

```
1  using System.Collections.Generic;
2  using UnityEngine;
3
4  /// <summary>
5  /// Logical representation of an AND gate.
6  /// </summary>
7  public class AndGate : Circuit
8  {
9      public AndGate() : this(Vector2.zero) { }
10
11     public AndGate(Vector2 startingPos) : base("AND", 2, 1, startingPos)
12         { }
13
14     /// <summary>
15     /// Returns an output to update if the output has changed due to
16     /// alterations in input power statuses.
17     /// </summary>
18     /// <returns>The list of outputs that should have their connections
19     /// called.</returns>
20     protected override List<Output> UpdateOutputs()
21     {
22         bool outputStatus = Outputs[0].Powered;
23         List<Output> outputs = new List<Output>();
24
25         // AND gate representation
26         Outputs[0].Powered = Inputs[0].Powered && Inputs[1].Powered;
27
28         if (outputStatus != Outputs[0].Powered || MaterialNotMatching())
29             outputs.Add(Outputs[0]);
30
31         return outputs;
32     }
33
34     /// <summary>
35     /// Checks all outputs to determine if the output node material is not
36     /// matching its power status.<br/><br/>
37     /// This is utilized within custom circuits to force update calls that
38     /// would normally not occur due to the nature of UpdateOutputs().
39     /// </summary>
40     /// <returns>Whether any output material does not match its power
41     /// status.</returns>
42     private bool MaterialNotMatching()
43     {
44         if (Outputs[0].StatusRenderer == null) return false;
45
46         return (Outputs[0].Powered && Outputs
47             [0].StatusRenderer.sharedMaterial !=
48             CircuitVisualizer.Instance.PowerOnMaterial) ||
49             (!Outputs[0].Powered && Outputs
```

...y Project\Assets\Scripts\Circuits\Starting\AndGate.cs

```
[0].StatusRenderer.sharedMaterial !=  
CircuitVisualizer.Instance.PowerOffMaterial);  
41     }  
42 }
```

2



```
1  using System.Collections.Generic;
2  using UnityEngine;
3
4  /// <summary>
5  /// Logical representation of a BUFFER gate.
6  /// </summary>
7  public class Buffer : Circuit
8  {
9      public Buffer() : this(Vector2.zero) { }
10
11     public Buffer(Vector2 startingPos) : base("BUFFER", 1, 1, startingPos)    ↵
12         { }
13
14     /// <summary>
15     /// Returns an output to update if the output has changed due to    ↵
16     /// alterations in input power statuses.
17     /// </summary>
18     /// <returns>The list of outputs that should have their connections    ↵
19     /// called.</returns>
20     protected override List<Output> UpdateOutputs()
21     {
22         bool outputStatus = Outputs[0].Powered;
23         List<Output> outputs = new List<Output>();
24
25         if (outputStatus != Outputs[0].Powered || MaterialNotMatching())    ↵
26             outputs.Add(Outputs[0]);
27
28         return outputs;
29     }
30
31     /// <summary>
32     /// Checks all outputs to determine if the output node material is not    ↵
33     /// matching its power status.<br/><br/>
34     /// This is utilized within custom circuits to force update calls that    ↵
35     /// would normally not occur due to the nature of UpdateOutputs().
36     /// </summary>
37     /// <returns>Whether any output material does not match its power    ↵
38     /// status.</returns>
39     private bool MaterialNotMatching()
40     {
41         if (Outputs[0].StatusRenderer == null) return false;
42
43         return (Outputs[0].Powered && Outputs    ↵
44             [0].StatusRenderer.sharedMaterial !=    ↵
45             CircuitVisualizer.Instance.PowerOnMaterial) ||    ↵
46             (!Outputs[0].Powered && Outputs    ↵
```

...ty Project\Assets\Scripts\Circuits\Starting\Buffer.cs

```
[0].StatusRenderer.sharedMaterial !=  
CircuitVisualizer.Instance.PowerOffMaterial);  
41     }  
42 }
```

2
↗

```
1  using System.Collections.Generic;
2  using UnityEngine;
3
4  ///<summary>
5  ///<summary>Logical representation of a DISPLAY.
6  ///</summary>
7  public class Display : Circuit
8  {
9      ///<summary>
10     ///Similar to <seealso cref="pins"/>, each value refers to the material that is rendered for a corresponding pin when the user's cursor hovers over its corresponding node. ↗
11     ///</summary>
12     private GameObject[] previewPins = new GameObject[8];
13
14     ///<summary>
15     ///Similar to <seealso cref="previewPins"/>, each value refers to the material that is rendered for a corresponding pin when its corresponding node is powered. ↗
16     ///</summary>
17     private MeshRenderer[] pins = new MeshRenderer[8];
18
19     public Display() : this(Vector2.zero) { }
20
21     public Display(Vector2 startingPos) : base("DISPLAY", 8, 0, startingPos) { }
22
23     ///<summary>
24     ///Updates each pin based on the power status of its corresponding node. ↗
25     ///</summary>
26     ///<returns>An empty list of outputs.</returns>
27     protected override List<Output> UpdateOutputs()
28     {
29         for (int i = 0; i < 8; i++) pins[i].material = Inputs[i].Powered ? ↗
30             CircuitVisualizer.Instance.PowerOnMaterial : ↗
31             CircuitVisualizer.Instance.PowerOffMaterial;
32
33         // Always returns an empty list as a DISPLAY has no output nodes.
34         return new List<Output>();
35     }
36
37     // Getter and setter method
38     public GameObject[] PreviewPins { get { return previewPins; } set { previewPins = value; } } ↗
39
40     // Setter method
41     public MeshRenderer[] Pins { set { pins = value; } }
```

```
1  using System.Collections.Generic;
2  using UnityEngine;
3
4  ///<summary>
5  ///<summary>Logical representation of an INPUT gate.
6  ///</summary>
7  public class InputGate : Circuit
8  {
9      ///<summary>
10     ///Powered status unique to an INPUT gate.
11     ///</summary>
12     private bool powered;
13
14     public InputGate() : this(Vector2.zero) { }
15
16     public InputGate(Vector2 startingPos) : base("INPUT", 0, 1, startingPos) { }
17
18     ///<summary>
19     ///Returns an output to update if the output has changed due to alterations in input power statuses.
20     ///</summary>
21     ///<returns>The list of outputs that should have their connections called.</returns>
22     protected override List<Output> UpdateOutputs()
23     {
24         bool outputStatus = Outputs[0].Powered;
25         List<Output> outputs = new List<Output>();
26
27         // INPUT gate representation
28         Outputs[0].Powered = powered;
29
30         if (outputStatus != Outputs[0].Powered) outputs.Add(Outputs[0]);
31
32         return outputs;
33     }
34
35     ///<summary>
36     ///Getter and setter method; setting the powered value of the input node will also create an update call.
37     ///</summary>
38     public bool Powered { get { return powered; } set { powered = value; Update(); UpdateChildren(); } }
39 }
```

```
1  using System.Collections.Generic;
2  using UnityEngine;
3
4  /// <summary>
5  /// Logical representation of an NAND (NOT AND) gate.<br/><br/>
6  /// The NAND gate is also a universal gate.
7  /// </summary>
8  public class NAndGate : Circuit
9  {
10     public NAndGate() : this(Vector2.zero) { }
11
12     public NAndGate(Vector2 startingPos) : base("NAND", 2, 1, startingPos) { }
13
14     /// <summary>
15     /// Returns an output to update if the output has changed due to alterations in input power statuses.
16     /// </summary>
17     /// <returns>The list of outputs that should have their connections called.</returns>
18     protected override List<Output> UpdateOutputs()
19     {
20         bool outputStatus = Outputs[0].Powered;
21         List<Output> outputs = new List<Output>();
22
23         // NAND gate representation
24         Outputs[0].Powered = !(Inputs[0].Powered && Inputs[1].Powered);
25
26         if (outputStatus != Outputs[0].Powered || MaterialNotMatching())
27             outputs.Add(Outputs[0]);
28
29         return outputs;
30     }
31
32     /// <summary>
33     /// Checks all outputs to determine if the output node material is not matching its power status.<br/><br/>
34     /// This is utilized within custom circuits to force update calls that would normally not occur due to the nature of UpdateOutputs().
35     /// </summary>
36     /// <returns>Whether any output material does not match its power status.</returns>
37     private bool MaterialNotMatching()
38     {
39         if (Outputs[0].StatusRenderer == null) return false;
40
41         return (Outputs[0].Powered && Outputs[0].StatusRenderer.sharedMaterial !=
42                 CircuitVisualizer.Instance.PowerOnMaterial) ||
```

```
41         (!Outputs[0].Powered && Outputs  
        [0].StatusRenderer.sharedMaterial !=  
        CircuitVisualizer.Instance.PowerOffMaterial);  
42     }  
43 }
```

```
1  using System.Collections.Generic;
2  using UnityEngine;
3
4  /// <summary>
5  /// Logical representation of an NOR (NOT OR) gate.<br/><br/>
6  /// The NOR gate is also a universal gate.
7  /// </summary>
8  public class NOrGate : Circuit
9  {
10     public NOrGate() : this(Vector2.zero) { }
11
12     public NOrGate(Vector2 startingPos) : base("NOR", 2, 1, startingPos)    ↵
13     { }
14
15     /// <summary>
16     /// Returns an output to update if the output has changed due to    ↵
17     /// alterations in input power statuses.                                ↵
18     /// </summary>
19     /// <returns>The list of outputs that should have their connections    ↵
20     /// called.</returns>
21     protected override List<Output> UpdateOutputs()
22     {
23         bool outputStatus = Outputs[0].Powered;
24         List<Output> outputs = new List<Output>();
25
26         // NOR gate representation
27         Outputs[0].Powered = !(Inputs[0].Powered || Inputs[1].Powered);
28
29         if (outputStatus != Outputs[0].Powered || MaterialNotMatching())    ↵
30             outputs.Add(Outputs[0]);
31
32         return outputs;
33     }
34
35     /// <summary>
36     /// Checks all outputs to determine if the output node material is not    ↵
37     /// matching its power status.<br/><br/>
38     /// This is utilized within custom circuits to force update calls that    ↵
39     /// would normally not occur due to the nature of UpdateOutputs().
40     /// </summary>
41     /// <returns>Whether any output material does not match its power    ↵
42     /// status.</returns>
43     private bool MaterialNotMatching()
44     {
45         if (Outputs[0].StatusRenderer == null) return false;
46
47         return (Outputs[0].Powered && Outputs
48             [0].StatusRenderer.sharedMaterial !=
49             CircuitVisualizer.Instance.PowerOnMaterial) ||
```

