>
266     public void ConfirmIOPress(TMP_InputField inputField)
267     {
268         string labelName = inputField.text;
269
270         // Implies the current labeling was done for an input
271         if (currentInput != null)
272         {
273             // Moves the current input to the ordered list and out of the empty list.
```

```
274             inputCount++;
275             emptyInputs.Remove(currentInput);
276             orderedInputs.Add(currentInput);
277             inputLabels.Add(labelName);
278         }
279 
280         // Implies the current labeling was done for an output
281         else
282         {
283             // Moves the current output to the ordered list and out of the empty list.
284             outputCount++;
285             emptyOutputs.Remove(currentOutput);
286             orderedOutputs.Add(currentOutput);
287             outputLabels.Add(labelName);
288         }
289 
290         inputField.text = "";
291         currentHover.GetComponent<MeshRenderer>().material = currentHover.layer == 9 ? CircuitVisualizer.Instance.InputMaterial : CircuitVisualizer.Instance.OutputMaterial;
292         currentHover = null;
293         SetHoverText(TextMode.NONE);
294         labelSelectionMode = false;
295         TaskbarManager.Instance.CloseMenu();
296         TaskbarManager.Instance.NullState();
297 
298         // All inputs/outputs have been labeled and/or ordered
299         if (inputCount == targetInputCount && outputCount == targetOutputCount)
300         {
301             hoverText.gameObject.SetActive(false);
302             enabled = false;
303             PreviewStructureManager.Instance.CreateCustomCircuit(orderedInputs, orderedOutputs, inputLabels, outputLabels); // Finally creates the custom circuit.
304         }
305 
306         else movementOverride = true;
307     }
308 
309     /// <summary>
310     /// Modifies the value of <seealso cref="hoverText"/> based on the assigned text mode.
311     /// </summary>
312     /// <param name="textMode">The current text mode.</param>
313     private void SetHoverText(TextMode textMode)
314     {
```

```
315         // Default text
316         string text = "hover over and select inputs/outputs to determine
              their order & label";
317
318         // Modifies the text if an input or output is implied.
319         switch (textMode)
320         {
321             case TextMode.INPUT:
322                 text = "input #" + (inputCount + 1);
323                 break;
324             case TextMode.OUTPUT:
325                 text = "output #" + (outputCount + 1);
326                 break;
327         }
328
329         hoverText.text = text;
330     }
331
332     // Getter methods
333     public static IOAssigner Instance { get { return instance; } }
334
335     public bool MovementOverride { get { return movementOverride; } }
336 }
```