...y Project\Assets\Scripts\Circuits\Starting\NOrGate.cs

2

```
41         (!Outputs[0].Powered && Outputs  
        [0].StatusRenderer.sharedMaterial !=  
        CircuitVisualizer.Instance.PowerOffMaterial);  
42     }  
43 }
```

```
1  using System.Collections.Generic;
2  using UnityEngine;
3
4  /// <summary>
5  /// Logical representation of a NOT gate.
6  /// </summary>
7  public class NotGate : Circuit
8  {
9      public NotGate() : this(Vector2.zero) { }
10
11     public NotGate(Vector2 startingPos) : base("NOT", 1, 1, startingPos)    ↵
12         { }
13
14     /// <summary>
15     /// Returns an output to update if the output has changed due to    ↵
16     /// alterations in input power statuses.
17     /// </summary>
18     /// <returns>The list of outputs that should have their connections    ↵
19     /// called.</returns>
20     protected override List<Output> UpdateOutputs()
21     {
22         bool outputStatus = Outputs[0].Powered;
23         List<Output> outputs = new List<Output>();
24
25         // NOT gate representation
26         Outputs[0].Powered = !Inputs[0].Powered;
27
28         if (outputStatus != Outputs[0].Powered || MaterialNotMatching())    ↵
29             outputs.Add(Outputs[0]);
30
31         return outputs;
32     }
33
34     /// <summary>
35     /// Checks all outputs to determine if the output node material is not    ↵
36     /// matching its power status.<br/><br/>
37     /// This is utilized within custom circuits to force update calls that    ↵
38     /// would normally not occur due to the nature of UpdateOutputs().
39     /// </summary>
40     /// <returns>Whether any output material does not match its power    ↵
41     /// status.</returns>
42     private bool MaterialNotMatching()
43     {
44         if (Outputs[0].StatusRenderer == null) return false;
45
46         return (Outputs[0].Powered && Outputs    ↵
47             [0].StatusRenderer.sharedMaterial !=    ↵
48             CircuitVisualizer.Instance.PowerOnMaterial) ||    ↵
49             (!Outputs[0].Powered && Outputs    ↵
50             [0].StatusRenderer.sharedMaterial !=    ↵
51             CircuitVisualizer.Instance.PowerOffMaterial);    ↵
52     }
53 }
```

...y Project\Assets\Scripts\Circuits\Starting\NotGate.cs

```
[0].StatusRenderer.sharedMaterial !=  
CircuitVisualizer.Instance.PowerOffMaterial);  
41     }  
42 }
```

2



```
1  using System.Collections.Generic;
2  using UnityEngine;
3
4  /// <summary>
5  /// Logical representation of an OR gate.
6  /// </summary>
7  public class OrGate : Circuit
8  {
9      public OrGate() : this(Vector2.zero) { }
10
11     public OrGate(Vector2 startingPos) : base("OR", 2, 1, startingPos) { }
12
13     /// <summary>
14     /// Returns an output to update if the output has changed due to    ↵
15     /// alterations in input power statuses.
16     /// </summary>
17     /// <returns>The list of outputs that should have their connections    ↵
18     /// called.</returns>
19     protected override List<Output> UpdateOutputs()
20     {
21         bool outputStatus = Outputs[0].Powered;
22         List<Output> outputs = new List<Output>();
23
24         // OR gate representation
25         Outputs[0].Powered = Inputs[0].Powered || Inputs[1].Powered;
26
27         if (outputStatus != Outputs[0].Powered || MaterialNotMatching())
28             outputs.Add(Outputs[0]);
29
30         return outputs;
31     }
32
33     /// <summary>
34     /// Checks all outputs to determine if the output node material is not    ↵
35     /// matching its power status.<br/><br/>
36     /// This is utilized within custom circuits to force update calls that    ↵
37     /// would normally not occur due to the nature of UpdateOutputs().
38     /// </summary>
39     /// <returns>Whether any output material does not match its power    ↵
40     /// status.</returns>
41     private bool MaterialNotMatching()
42     {
43         if (Outputs[0].StatusRenderer == null) return false;
44
45         return (Outputs[0].Powered && Outputs
46                 [0].StatusRenderer.sharedMaterial !=
47                 CircuitVisualizer.Instance.PowerOnMaterial) ||
48                (!Outputs[0].Powered && Outputs
49                 [0].StatusRenderer.sharedMaterial !=
```

```
41     }
42 }
```

```
1  using System.Collections.Generic;
2  using UnityEngine;
3
4  /// <summary>
5  /// Logical representation of an XOR (EXCLUSIVE OR) gate.
6  /// </summary>
7  public class XOrGate : Circuit
8  {
9      public XOrGate() : this(Vector2.zero) { }
10
11     public XOrGate(Vector2 startingPos) : base("XOR", 2, 1, startingPos)    ↵
12         { }
13
14     /// <summary>
15     /// Returns an output to update if the output has changed due to    ↵
16     /// alterations in input power statuses.                                ↵
17     /// </summary>
18     /// <returns>The list of outputs that should have their connections    ↵
19     /// called.</returns>
20     protected override List<Output> UpdateOutputs()
21     {
22         bool outputStatus = Outputs[0].Powered;
23         List<Output> outputs = new List<Output>();
24
25         // XOR gate representation
26         Outputs[0].Powered = Inputs[0].Powered && !Inputs[1].Powered || !    ↵
27             Inputs[0].Powered && Inputs[1].Powered;
28
29         if (outputStatus != Outputs[0].Powered || MaterialNotMatching())    ↵
30             outputs.Add(Outputs[0]);
31
32         return outputs;
33     }
34
35     /// <summary>
36     /// Checks all outputs to determine if the output node material is not    ↵
37     /// matching its power status.<br/><br/>
38     /// This is utilized within custom circuits to force update calls that    ↵
39     /// would normally not occur due to the nature of UpdateOutputs().        ↵
40     /// </summary>
41     /// <returns>Whether any output material does not match its power    ↵
42     /// status.</returns>
43     private bool MaterialNotMatching()
44     {
45         if (Outputs[0].StatusRenderer == null) return false;
46
47         return (Outputs[0].Powered && Outputs    ↵
48             [0].StatusRenderer.sharedMaterial !=    ↵
49                 CircuitVisualizer.Instance.PowerOnMaterial) ||
```

...y Project\Assets\Scripts\Circuits\Starting\XorGate.cs

2

```
40         (!Outputs[0].Powered && Outputs  
        [0].StatusRenderer.sharedMaterial !=  
        CircuitVisualizer.Instance.PowerOffMaterial);  
41     }  
42 }
```

```
1  using System.Collections.Generic;
2  using System.Linq;
3  using UnityEngine;
4
5  /// <summary>
6  /// Circuit is the parent of every other concretized circuit, containing    ↵
7  /// several predefined constructors, methods, and values.<br/><br/>
8  /// Each circuit must implement the abstract method <seealso             ↵
9  /// cref="UpdateOutputs"/> with the logic to update its outputs.          ↵
10 /// </summary>
11 public abstract class Circuit
12 {
13     /// <summary>
14     /// The time it takes for an update call to occur. This value is      ↵
15     /// measured in seconds.                                                 ↵
16     /// </summary>
17     public readonly static float clockSpeed = 0.075f;
18
19     /// <summary>
20     /// The custom circuit associated with this input.<br/><br/>
21     /// This value will not be null if and only if the circuit is           ↵
22     /// internally within a custom circuit.                                     ↵
23     /// </summary>
24     public CustomCircuit customCircuit;
25
26     /// <summary>
27     /// Whether the circuit should have a representative in-scene mesh      ↵
28     /// generated by <see cref="CircuitVisualizer"/>.                         ↵
29     /// </summary>
30     private bool visible;
31
32     /// <summary>
33     /// The physical mesh generated by <see cref="CircuitVisualizer"/> for    ↵
34     /// this circuit.<br/><br/>
35     /// This value will be null if and only if the circuit is internally     ↵
36     /// within a custom circuit, i.e. no mesh will be generated.            ↵
37     /// </summary>
38     private GameObject physicalObject;
39
40     /// <summary>
41     /// The list of input nodes belonging to the circuit.                   ↵
42     /// </summary>
43     private Input[] inputs;
44
45     /// <summary>
46     /// The list of output nodes belonging to this circuit.                 ↵
47     /// </summary>
48     private Output[] outputs;
```

```
43     /// <summary>
44     /// The list of outputs belonging to this circuit whose power statuses >
45     /// have changed after calling <seealso cref="UpdateOutputs"/>.<br/><br/> >
46     /// Functionally, if an output does not have its power status change >
47     /// before and after an UpdateOutputs() call, it will short circuit and >
48     /// not call any circuits it is connected to.
49     /// </summary>
50     private List<Output> outputsToUpdate;
51
52     /// <summary>
53     /// The name of the circuit.
54     /// </summary>
55     private string circuitName;
56
57     /// <summary>
58     /// Input represents all required members of an input node that belong >
59     /// to a circuit.<br/><br/>
60     /// An input can only have one connection.
61     /// </summary>
62     public class Input
63     {
64         public Input(Circuit parentCircuit) { this.parentCircuit = >
65             parentCircuit; }
66
67         /// <summary>
68         /// Whether the input is powered.
69         /// </summary>
70         private bool powered;
71
72         /// <summary>
73         /// The circuit the input composes.
74         /// </summary>
75         private Circuit parentCircuit;
76
77         /// <summary>
78         /// The connection related to the input, if any.
79         /// </summary>
80         private CircuitConnector.Connection connection;
81
82         /// <summary>
83         /// Contains the material that visually displays whether the input >
84         /// is powered or not.
85         /// </summary>
86         private MeshRenderer statusRenderer;
87
88         /// <summary>
89         /// The output connecting to the input, if any.
90         /// </summary>
```

```
85     private Output parentOutput;
86
87     /// <summary>
88     /// Transform of the GameObject representing the input, if any.
89     /// </summary>
90     private Transform transform;
91
92     // Getter and setter methods
93     public bool Powered { get { return powered; } set { powered = value; } }
94
95     public Circuit ParentCircuit { get { return parentCircuit; } set { parentCircuit = value; } }
96
97     public CircuitConnector.Connection Connection { get { return connection; } set { connection = value; } }
98
99     public MeshRenderer StatusRenderer { get { return statusRenderer; } set { statusRenderer = value; } }
100
101    public Output ParentOutput { get { return parentOutput; } set { parentOutput = value; } }
102
103    public Transform Transform { get { return transform; } set { transform = value; } }
104 }
105
106 /// <summary>
107 /// Output represents all required members of an output node that
108 /// belong to a circuit.<br/><br/>
109 /// An output can more than one connection.
110 /// </summary>
111 public class Output
112 {
113     public Output(Circuit parentCircuit) { this.parentCircuit =
114         parentCircuit; }
115
116     /// <summary>
117     /// Whether the output is powered.
118     /// </summary>
119     private bool powered;
120
121     /// <summary>
122     /// The circuit the output composes.
123     /// </summary>
124     private Circuit parentCircuit;
125
126     /// <summary>
127     /// The connections related to the output, if any.
```

```
126     /// </summary>
127     private List<CircuitConnector.Connection> connections = new
128         List<CircuitConnector.Connection>();
129
130     /// <summary>
131     /// The inputs connecting to the output, if any.
132     /// </summary>
133     private List<Input> childInputs = new List<Input>();
134
135     /// <summary>
136     /// Contains the material that visually displays whether the
137     /// output is powered or not.
138     /// </summary>
139     private MeshRenderer statusRenderer;
140
141     /// <summary>
142     /// Transform of the GameObject representing the output, if any.
143     /// </summary>
144     private Transform transform;
145
146     // Getter and setter methods
147     public bool Powered { get { return powered; } set { powered =
148         value; } }
149
150     public Circuit ParentCircuit { get { return parentCircuit; } set {
151         parentCircuit = value; } }
152
153     public List<CircuitConnector.Connection> Connections { get {
154         return connections; } set { connections = value; } }
155
156     public List<Input> ChildInputs { get { return childInputs; } set {
157         childInputs = value; } }
158
159     public MeshRenderer StatusRenderer { get { return
160         statusRenderer; } set { statusRenderer = value; } }
161
162     public Transform Transform { get { return transform; } set {
163         transform = value; } }
164
165     /// <summary>
166     /// UpdateCall represents an attempt to alter an input node from a
167     /// given output node.<br/><br/>
168     /// An update call does not occur instantly, rather after <seealso
169     /// cref="clockSpeed"/> seconds have passed.<br/>
170     /// This prevents any potential stack overflows caused by loops within
171     /// circuits.
172     /// </summary>
173     public class UpdateCall
```

```
164    {
165        /// <summary>
166        /// Whether the input should be powered.
167        /// </summary>
168        private bool powered;
169
170        /// <summary>
171        /// The input pertaining to this update call.
172        /// </summary>
173        private Input input;
174
175        /// <summary>
176        /// The output pertaining to this update call.
177        /// </summary>
178        private Output output;
179
180        public UpdateCall(bool powered, Input input, Output output)
181    {
182            this.powered = powered;
183            this.input = input;
184            this.output = output;
185        }
186
187        // Getter methods
188        public bool Powered { get { return powered; } }
189
190        public Input Input { get { return input; } }
191
192        public Output Output { get { return output; } }
193    }
194
195    /// <summary>
196    /// Utilized by custom circuits to initialize a circuit with a ↵
197    /// variable number of input and output nodes.<br/><br/> ↵
198    /// With this constructor, it is expected that <seealso cref="Inputs"/> ↵
199    /// and <seealso cref="Outputs"/> will be overriden within <see ↵
200    /// cref="CustomCircuit"/>.
201    /// </summary>
202    /// <param name="circuitName">Name of the circuit.</param>
203    /// <param name="startingPosition">Starting position of the circuit.</param>
204    public Circuit(string circuitName, Vector2 startingPosition) : this ↵
205        (circuitName, 0, 0, startingPosition, false) { }
206
207    /// <summary>
208    /// Utilized by inherited circuits to determine the specific number of ↵
209    /// input and output nodes.
210    /// </summary>
211    /// <param name="circuitName">Name of the circuit.</param>
```

...lde\Unity Project\Assets\Scripts\Circuits\Circuit.cs 6

```
207     /// <param name="numInputs">Number of inputs associated with the    ↵
208     /// circuit.</param>
209     /// <param name="numOutputs">Number of outputs associated with the    ↵
210     /// circuit.</param>
211     /// <param name="startingPosition">Starting position of the circuit.</param>
212     public Circuit(string circuitName, int numInputs, int numOutputs,    ↵
213     Vector2 startingPosition) : this(circuitName, numInputs, numOutputs,    ↵
214     startingPosition, true) { }
215
216     /// <summary>
217     /// Primary constructor that all other constructors reference.
218     /// </summary>
219     /// <param name="circuitName">Name of the circuit.</param>
220     /// <param name="numInputs">Number of inputs associated with the    ↵
221     /// circuit.</param>
222     /// <param name="numOutputs">Number of outputs associated with the    ↵
223     /// circuit.</param>
224     /// <param name="startingPosition">Starting position of the circuit.</param>
225     /// <param name="createIO">Whether each input and output should be    ↵
226     /// initialized.</param>
227     public Circuit(string circuitName, int numInputs, int numOutputs,    ↵
228     Vector2 startingPosition, bool createIO)
229     {
230         this.circuitName = circuitName;
231
232         // Initializes inputs and outputs if specified
233         if (createIO)
234         {
235             inputs = new Input[numInputs];
236             outputs = new Output[numOutputs];
237
238             for (int i = 0; i < numInputs; i++) { inputs[i] = new Input    ↵
239                 (this); }
240
241             for (int i = 0; i < numOutputs; i++) { outputs[i] = new Output    ↵
242                 (this); }
243         }
244
245         /* Determines whether this circuit is meant to be visible.
246          * Within this project, Vector2.PositiveInfinity implicitly    ↵
247          * defines an invisible circuit.
248          * The only circuits that are invisible are ones that are part of    ↵
249          * custom circuits.
250          */
251         visible = startingPosition.x != float.PositiveInfinity &&    ↵
252             startingPosition.y != float.PositiveInfinity;
```

```
241         // Creates a corresponding mesh if the circuit is visible.
242         if (visible) CircuitVisualizer.Instance.VisualizeCircuit(this,      ↵
243             startingPosition);
244     }
245 
246     /// <summary>
247     /// Alternate signature of UpdateCircuit() that assumes the specified ↵
248     /// output is not null.
249     /// </summary>
250     /// <param name="input">The input to update.</param>
251     /// <param name="output">The output that caused the update.</param>
252     public static void UpdateCircuit(Input input, Output output)      ↵
253         { UpdateCircuit(output.Powered, input, output); }
254 
255     /// <summary>
256     /// Updates the circuit belonging to the specified input based on the ↵
257     /// given power status.<br/><br/>
258     /// Afterward, the circuit belonging to the specified input will      ↵
259     /// update all circuits connected to its output(s).
260     /// </summary>
261     /// <param name="powered">Whether the specified input should be    ↵
262     /// powered.</param>
263     /// <param name="input">The input to update.</param>
264     /// <param name="output">The output that caused the update.</param>
265     public static void UpdateCircuit(bool powered, Input input, Output    ↵
266         output)
267     {
268         input.Powered = powered;
269 
270         // Updates input power status material, if applicable
271         if (input.StatusRenderer != null) input.StatusRenderer.material =    ↵
272             powered ? CircuitVisualizer.Instance.PowerOnMaterial :    ↵
273                 CircuitVisualizer.Instance.PowerOffMaterial;
274 
275         // Updates the connection wire material associated with the input, ↵
276         // if applicable
277         if (input.Connection != null)
278             CircuitConnector.UpdateConnectionMaterial(input.Connection,    ↵
279                 powered);
280 
281         input.ParentOutput = output;
282         input.ParentCircuit.Update();
283         input.ParentCircuit.UpdateChildren();
284     }
285 
286     /// <summary>
287     /// Obtains the outputs that should be accessed by <seealso href="UpdateChildren"/> as well as updating their <seealso href="Output.statusRenderer"/> materials.
288 
```

```
276     /// </summary>
277     public void Update()
278     {
279         // If all outputs should be checked, disregard any potential short >
280         // circuiting optimization.
281         bool shouldCheckAllOutputs = customCircuit != null &&
282             customCircuit.finalOutputs.Count > 0;
283
284         outputsToUpdate = UpdateOutputs();
285
286         if (shouldCheckAllOutputs) { outputsToUpdate = Outputs.ToList(); }
287
288         UpdateStatuses();
289     }
290
291     /// <summary>
292     /// Calls and updates all connections associated to each valid
293     /// output.<br/><br/>
294     /// This method can be called recursively, i.e. trigger a chain
295     /// reaction.
296     /// </summary>
297     public void UpdateChildren()
298     {
299         List<UpdateCall> updateList = new List<UpdateCall>();
300
301         foreach (Output output in outputsToUpdate)
302         {
303             if (customCircuit != null &&
304                 customCircuit.finalOutputs.Contains(output))
305                 customCircuit.finalOutputs.Remove(output);
306
307             foreach (Input input in output.ChildInputs) updateList.Add(new >
308                 UpdateCall(output.Powered, input, output));
309         }
310
311         CircuitCaller.InitiateUpdateCalls(updateList);
312     }
313
314     /// <summary>
315     /// Updates the materials of each valid output.
316     /// </summary>
317     private void UpdateStatuses()
318     {
319         foreach (Output output in outputsToUpdate)
320         {
321             if (output.StatusRenderer == null) continue;
322
323             output.StatusRenderer.material = output.Powered ?
324                 CircuitVisualizer.Instance.PowerOnMaterial :
```

```
            }
318        }
319
320        /// <summary>
321        /// Abstract implementation representing the input to output logic of ↵
322        /// a circuit.<br/>
323        /// Utilizes all inputs to recalculate the power status of each ↵
324        /// output.
325        /// </summary>
326        /// <returns>The list of outputs that have changed before and during ↵
327        /// this method.</returns>
328    protected abstract List<Output> UpdateOutputs();
329
330    // Getter and setter methods
331    public bool Visible { get { return visible; } set { visible = ↵
332        value; } }
333
334    public GameObject PhysicalObject { get { return physicalObject; } set ↵
335        { physicalObject = value; } }
336
337    public Input[] Inputs { get { return inputs; } set { inputs = ↵
338        value; } }
339
340    public Output[] Outputs { get { return outputs; } set { outputs = ↵
341        value; } }
342
343    public string CircuitName { get { return circuitName; } set ↵
344        { circuitName = value; } }
345
346    }
```

```
1  using System.Collections.Generic;
2  using UnityEngine;
3
4  /// <summary>
5  /// Logical representation of a custom circuit consisting of a variable
6  /// number/type of other circuits.
7  /// </summary>
8  public class CustomCircuit : Circuit
9  {
10     /// <summary>
11     /// The current custom circuit that is being rendered.<br/><br/>
12     /// This value is utilized to differentiate between external and
13     /// internal (part of a custom circuit) custom circuits.
14     /// </summary>
15     private static CustomCircuit currentCustomCircuit;
16
17     /// <summary>
18     /// Whether or not the custom circuit has been removed and therefore
19     /// dereferenced by its child circuits.
20     /// </summary>
21     private bool shouldDereference;
22
23     /// <summary>
24     /// The list of all internal circuits within the custom circuit.
25     /// </summary>
26     private List<Circuit> circuitList = new List<Circuit>();
27
28     /// <summary>
29     /// The parent GameObject under which all internal connections are
30     /// attached.
31     /// </summary>
32     private GameObject connections;
33
34     /// <summary>
35     /// The list of all inputs within the custom circuit that have no
36     /// connections.<br/><br/>
37     /// All empty inputs are rendered by <see cref="CircuitVisualizer"/>,
38     /// meaning they can be externally connected to other circuits within a
39     /// scene.
40     /// </summary>
41     private List<Input> emptyInputs = new List<Input>();
42
43     /// <summary>
44     /// The list of all inputs within the custom circuit.
45     /// </summary>
46     private List<Input> inputs = new List<Input>();
47
48     /// <summary>
```

...nity Project\Assets\Scripts\Circuits\CustomCircuit.cs 2

```
43     /// The list of all outputs within the custom circuit that have no    ↵
        connections.<br/><br/>
44     /// All empty outputs are rendered by <see cref="CircuitVisualizer"/>, ↵
        meaning they can be externally connected to other circuits within a ↵
        scene.
45     /// </summary>
46     private List<Output> emptyOutputs = new List<Output>();
47
48     /// <summary>
49     /// The list of all empty outputs yet to have received an update    ↵
        call.<br/><br/>
50     /// This list is utilized to ensure that any placed custom circuit is ↵
        properly updated by allowing for update call overrides that would ↵
        otherwise not occur.
51     /// </summary>
52     public List<Output> finalOutputs;
53
54     /// <summary>
55     /// The list of all outputs within the custom circuit.
56     /// </summary>
57     private List<Output> outputs = new List<Output>();
58
59     /// <summary>
60     /// The preview structure the custom circuit is referring to.
61     /// </summary>
62     private PreviewStructure previewStructure;
63
64     /// <summary>
65     /// Alternate signature intended for creating custom circuits that is ↵
        not inside a custom circuit, i.e. external.
66     /// </summary>
67     /// <param name="previewStructure"></param>
68     public CustomCircuit(PreviewStructure previewStructure) : this    ↵
        (previewStructure, Vector2.zero, true) {}
69
70     /// <summary>
71     /// Primary constructor for instantiating any custom circuit.
72     /// </summary>
73     /// <param name="previewStructure">The preview structure the custom    ↵
        circuit is referring to.</param>
74     /// <param name="startingPos">The in-scene position of the circuit    ↵
        (not applicable if the custom circuit is not visible).</param>
75     /// <param name="isFirst">Whether the custom circuit is external, in    ↵
        which case it will be visibly rendered.</param>
76     public CustomCircuit(PreviewStructure previewStructure, Vector2    ↵
        startingPos, bool isFirst) : base(previewStructure.Name,
        Vector2.positiveInfinity)
77     {
78         // If this custom circuit is external, it should be marked as the ↵
```

```
    current custom circuit to be built as well as visible.
79        if (isFirst) { shouldDereference = false; currentCustomCircuit = this; Visible = true; }
80
81        CircuitName = previewStructure.Name;
82        this.previewStructure = previewStructure;
83        CreateCircuit(startingPos);
84    }
85
86    private void CreateCircuit(Vector2 startingPos)
87    {
88        connections = new GameObject("Connections [CUSTOM CIRCUIT]");
89
90        // Instantiates each internal circuit within the custom circuit
91        foreach (CircuitIdentifier circuitIdentifier in previewStructure.Circuits)
92        {
93            Circuit circuit = CircuitIdentifier.RestoreCircuit(circuitIdentifier, false);
94
95            // All non-custom circuits are designated as the child of the current custom circuit
96            if (circuit.GetType() != typeof(CustomCircuit))
97                circuit.customCircuit = this;
98
99            circuitList.Add(circuit);
100
101            foreach (Input input in circuit.Inputs) inputs.Add(input);
102            foreach (Output output in circuit.Outputs) outputs.Add(output);
103        }
104
105        int inputAmount = previewStructure.InputLabels.Count;
106
107        // Restores all empty inputs as designated by the assigned preview structure.
108        for (int i = 0; i < inputAmount; i++) emptyInputs.Add(inputs[previewStructure.InputOrders.IndexOf(i)]);
109
110        int outputAmount = previewStructure.OutputLabels.Count;
111
112        // Restores all empty outputs as designated by the assigned preview structure.
113        for (int i = 0; i < outputAmount; i++) emptyOutputs.Add(outputs[previewStructure.OutputOrders.IndexOf(i)]);
114
115        // Sets the inputs and outputs as ONLY the empty inputs and outputs.
```

```
...nity Project\Assets\Scripts\Circuits\CustomCircuit.cs 4
116         Inputs = emptyInputs.ToArray(); Outputs = emptyOutputs.ToArray();
117
118         int index = 0;
119
120         finalOutputs = new List<Output>(emptyOutputs);
121
122         // If the custom circuit is external/visible (synonymous with one ↵
123         // another), render it into the scene.
124         if (Visible) CircuitVisualizer.Instance.VisualizeCustomCircuit ↵
125             (this, startingPos);
126
127         List<UpdateCall> updateCalls = new List<UpdateCall>();
128
129         // Within the custom circuit, reinstate every connection.
130         foreach (InternalConnection internalConnection in ↵
131             previewStructure.Connections)
132         {
133             CircuitConnector.Connection connection =
134                 connections.AddComponent<CircuitConnector.Connection>();
135             Input input = inputs[internalConnection.InputIndex];
136             Output output = outputs[internalConnection.OutputIndex];
137
138             // Sets all values of the current connection
139             connection.Input = input;
140             connection.Output = output;
141             input.Connection = connection;
142             input.ParentOutput = output;
143             output.Connections.Add(connection);
144             output.ChildInputs.Add(input);
145             updateCalls.Add(new UpdateCall(output.Powered, input,
146                 output));
147             index++;
148
149             // Begins to call each connection.
150             CircuitCaller.InitiateUpdateCalls(updateCalls);
151
152             // Begins the chain reaction to inevitably update the outputs.
153             UpdateOutputs();
154
155             /* Implies that the current custom circuit is a part of another ↵
156             * custom circuit.
157             * As such, it points its custom circuit to the external custom ↵
158             * circuit (parent).
159             * Furthermore, the GameObject holding its connection information ↵
160             * becomes the child of the parent's connection GameObject.
161             */
162             if (!Visible)
163             {
```

```
157         customCircuit = currentCustomCircuit;
158         connections.transform.SetParent
159             (customCircuit.Connections.transform);
160
161     // Implies the current custom circuit IS the external custom
162     // circuit (i.e. currentCustomCircuit == null --> parent custom
163     // circuit).
164
165     else currentCustomCircuit = null;
166 }
167
168 /// <summary>
169 /// Utilized after the instantiation of a custom circuit to update its
170 /// logic to default status.<br/><br/>
171 /// Since a custom circuit does not store the exact predicate that
172 /// controls the output, this method aims to bring about a chain
173 /// reaction from the known inputs to eventually update the outputs in
174 /// variable time.<br/><br/>
175 /// Furthermore, a custom circuit never has its UpdateOutputs() method
176 /// accessed; as such, the return value is not necessary and thus
177 /// yields null.
178 /// </summary>
179 protected override List<Output> UpdateOutputs()
180 {
181     foreach (Input input in emptyInputs) UpdateCircuit(false, input,
182         null);
183
184     return null;
185 }
186
187 // Getter and setter method
188 public bool ShouldDereference { get { return shouldDereference; } set
189     { shouldDereference = value; } }
190
191 // Getter methods
192 public GameObject Connections { get { return connections; } }
193
194 public PreviewStructure PreviewStructure { get { return
195     previewStructure; } }
```

```
1  using System;
2  using System.Collections.Generic;
3  using TMPro;
4  using UnityEngine;
5  using UnityEngine.EventSystems;
6  using UnityEngine.XR;
7  using static UnityEditor.PlayerSettings;
8
9  /// <summary>
10 /// BehaviorManager primarily handles the bulk of all dynamic and scripted ↵
11 /// non-UI events triggerable by the user.
12 /// </summary>
13 public class BehaviorManager : MonoBehaviour
14 {
15     // Singleton state reference
16     private static BehaviorManager instance;
17
18     /// <summary>
19     /// The current state that the editor scene is in.
20     /// </summary>
21     public enum GameState { GRID_HOVER, CIRCUIT_HOVER, CIRCUIT_MOVEMENT, ↵
22         CIRCUIT_PLACEMENT, IO_HOVER, IO_PRESS, USER_INTERFACE, WIRE_HOVER, ↵
23         WIRE_PRESS }
24
25     /// <summary>
26     /// Utilized alongside a <seealso cref="GameState"/> to determine the ↵
27     /// consequences of each game state.<br/><br/>
28     /// <seealso cref="UNRESTRICTED"/>: nothing occurs.<br/>
29     /// <seealso cref="LOCKED"/>: most/all other non-UI elements are ↵
30     /// locked; UI is still enabled.<br/>
31     /// <seealso cref="PAUSED"/>: all non-UI elements are locked.
32     /// </summary>
33     public enum StateType { UNRESTRICTED, LOCKED, PAUSED }
34
35     /// <summary>
36     /// Cancels most non-UI and UI related events when pressed.<br/><br/>
37     /// More often than not, an alternate option to cancel such events ↵
38     /// also occur with the right mouse button.
39     /// </summary>
40     [SerializeField]
41     KeyCode cancelKey;
42
43     /// <summary>
44     /// Displays the name of any hovered inputs/outputs belonging to a ↵
45     /// custom circuit, if any.
46     /// </summary>
47     [SerializeField]
48     TextMeshProUGUI ioText;
```

```
43     /// <summary>
44     /// Reserved for some UI events that should only occur once.
45     /// </summary>
46     private bool doOnce;
47
48     /// <summary>
49     /// Whether the left mouse button was pressed when hovered on an input or output node.
50     /// </summary>
51     private bool ioLMB;
52
53     /// <summary>
54     /// Whether all non-UI elements should remain cancelled no matter what.<br/><br/>.
55     /// If this value is enabled, then control must be restored to the user through external means.
56     /// </summary>
57     private bool lockUI;
58
59     /// <summary>
60     /// Utilized internally for placing, moving, and deleting circuits and their physical GameObjects.
61     /// </summary>
62     private Circuit currentCircuit;
63
64     /// <summary>
65     /// The current input that the user is attempting to connect.
66     /// </summary>
67     private Circuit.Input currentInput;
68
69     /// <summary>
70     /// The current output that the user is attempting to connect.
71     /// </summary>
72     private Circuit.Output currentOutput;
73
74     /// <summary>
75     /// The current preview pin corresponding to an input node that the user is hovered on.
76     /// </summary>
77     private GameObject currentPreviewPin;
78
79     /// <summary>
80     /// Keeps track of the current game state as well as the previous game state if the game is currently paused.
81     /// </summary>
82     private GameState gameState, unpauseGameState;
83
84     /// <summary>
85     /// The opposite layer of the first input or output the user has
```

```
    pressed in a new connection attempt.<br/><br/>
86    /// This helps determine whether the next element to press should be ↵
87    /// an input or output.<br/>
88    /// If an input was first pressed, then the second valid press must be ↵
89    /// for an output, and vice-versa.
90    /// </summary>
91    private int ioLayerCheck;
92
93    /// <summary>
94    /// Utilized for raycasting all in-scene GameObjects to determine the ↵
95    /// current game state and state type.
96    /// </summary>
97    private Ray ray;
98
99    /// <summary>
100   /// Keeps track of the current state type as well as the previous ↵
101   /// state type if the game is currently paused.
102   /// </summary>
103   private StateType stateType, unpauseStateType;
104
105  /// <summary>
106  /// Utilized for moving circuits around the editor scene.
107  /// </summary>
108  private Vector3 deltaPos, endingOffset, prevDeltaPos, startingOffset, ↵
109  /// startingPos;
110
111 // Enforces a singleton state pattern
112 private void Awake()
113 {
114     if (instance != null)
115     {
116         Destroy(this);
117         throw new Exception("BehaviorManager instance already ↵
118         established; terminating.");
119     }
120
121     instance = this;
122
123     // Listens to and acts on additional UI-based events.
124     private void Update()
125     {
126         // If the scene is currently listening to UI, return and disable ↵
127         // set values
128         if (EventSystem.current.IsPointerOverGameObject() || lockUI)
129         {
130             if (currentPreviewPin != null)
131             {
132                 currentPreviewPin.SetActive(false);
133             }
134         }
135     }
136 }
```

```
127             currentPreviewPin = null;
128         }
129
130         // Disables the currently hovered display pin, if any
131         // This specifically occurs once as ioText is externally used ↵
132         // and can have non-empty text.
133         if (doOnce && ioText.text != "") ioText.text = "";
134
135         doOnce = false;
136         return;
137     }
138
139     // Otherwise, checks for all relevant events by beginning to ↵
140     // raycast.
141     doOnce = true;
142     ray = CameraMovement.Instance.PlayerCamera.ScreenPointToRay
143         (Input.mousePosition);
144
145     // If nothing is raycasted, also return and disable set values.
146     if (!Physics.Raycast(ray, out RaycastHit hitInfo))
147     {
148         if (currentPreviewPin != null)
149         {
150             currentPreviewPin.SetActive(false);
151             currentPreviewPin = null;
152         }
153
154         if (ioText.text != "") ioText.text = "";
155
156         return;
157     }
158
159     GameObject hitObj = hitInfo.transform.gameObject;
160
161     // If hovered on an input or output belonging to a custom circuit, ↵
162     // obtain its label.
163     if ((hitObj.layer == 9 || hitObj.layer == 10) &&
164         hitObj.GetComponentInParent<CircuitReference>().Circuit.GetType() ↵
165         () == typeof(CustomCircuit))
166     {
167         CustomCircuit customCircuit = (CustomCircuit)
168             hitObj.GetComponentInParent<CircuitReference>().Circuit;
169
170         int index;
171         string label;
172
173         // Is an input, therefore looks in relevant input variables.
174         if (hitObj.layer == 9)
175         {
```

```
...ject\Assets\Scripts\Editor Scripts\BehaviorManager.cs 5
169         index = Array.IndexOf(customCircuit.Inputs,      ↵
170             hitObj.GetComponent<CircuitVisualizer.InputReference>    ↵
171             ().Input);
172         label = customCircuit.PreviewStructure.InputLabels[index];
173     }
174 
175     // Is an output, therefore looks in relevant output variables.
176     else
177     {
178         index = Array.IndexOf(customCircuit.Outputs,      ↵
179             hitObj.GetComponent<CircuitVisualizer.OutputReference>    ↵
180             ().Output);
181         label = customCircuit.PreviewStructure.OutputLabels      ↵
182             [index];
183     }
184 
185     ioText.text = label;
186 }
187 
188 // Otherwise, therre is no label to display
189 else if (ioText.text != "") ioText.text = "";
190 
191 // If hovered on an input belonging to a display, enable its
192 // corresponding preview pin
193 if (hitObj.layer == 9 &&
194     hitObj.GetComponentInParent<CircuitReference>().Circuit.GetType()
195     () == typeof(Display))
196 {
197     // Occurs if still hovered on the same input node
198     if (currentPreviewPin == hitObj.transform) return;
199 
200     Display display = (Display)      ↵
201         hitObj.GetComponentInParent<CircuitReference>().Circuit;
202     int index = -1;
203 
204     // Determines which preview pin should be enabled
205     for (int i = 0; i < 8; i++)
206     {
207         if (display.Inputs[i].Transform.gameObject == hitObj)
208         {
209             index = i;
210             break;
211         }
212     }
213 
214     // If the last frame focused on a separate preview pin,
215     // disable that first
216     if (currentPreviewPin != null) currentPreviewPin.SetActive
217         (false);
```

```
207
208         // Enable the current preview pin
209         currentPreviewPin = display.PreviewPins[index];
210         currentPreviewPin.SetActive(true);
211     }
212
213     // Otherwise, disable the current preview pin, if any
214     else if (currentPreviewPin != null)
215     {
216         currentPreviewPin.SetActive(false);
217         currentPreviewPin = null;
218     }
219 }
220
221 // Obtains a new game state/state type, and if applicable, listens to ↵
222 // input/events corresponding to the game state.
223 private void LateUpdate()
224 {
225     gameState = UpdateGameState();
226     GameStateListener();
227 }
228
229 /// <summary>
230 /// Obtains a new GameState by performing a raycast in combination ↵
231 /// with the current game state.
232 /// </summary>
233 /// <returns>The new game state to switch to</returns>
234 private GameState UpdateGameState()
235 {
236     // Current state is UI
237     if (EventSystem.current.IsPointerOverGameObject() || lockUI)
238     {
239         if (gameState == GameState.USER_INTERFACE) return ↵
240             gameState; // Last state was UI, return.
241
242         // The UI state pauses the previous game state/state type, ↵
243         // storing it in separate paused values.
244         unpauseGameState = gameState;
245         unpauseStateType = stateType;
246         stateType = StateType.PAUSED;
247         Cursor.visible = true;
248         CursorManager.SetMouseTexture(true);
249         return GameState.USER_INTERFACE;
250     }
251
252     // Current game state is not UI but the previous game state was.
253     // Therefore, restore the game state/state type present before the ↵
254     // user hovered onto UI.
255     if (gameState == GameState.USER_INTERFACE)
```

```
251     {
252         gameState = unpauseGameState;
253         stateType = unpauseStateType;
254
255         // Conditions for a visible cursor
256         Cursor.visible = unpauseGameState != GameState.CIRCUIT_MOVEMENT && unpauseGameState != GameState.CIRCUIT_PLACEMENT;
257     }
258
259         // Locked states must change manually, not automatically.
260         if (stateType == StateType.LOCKED) return gameState;
261
262         // The raycast reached nothing -- defaults to the grid hover state.
263         if (!Physics.Raycast(ray, out RaycastHit hitInfo))
264     {
265             stateType = StateType.UNRESTRICTED;
266             CursorManager.SetMouseTexture(true);
267             return GameState.GRID_HOVER;
268         }
269
270         GameObject hitObject = hitInfo.transform.gameObject;
271
272         // Mouse is on top of a circuit & LMB and/or RMB have been pressed
273         if (gameState == GameState.CIRCUIT_HOVER && (Input.GetMouseButtonDown(0) || Input.GetMouseButtonDown(1)))
274     {
275             currentCircuit =
276                 hitObject.GetComponentInParent<CircuitReference>().Circuit;
277
278             // Left click (or both): circuit movement begins.
279             if (Input.GetMouseButtonDown(0))
280             {
281                 CircuitPress();
282                 stateType = StateType.LOCKED;
283                 CursorManager.SetMouseTexture(false);
284                 Cursor.visible = false;
285             }
286
287             // Right click: destroy current circuit.
288             else
289             {
290                 CircuitCaller.Destroy(currentCircuit);
291                 stateType = StateType.UNRESTRICTED;
292             }
293
294             return GameState.CIRCUIT_MOVEMENT;
295     }
```

```
295         // Mouse is on top of a circuit
296         if (hitObject.layer == 8) // 8 --> circuit base layer
297         {
298             stateType = StateType.UNRESTRICTED;
299             CursorManager.SetMouseTexture(false);
300             return GameState.CIRCUIT_HOVER;
301         }
302     }
303
304     // Mouse is on top of an input/output & LMB and/or RMB and/or MMB ↵
305     // have been pressed
306     if (gameState == GameState.IO_HOVER && (Input.GetMouseButtonDown(0) ↵
307         || Input.GetMouseButtonDown(1) || Input.GetMouseButtonDown(2)))
308     {
309         ioLMB = Input.GetMouseButtonDown(0);
310         stateType = StateType.LOCKED;
311
312         // Left click (perahsp with other inputs, but LMB has the ↵
313         // highest preference): begins the connection process
314         if (ioLMB)
315         {
316             IOLMBPress(hitObject);
317             CursorManager.SetMouseTexture(true);
318         }
319
320         // Right click or middle mouse button: alternate press ↵
321         // RMB: deletes all connections attached to the input/output ↵
322         // in question
323         // MMB (only applicable if hovered onto an input gate's ↵
324         // output): switch power states
325         else IOAlternatePress(hitObject);
326
327         return GameState.IO_PRESS;
328     }
329
330     // Mouse is on top of any input or output
331     if (hitObject.layer == 9 || hitObject.layer == 10)
332     {
333         stateType = StateType.UNRESTRICTED;
334         CursorManager.SetMouseTexture(false);
335         return GameState.IO_HOVER;
336     }
337
338     // Mouse is on top of a wire & RMB has been pressed
339     if (gameState == GameState.WIRE_HOVER && Input.GetMouseButtonDown(1))
340     {
341         stateType = StateType.LOCKED;
```

```
337         CursorManager.SetMouseTexture(true);
338         WirePress(hitObject); // Deletes the wire and its
339         // corresponding connection
340     }
341
342     // Mouse is on top of a wire
343     if (hitObject.layer == 11)
344     {
345         stateType = StateType.UNRESTRICTED;
346         CursorManager.SetMouseTexture(false);
347         return GameState.WIRE_HOVER;
348     }
349
350     // If none of the other conditions were met, default to the grid
351     // hover state instead.
352     stateType = StateType.UNRESTRICTED;
353     CursorManager.SetMouseTexture(true);
354     return GameState.GRID_HOVER;
355 }
356
357 /// <summary>
358 /// Begins the connection process.
359 /// </summary>
360 /// <param name="hitObject">The GameObject that was raycasted.</param>
361 private void IOlMBPress(GameObject hitObject)
362 {
363     Vector3 startingPos;
364
365     // Input layer was pressed; next press should be on an output
366     // layer
367     if (hitObject.layer == 9)
368     {
369         currentInput =
370             hitObject.GetComponent<CircuitVisualizer.InputReference>()
371             .Input;
372         ioLayerCheck = 10;
373         startingPos = currentInput.Transform.position;
374     }
375
376     // Output layer was pressed; next press should be on an input
377     // layer
378     else
379     {
380         currentOutput =
381             hitObject.GetComponent<CircuitVisualizer.OutputReference>()
382             .Output;
383         ioLayerCheck = 9;
384         startingPos = currentOutput.Transform.position;
385     }
386 }
```

```
378         }
379
380         CircuitConnector.Instance.BeginConnectionProcess(startingPos);
381     }
382
383     /// <summary>
384     /// Based on context, deletes all connections belonging to an input or &
385     /// output node or alternates the power status of an input gate.
386     /// </summary>
387     /// <param name="hitObject"></param>
388     private void IOAlternatePress(GameObject hitObject)
389     {
390         // If the raycasted object was an input and the MMB is pressed, &
391         // alternate its input
392         if (Input.GetMouseButtonDown(2))
393         {
394             if (hitObject.layer == 10 &&
395                 hitObject.GetComponentInParent<CircuitReference>()
396                 .Circuit.GetType() == typeof(InputGate))
397             {
398                 InputGate gate = (InputGate)
399                     hitObject.GetComponentInParent<CircuitReference>()
400                     .Circuit;
401
402                 gate.Powered = !gate.Powered;
403                 EditorStructureManager.Instance.DisplaySavePrompt =
404                     true; // Important enough to trigger the save prompt
405             }
406
407             // RMB on an input -- begin disconnection process
408             else if (hitObject.layer == 9)
409             {
410                 Circuit.Input input =
411                     hitObject.GetComponent<CircuitVisualizer.InputReference>()
412                     .Input;
413
414                 // If there is a connection, disconnect it.
415                 if (input.Connection != null) CircuitConnector.Disconnect
416                     (input.Connection);
417             }
418
419             // RMB on an output -- begin disconnection process
420             else
421             {
422                 Circuit.Output output =
423                     hitObject.GetComponent<CircuitVisualizer.OutputReference>()
424                     .Output;
```

```
415         List<CircuitConnector.Connection> connections = new List<CircuitConnector.Connection>(output.Connections);
416
417         // Disconnects each connection associated with this output, if any.
418         foreach (CircuitConnector.Connection connection in connections) CircuitConnector.Disconnect(connection);
419     }
420
421     stateType = StateType.UNRESTRICTED;
422 }
423
424 /// <summary>
425 /// Called after a new circuit has been instantiated; sets initial values.
426 /// </summary>
427 /// <param name="currentCircuit">The circuit that has just been created.</param>
428 public void CircuitPlacement(Circuit currentCircuit)
429 {
430     // If there is already a circuit in the process of being placed, destroy it.
431     if (this.currentCircuit != null) { CircuitCaller.Destroy(this.currentCircuit); }
432
433     this.currentCircuit = currentCircuit;
434     currentCircuit.PhysicalObject.transform.position =
435         Coordinates.Instance.MousePos;
436
437     /// <summary>
438     /// Deletes a wire GameObject and its associated connection
439     /// </summary>
440     /// <param name="hitObject">The wire to delete.</param>
441     private void WirePress(GameObject hitObject)
442     {
443         CircuitConnector.Connection connection;
444
445         // Determines if the raycasted object is the parent mesh (aka not the starting/ending wire mesh).
446         if (hitObject.transform.parent == null)
447         {
448             connection =
449                 hitObject.GetComponent<CircuitConnector.Connection>();
450             Destroy(hitObject.transform.gameObject);
451         }
452
453         // Otherwise, is a starting or ending wire.
454         else
```

```
454         {
455             connection =
456                 hitObject.GetComponentInParent<CircuitConnector.Connection>
457                     ();
458             Destroy(hitObject.transform.parent.parent.gameObject);
459         }
460     CircuitConnector.Disconnect(connection); // Disconnects the logic
461     stateType = StateType.UNRESTRICTED;
462 }
463 /// <summary>
464 /// Called after a circuit has been pressed; sets initial values.
465 /// </summary>
466 private void CircuitPress()
467 {
468     Vector3 mousePos = Coordinates.Instance.MousePos;
469
470     startingPos = currentCircuit.PhysicalObject.transform.position;
471     endingOffset = startingOffset = mousePos;
472 }
473
474 /// <summary>
475 /// Cancels the connection process.
476 /// </summary>
477 public void CancelWirePlacement()
478 {
479     CircuitConnector.Instance.CancelConnectionProcess();
480     currentInput = null; currentOutput = null;
481 }
482
483 /// <summary>
484 /// Cancels the circuit movement process.
485 /// </summary>
486 public void CancelCircuitMovement()
487 {
488     Cursor.visible = true;
489     currentCircuit = null;
490 }
491
492 /// <summary>
493 /// Called after obtaining a new game state; listens to input/events
494     corresponding to the game state.
495 /// </summary>
496 private void GameStateListener()
497 {
498     switch (gameState)
499     {
```

```
499         case GameState.GRID_HOVER:
500             // Opens the bookmarked circuits menu.
501             if (Input.GetMouseButtonDown(1) &&
502                 TaskbarManager.Instance.CurrentMenu == null &&
503                 TaskbarManager.Instance.ReopenBookmarks)
504                 TaskbarManager.Instance.OpenBookmarks();
505
506             return;
507         case GameState.IO_PRESS:
508             if (!ioLMB) return; // The left mouse button was not
509             pressed, therefore the corresponding connection code
510             should be skipped.
511
512             // Checks to see if the user is hovered on a valid
513             // GameObject to complete the connection process.
514             if (Physics.Raycast
515                 (CameraMovement.Instance.PlayerCamera.ScreenPointToRay
516                  (Input.mousePosition), out RaycastHit hitInfo) &&
517                 hitInfo.transform.gameObject.layer == ioLayerCheck)
518             {
519                 // Output layer was initially pressed, therefore this
520                 // is an input node
521                 if (ioLayerCheck == 9) currentInput =
522                     hitInfo.transform.GetComponent<CircuitVisualizer.InputReference>().Input;
523
524                 // Input layer was initially pressed, therefore this
525                 // is an output node
526                 else currentOutput =
527                     hitInfo.transform.GetComponent<CircuitVisualizer.OutputReference>().Output;
528
529                 CursorManager.SetMouseTexture(false);
530
531                 // The user completes the connection process by
532                 // hovering on a valid input AND pressing the left mouse
533                 // button.
534                 if (Input.GetMouseButtonDown(0))
535                 {
536                     EditorStructureManager.Instance.DisplaySavePrompt
537                     = true; // Important enough to trigger the save prompt
538
539                     // Disconnects the current connection to the
540                     // input, if there is one
541                     if (currentInput.ParentOutput != null)
542                         CircuitConnector.Disconnect(currentInput.Connection);
543
544                     CircuitConnector.Connection connection =
545                     CircuitConnector.Instance.CurrentConnection;
```

```
527
528             CircuitConnector.Connect(currentInput,           ↵
529             currentOutput); // Ensures the connection is logically    ↵
530             accounted for
531
532             // If the order of selection was not output ->      ↵
533             input, the starting and ending wires are swapped with    ↵
534             one another.
535             // This occurs because the starting wire is always    ↵
536             associated with the input node, hence the GameObjects    ↵
537             are swapped to maintain this rule.
538             if (ioLayerCheck == 10)
539             {
540                 GameObject temp = connection.StartingWire;
541
542                 // Swaps the starting and ending wires within    ↵
543                 the connection
544                 connection.StartingWire =
545                 connection.EndingWire;
546                 connection.EndingWire = temp;
547
548                 // Ensures the serialization process works as    ↵
549                 intended by keeping the hierarchy order of the wires the    ↵
550                 same, regardless of connection order.
551                 if (connection.StartingWire !=           ↵
552                 connection.EndingWire)
553                 {
554                     connection.StartingWire.name = "Starting    ↵
555                     Wire";
556                     connection.EndingWire.name = "Ending    ↵
557                     Wire";
558
559                     connection.StartingWire.transform.SetAsFirstSibling();
560                 }
561
562             stateType = StateType.UNRESTRICTED;
563             currentInput = null; currentOutput = null;
564             return;
565         }
566     }
567
568     else CursorManager.SetMouseTexture(true);
569
570     // Cancels the connection process.
571     if (Input.GetKeyDown(cancelKey) ||
572         Input.GetMouseButtonUp(1))
573     {
574         CancelWirePlacement();
```

```
561             stateType = StateType.UNRESTRICTED;
562         }
563
564         break;
565     case GameState.CIRCUIT_MOVEMENT:
566         // Cancels the circuit movement process if the left mouse ↵
567         // button is not held.
568         if (!Input.GetMouseButton(0))
569         {
570             CancelCircuitMovement();
571             stateType = StateType.UNRESTRICTED;
572             return;
573         }
574
575         // Calculates the delta mouse movement from the last frame
576         endingOffset = Coordinates.Instance.MousePos;
577         prevDeltaPos = deltaPos;
578         deltaPos = endingOffset - startingOffset + startingPos;
579
580         // Snaps the obtained position to the grid if grid
581         // snapping is enabled.
582         if (Coordinates.Instance.CurrentSnappingMode == ↵
583             Coordinates.SnappingMode.GRID) deltaPos = ↵
584             Coordinates.NormalToGridPos(deltaPos);
585
586         currentCircuit.PhysicalObject.transform.position = ↵
587             deltaPos;
588
589         if (prevDeltaPos != deltaPos) // Ensures the circuit has ↵
590         // moved from its previous position before updating the ↵
591         // transforms of both wire GameObjects.
592         {
593             EditorStructureManager.Instance.DisplaySavePrompt = ↵
594                 true; // Important enough to trigger the save prompt
595
596             // Updates the position/scale each valid connection
597             // associated with the inputs of the moved circuit.
598             // This occurs so that each physical wire continues to ↵
599             // stretch/shrink and follow each circuit within the ↵
600             // scene.
601             foreach (Circuit.Input input in currentCircuit.Inputs)
602             {
603                 if (input.Connection != null)
604                 {
605                     bool isCentered = input.Connection.EndingWire ↵
606                     == input.Connection.StartingWire;
607                     Vector3 fromPos = isCentered ? ↵
608                         input.Connection.Output.Transform.position : ↵
609                         input.Connection.EndingWire.transform.position;
```

```
596
597             CircuitConnector.UpdatePosition
598             (input.Connection.EndingWire, fromPos,
599              input.Transform.position, isCentered);
600         }
601     }
602
603     // Updates the position/scale each valid connection
604     // associated with the outputs of the moved circuit.
605     // This occurs so that each physical wire continues to
606     // stretch/shrink and follow each circuit within the
607     // scene.
608     foreach (Circuit.Output output in
609             currentCircuit.Outputs)
610     {
611         foreach (CircuitConnector.Connection connection in
612             output.Connections)
613         {
614             bool isCentered = connection.EndingWire ==
615             connection.StartingWire;
616             Vector3 fromPos = isCentered ?
617             connection.Input.Transform.position :
618             connection.StartingWire.transform.position;
619
620             CircuitConnector.UpdatePosition
621             (connection.StartingWire, fromPos,
622              output.Transform.position, isCentered);
623         }
624     }
625
626     break;
627 case GameState.CIRCUIT_PLACEMENT:
628     // Until its placement is confirmed, the circuit follows
629     // the mouse cursor.
630     currentCircuit.PhysicalObject.transform.position =
631         Coordinates.Instance.ModePos;
632
633     // Placement is confirmed
634     if (Input.GetMouseButtonDown(0))
635     {
636         Cursor.visible = true;
637         EditorStructureManager.Instance.Circuits.Add
638         (currentCircuit); // Adds circuit for potential
639         serialization
640         EditorStructureManager.Instance.DisplaySavePrompt =
641         true;
642         currentCircuit = null;
643         stateType = StateType.UNRESTRICTED;
```

```
628             LateUpdate();
629             return;
630         }
631
632         // Placement is cancelled; delete the circuit.
633         if (Input.GetKeyDown(cancelKey) ||
634             Input.GetMouseButtonUp(1))
635         {
636             Cursor.visible = true;
637             CircuitCaller.Destroy(currentCircuit);
638             currentCircuit = null;
639             stateType = StateType.UNRESTRICTED;
640         }
641         break;
642     }
643 }
644
645 // Getter and setter methods
646 public GameState UnpausedGameState { get { return unpausedGameState; } } ↵
647             set { unpausedGameState = value; } }
648
649 public StateType UnpausedStateType { get { return unpausedStateType; } } ↵
650             set { unpausedStateType = value; } }
651
652 // Getter methods
653 public static BehaviorManager Instance { get { return instance; } }
654
655 public bool LockUI { get { return lockUI; } set { lockUI = value; } }
656
657 public GameState CurrentGameState { get { return gameState; } }
658
659 public int IOLayerCheck { get { return ioLayerCheck; } }
660 }
```

```
1  using System;
2  using UnityEngine;
3
4  /// <summary>
5  /// CameraMovement handles all player movement within the editor
6  /// scene.<br/><br/>
7  /// Some behaviors (e.g. scrolling) will be enabled or disabled based on
8  /// the state of several other scripts.
9  /// </summary>
10 public class CameraMovement : MonoBehaviour
11 {
12     // Singleton state reference
13     private static CameraMovement instance;
14
15     /// <summary>
16     /// The primary camera utilized by the player.
17     /// </summary>
18     [SerializeField]
19     Camera playerCamera;
20
21     /// <summary>
22     /// The speed under which the player can move around scenes.
23     /// </summary>
24     [SerializeField]
25     float movementSpeed;
26
27     /// <summary>
28     /// The speed under which the player can scroll around scenes.
29     /// </summary>
30     [SerializeField]
31     float scrollSpeed;
32
33     /// <summary>
34     /// How low and high the player can vertically go.
35     /// </summary>
36     [SerializeField]
37     float minHeight, maxHeight;
38
39     /// <summary>
40     /// Moves the player up and down respectively.
41     /// </summary>
42     [SerializeField]
43     KeyCode upKey, downKey;
44
45     /// <summary>
46     /// Keeps track of the mouse position in the current frame.
47     /// </summary>
48     private Vector3 mousePosCurrent;
```

```
48     // Enforces a singleton state pattern and initializes camera values.
49     private void Awake()
50     {
51         if (instance != null)
52         {
53             Destroy(this);
54             throw new Exception("CameraMovement instance already
55                             established; terminating.");
56         }
57         instance = this;
58         ClampPos();
59     }
60
61     private void Start() { mousePosCurrent =
62                         Coordinates.Instance.MousePos; }
63
64     // Listens to key inputs and updates movements each frame.
65     private void Update()
66     {
67         float x, y, z;
68
69         Vector3 mousePosPrev = mousePosCurrent;
70
71         mousePosCurrent = Coordinates.Instance.MousePos;
72
73         // If the scene is paused or there is an override, no movement can occur.
74         if (BehaviorManager.Instance.CurrentStateType ==
75             BehaviorManager.StateType.PAUSED && !
76             IOAssigner.Instance.MovementOverride) return;
77
78         // Otherwise, some/all movement features are allowed depending on whether the game is unrestricted or locked.
79         // X-Z movement via mouse drag
80         if (Input.GetMouseButton(0) &&
81             BehaviorManager.Instance.CurrentStateType ==
82             BehaviorManager.StateType.UNRESTRICTED &&
83             BehaviorManager.Instance.CurrentGameState !=
84             BehaviorManager.GameState.CIRCUIT_HOVER)
85         {
86             Vector3 mousePosDelta = mousePosPrev - mousePosCurrent;
87
88             x = mousePosDelta.x;
89             z = mousePosDelta.z;
90         }
91
92         // X-Z movement via WASD
93         else
```

```
87     {
88         // Obtains x and z axis values based on input
89         x = Input.GetAxisRaw("Horizontal") * movementSpeed * Time.deltaTime;
90         z = Input.GetAxisRaw("Vertical") * movementSpeed * Time.deltaTime;
91     }
92
93     // Y movement via scroll wheel
94     if (Mathf.Abs(Input.mouseScrollDelta.y) > 0)
95     {
96         y = -Input.mouseScrollDelta.y * scrollSpeed * Time.deltaTime;
97     }
98
99     // Y movement via upKey and/or downKey
100    else
101    {
102        y = 0;
103
104        // Determines y axis values (holding both "upKey" and
105        // "downKey" will negate one another)
106        if (Input.GetKey(upKey))
107        {
108            y += movementSpeed * Time.deltaTime;
109        }
110        if (Input.GetKey(downKey))
111        {
112            y -= movementSpeed * Time.deltaTime;
113        }
114    }
115
116    // Adds obtained values and updates position
117    transform.position += x * Vector3.right + y * -CameraRay.direction +
118        + z * Vector3.forward;
119    ClampPos();
120    mousePosCurrent = Coordinates.Instance.MousePos;
121
122    /// <summary>
123    /// Clamps values to ensure the user cannot traverse out of bounds.
124    /// </summary>
125    private void ClampPos()
126    {
127        Vector3 pos = transform.position;
128
129        pos.y = Mathf.Clamp(pos.y, minHeight, maxHeight);
130        transform.position = pos;
131    }
```

```
132
133     // Getter methods
134     public static CameraMovement Instance { get { return instance; } }
135
136     public Camera PlayerCamera { get { return playerCamera; } }
137
138     private Ray CameraRay { get { return playerCamera.ScreenPointToRay
139         (Input.mousePosition); } }
```

```
1  using System;
2  using TMPro;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  /// <summary>
7  /// Coordinates keeps track of the world position as well as the grid      ↵
8  /// snapping mode.
9  /// </summary>
10 public class Coordinates : MonoBehaviour
11 {
12     // Singleton state reference
13     private static Coordinates instance;
14
15     /// <summary>
16     /// Dictates how the current world position within the editor scene      ↵
17     /// should be interpreted.<br/>
18     /// This modified position is utilized for several actions within the      ↵
19     /// scene, such as placing wires and moving circuits.<br/><br/>
20     /// <seealso cref="GRID"/>: snap current mouse position to the visual      ↵
21     /// grid.<br/>
22     /// <seealso cref="NONE"/>: keep the current mouse position as is.
23     /// </summary>
24     public enum SnappingMode { GRID, NONE }
25
26     /// <summary>
27     /// The transparency value of <seealso cref="gridStatus"/> when      ↵
28     /// <seealso cref="SnappingMode.NONE"/> is enabled.
29     /// </summary>
30     [SerializeField] float gridTransparencyConstant;
31
32     /// <summary>
33     /// In-scene icon that visualizes the status of <seealso      ↵
34     /// cref="snappingMode"/>.
35     /// </summary>
36     [SerializeField] Image gridStatus;
37
38     /// <summary>
39     /// Toggles the <seealso cref="SnappingMode"/> currently not in use.
40     /// </summary>
41     [SerializeField] KeyCode snapToggleKey;
42
43     /// <summary>
44     /// Displays the current world coordinates to the user.
45     /// </summary>
46     [SerializeField] TextMeshProUGUI coordinateText;
```

```
        >.  
44     /// </summary>  
45     private Color gridStatusColor;  
46  
47     /// <summary>  
48     /// Utilized to perform a raycast to calculate <seealso  
     cref="mousePos"/>.  
49     /// </summary>  
50     private Plane raycastPlane;  
51  
52     /// <summary>  
53     /// The current <seealso cref="SnappingMode"/>.  
54     /// </summary>  
55     private SnappingMode snappingMode;  
56  
57     /// <summary>  
58     /// Stores the calculated mouse to world position.  
59     /// </summary>  
60     private Vector3 mousePos;  
61  
62     private void Update()  
63     {  
64         // If the snap toggle key is pressed at a valid time, switch  
         states.  
65         if (Input.GetKeyDown(snapToggleKey) &&  
             BehaviorManager.Instance.CurrentStateType !=  
             BehaviorManager.StateType.PAUSED)  
66         {  
67             snappingMode = snappingMode == SnappingMode.GRID ?  
                 SnappingMode.NONE : SnappingMode.GRID;  
68             CurrentSnappingMode = snappingMode; // Ensures the UI is also  
                 updated.  
69         }  
70     }  
71  
72     private void Awake()  
73     {  
74         // Enforces a singleton state pattern  
75         if (instance != null)  
76         {  
77             Destroy(this);  
78             throw new Exception("Coordinates instance already established; terminating.");  
79         }  
80  
81         instance = this;  
82  
83         // Initializes private values  
84         raycastPlane = new Plane(Vector3.down,
```

```
85         gridStatusColor = gridStatus.color;
86     }
87
88     /// <summary>
89     /// Snaps the specified position to the grid.
90     /// </summary>
91     /// <param name="normalPos">The position that should be snapped to the >
92     /// grid.</param>
93     /// <returns>The grid position.</returns>
94     public static Vector3 NormalToGridPos(Vector3 normalPos) { return new >
95         Vector3((int)(normalPos.x + 0.5f * Mathf.Sign(normalPos.x)), >
96             GridMaintenance.Instance.GridHeight, (int)(normalPos.z + 0.5f * >
97             Mathf.Sign(normalPos.z))); }
98
99     // Getter methods
100    public static Coordinates Instance { get { return instance; } }
101
102    /// <summary>
103    /// Returns a new ray from the camera to the current mouse position.
104    /// </summary>
105    private Ray CameraRay { get { return
106        CameraMovement.Instance.PlayerCamera.ScreenPointToRay
107        (Input.mousePosition); } }
108
109    /// <summary>
110    /// Calculates and returns the current grid position.
111    /// </summary>
112    public Vector3 GridPos { get { return NormalToGridPos(mousePos); } }
113
114    /// <summary>
115    /// Calculates and returns the current mouse position.
116    /// </summary>
117    public Vector3 MousePos
118    {
119        get
120        {
121            Ray ray = CameraRay;
122
123            if (raycastPlane.Raycast(ray, out float distance))
124            {
125                mousePos = ray.GetPoint(distance);
126
127                // Updates the coordinates UI if the game is not currently >
128                // paused
129                if (BehaviorManager.Instance.CurrentStateType !=
130                    BehaviorManager.StateType.PAUSED) coordinateText.text =
131                    "(" + mousePos.x.ToString("0.0") + ", " +
132                    mousePos.z.ToString("0.0") + ")";
133            }
134        }
135    }
```

```
123             return new Vector3(mousePos.x,
124                                 GridMaintenance.Instance.GridHeight, mousePos.z);
125         }
126
127         throw new Exception("Unable to obtain new mouse position -- >
128             raycast failed.");
129     }
130
131     /// <summary>
132     /// Returns a modified version of <seealso cref="mousePos"/> based on >
133     /// <seealso cref="snappingMode"/>.
134     public Vector3 ModePos { get { return snappingMode ==
135         SnappingMode.GRID ? GridPos : MousePos; } }
136
137     /// <summary>
138     /// Serves as a getter method as well as a setter method for both >
139     /// <seealso cref="snappingMode"/> and <seealso cref="gridStatus"/>.
140     public SnappingMode CurrentSnappingMode { get { return snappingMode; } }
141     set
142     {
143         snappingMode = value;
144
145         if (value == SnappingMode.GRID)
146         {
147             gridStatus.color = gridStatusColor;
148
149         }
150         else
151         {
152             Color temp = gridStatusColor;
153
154             temp.a = gridTransparencyConstant;
155             gridStatus.color = temp;
156         }
157     }
158 }
```

```
1  using System;
2  using UnityEngine;
3
4  /// <summary>
5  /// DisplayReference is assigned to and stores values unique to the Display ↵
6  ///     prefab.
7  /// </summary>
8  public class DisplayReference : MonoBehaviour
9 {
10    /// <summary>
11    /// The inputs of the display.
12    /// </summary>
13    [SerializeField]
14    GameObject[] inputs = new GameObject[8];
15
16    /// <summary>
17    /// The preview pins (on hover) corresponding to each value within      ↵
18    ///     <seealso cref="inputs"/>.
19    /// </summary>
20    [SerializeField]
21    GameObject[] previewPins = new GameObject[8];
22
23    /// <summary>
24    /// The input statuses of the display.
25    /// </summary>
26    [SerializeField]
27    MeshRenderer[] inputStatuses = new MeshRenderer[8];
28
29    /// <summary>
30    /// The pins (on power) corresponding to each value within <seealso      ↵
31    ///     cref="inputs"/>.
32    /// </summary>
33    [SerializeField]
34    MeshRenderer[] pins = new MeshRenderer[8];
35
36    // Ensures the sizes of each array cannot be modified within the      ↵
37    //     inspector
38    private void OnValidate()
39    {
40        if (inputs.Length != 8) Array.Resize(ref inputs, 8);
41
42        if (inputStatuses.Length != 8) Array.Resize(ref inputStatuses, 8);
43
44        if (pins.Length != 8) Array.Resize(ref pins, 8);
45
46        if (previewPins.Length != 8) Array.Resize(ref previewPins, 8);
47    }
48
49    // Getter methods
```

```
46     public GameObject[] Inputs { get { return inputs; } }
```

```
47     public GameObject[] PreviewPins { get { return previewPins; } }
```

```
48     public MeshRenderer[] InputStatuses { get { return inputStatuses; } }
```

```
49     public MeshRenderer[] Pins { get { return pins; } }
```

```
50 }
```

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5
6  /// <summary>
7  /// EditorStructureManager accesses the current <see href="EditorStructure"/> in use and directly manages its serialization/ deserialization. >
8  /// </summary>
9  public class EditorStructureManager : MonoBehaviour
10 {
11     // Singleton state reference
12     private static EditorStructureManager instance;
13
14     /// <summary>
15     /// List of instantiated circuits in the current editor scene.
16     /// </summary>
17     [HideInInspector]
18     List<Circuit> circuits = new List<Circuit>();
19
20     /// <summary>
21     /// Index representation of all currently bookmarked circuits.
22     /// </summary>
23     [HideInInspector]
24     List<int> bookmarks = new List<int>();
25
26     /// <summary>
27     /// List of instantiated connections in the current editor scene.
28     /// </summary>
29     [HideInInspector]
30     List<CircuitConnector.Connection> connections = new List<CircuitConnector.Connection>();
31
32     /// <summary>
33     /// Whether a prompt to save the scene should appear or not when attempting to save.<br/><br/>
34     /// False after a successful save and true if any important actions occur within a scene. >
35     /// </summary>
36     private bool displaySavePrompt = false;
37
38     // Enforces a singleton state pattern
39     private void Awake()
40     {
41         if (instance != null)
42         {
43             Destroy(this);
44             throw new Exception("EditorStructureManager instance already >
```

```
    established; terminating.");  
45        }  
46  
47        instance = this;  
48    }  
49  
50    // Begins the deserialization process as the scene is loaded.  
51    private void Start() { Deserialize(); }  
52  
53    /// <summary>  
54    /// Begins the serialization process by calling its corresponding  
55    /// coroutine.  
56    /// </summary>  
57    public void Serialize()  
58    {  
59        displaySavePrompt = false;  
60        StartCoroutine(SerializeCoroutine());  
61    }  
62  
63    /// <summary>  
64    /// Gathers all relevant information from the current scene and loads it  
65    /// into the current <see cref="EditorStructure"/> before saving it to the  
66    /// directory.  
67    /// </summary>  
68    public IEnumerator SerializeCoroutine()  
69    {  
70        // By skipping a frame, the UI loading screen is able to appear  
        // for more complex scenes.  
71        yield return null;  
72  
73        // Obtains the current editor structure to override  
74        int sceneIndex = MenuLogicManager.Instance.CurrentSceneIndex;  
75        EditorStructure editorStructure =  
76            MenuSetupManager.Instance.EditorStructures[sceneIndex];  
77  
78        List<bool> isPoweredInput = new List<bool>();  
79        List<CircuitIdentifier> circuitIdentifiers = new  
80            List<CircuitIdentifier>();  
81  
82        // Iterates through all circuits within the scene to assign all  
        // circuit identifiers and powered inputs.  
83        foreach (Circuit circuit in circuits)  
84        {  
85            circuitIdentifiers.Add(new CircuitIdentifier(circuit));  
86            isPoweredInput.Add(circuit.GetType() == typeof(InputGate) &&  
87                ((InputGate)circuit).Powered);  
88        }  
89  
90        // After obtaining all relevant values, override the editor
```

```
        structure
85         editorStructure.InGridMode =
86             Coordinates.Instance.CurrentSnappingMode ==
87                 Coordinates.SnappingMode.GRID;
88         editorStructure.IsPoweredInput = isPoweredInput;
89         editorStructure.Circuits = circuitIdentifiers;
90         editorStructure.Bookmarks = bookmarks;
91         editorStructure.BookmarkIDs = TaskbarManager.Instance.BookmarkIDs;
92         editorStructure.CameraLocation =
93             CameraMovement.Instance.PlayerCamera.transform.position;
94
95         // Saves the new editor structure to the directory and begins
96         // serializing all connections within the scene.
97         MenuSetupManager.Instance.UpdateEditorStructure(sceneIndex,
98             editorStructure);
99         MenuSetupManager.Instance.GenerateConnections(true, sceneIndex,
100            connections);
101
102         TaskbarManager.Instance.CloseMenu();
103     }
104
105     /// <summary>
106     /// Opens an editor scene based on a specified <see
107     /// cref="EditorStructure"/>, restoring components based on its save
108     /// files.
109     /// </summary>
110     public void Deserialize()
111     {
112         // Obtains the current editor structure to reference
113         int sceneIndex = MenuLogicManager.Instance.CurrentSceneIndex;
114         EditorStructure editorStructure =
115             MenuSetupManager.Instance.EditorStructures[sceneIndex];
116
117         // If this is the first time the scene has been opened, set
118         // default values, save to directory, and return.
119         if (MenuLogicManager.Instance.FirstOpen)
120         {
121             bool inGridMode = Coordinates.Instance.CurrentSnappingMode ==
122                 Coordinates.SnappingMode.GRID;
123             Vector3 cameraLocation =
124                 CameraMovement.Instance.PlayerCamera.transform.position;
125
126             editorStructure.InGridMode = inGridMode;
127             editorStructure.CameraLocation = cameraLocation;
128             TaskbarManager.Instance.RestoreCustomCircuits(); // Ensures
129                 all custom circuits still appear in the add menu
130             MenuSetupManager.Instance.UpdateEditorStructure(sceneIndex,
131                 editorStructure); // Saves to directory
132             return;
133         }
134     }
```

```
119     }
120
121     Coordinates.Instance.CurrentSnappingMode =
122         editorStructure.InGridMode ? Coordinates.SnappingMode.GRID :
123             Coordinates.SnappingMode.NONE;
124
125     CameraMovement.Instance.PlayerCamera.transform.position =
126         editorStructure.CameraLocation;
127
128     // Instantiates all circuits back to the scene
129     foreach (CircuitIdentifier circuitIdentifier in
130         editorStructure.Circuits)
131     {
132         circuits.Add(CircuitIdentifier.RestoreCircuit
133             (circuitIdentifier));
134     }
135
136     // Restores all connections back to the scene
137     MenuSetupManager.Instance.RestoreConnections(sceneIndex);
138
139     int index = 0;
140
141     // After all circuits have been established and connected, turn on any previously powered input gate.
142     foreach (bool isPoweredInput in editorStructure.IsPoweredInput)
143     {
144         if (isPoweredInput) ((InputGate)circuits[index]).Powered =
145             true;
146         index++;
147     }
148
149     // Lastly, all bookmarks and custom circuits are restored to their respective menus.
150     TaskbarManager.Instance.RestoreBookmarks
151         (editorStructure.Bookmarks, editorStructure.BookmarkIDs);
152     TaskbarManager.Instance.RestoreCustomCircuits();
153
154     // Getter methods
155     public static EditorStructureManager Instance { get { return
156         instance; } }
157
158     public bool DisplaySavePrompt { get { return displaySavePrompt; } set
159         { displaySavePrompt = value; } }
160
161     public List<Circuit> Circuits { get { return circuits; } }
162
163     public List<CircuitConnector.Connection> Connections { get { return
164         connections; } }
```

```
156     public List<int> Bookmarks { get { return bookmarks; } }  
157 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using TMPro;
4  using UnityEngine;
5
6  /// <summary>
7  /// IOAssigner is enabled in the editor scene after a custom circuit is      ↵
8  /// validated to assign input/output orders and labels.
9  /// </summary>
10 public class IOAssigner : MonoBehaviour
11 {
12     // Singleton state reference
13     private static IOAssigner instance;
14
15     /// <summary>
16     /// The type of text <seealso cref="hoverText"/> should display.<br/      ↵
17     /// ><br/>
18     /// <seealso cref="NONE"/>: prompts the user to hover on valid inputs/      ↵
19     /// outputs.<br/>
20     /// <seealso cref="INPUT"/>: currently hovered onto an input; display      ↵
21     /// its potential order.<br/>
22     /// <seealso cref="OUTPUT"/>: currently hovered onto an output;      ↵
23     /// display its potential order.
24     /// </summary>
25     private enum TextMode { NONE, INPUT, OUTPUT }
26
27     /// <summary>
28     /// Exits the IOAssigner phase; the circuit must be validated again to      ↵
29     /// reach this point.
30     /// </summary>
31     [SerializeField]
32     KeyCode exitKey;
33
34     /// <summary>
35     /// The material applied to all empty inputs that are active and not      ↵
36     /// being hovered on.
37     /// </summary>
38     [SerializeField]
39     Material emptyInputMaterial;
40
41     /// <summary>
42     /// The material applied to all empty outputs that are active and not      ↵
43     /// being hovered on.
44     /// </summary>
45     [SerializeField]
46     Material emptyOutputMaterial;
47
48     /// <summary>
49     /// The material applied to an empty input/output that is currently      ↵
50     ///
```

```
        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>
```

...y Project\Assets\Scripts\Editor Scripts\IOAssigner.cs 3

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

```
        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         {
```

...y Project\Assets\Scripts\Editor Scripts\IOAssigner.cs 5

```
163         output =
164             hitObject.GetComponent<CircuitVisualizer.OutputReference>().Output;
165
166             // If not within emptyOutputs, not valid.
167             if (!emptyOutputs.Contains(output)) return;
168
169             // Updates the interface and input/output material if the
170             // current hit object is not the hovered object from the last
171             // frame.
172             if (currentHover != hitObject)
173             {
174                 // If the last hit object was something, restore its
175                 // default material.
176                 if (currentHover != null)
177                     currentHover.GetComponent<MeshRenderer>().material =
178                     currentHover.layer == 9 ? emptyInputMaterial :
179                     emptyOutputMaterial;
180
181                 // Update material and set text based on GameObject layer
182                 // (i.e. input or output)
183                 hitObject.GetComponent<MeshRenderer>().material =
184                     hoveredMaterial;
185                 SetHoverText(hitObject.layer == 9 ? TextMode.INPUT :
186                             TextMode.OUTPUT);
187
188                 // Store as current hover object
189                 currentHover = hitObject;
190             }
191
192             // Also begins the labeling process if LMB is pressed
193             if (Input.GetMouseButtonDown(0)) OnIOPress(input, output);
194         }
195
196         // Restores to default values if the raycast was unsuccessful.
197         else if (currentHover != null)
198         {
199             currentHover.GetComponent<MeshRenderer>().material =
200                 currentHover.layer == 9 ? emptyInputMaterial :
201                 emptyOutputMaterial;
202             currentHover = null;
203             SetHoverText(TextMode.NONE);
204         }
205
206     }
207
208     /// <summary>
209     /// Enables IOAssigner and starts the labeling/ordering process.
210     /// </summary>
```

```
...y Project\Assets\Scripts\Editor Scripts\IOAssigner.cs 6
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
235     Unity.
236 }
237
238 /// <summary>
239 /// Cancels the label selection process for the current input or
240 /// output.
241 /// </summary>
242 public void CancelIOPress()
243 {
244     movementOverride = true; labelSelectionMode = false;
245     TaskbarManager.Instance.CloseMenu();
246     TaskbarManager.Instance.NullState();
247 }
248
249 /// <summary>
250 /// Begins the label selection process for the current input or
251 /// output.<br/><br/>
252 /// While there is both an input and output parameter, one of them
253 /// will always be null.
254 /// </summary>
255 /// <param name="input">The input to label.</param>
256 /// <param name="output">The output to label.</param>
257 private void OnIOPress(Circuit.Input input, Circuit.Output output)
258 {
259     movementOverride = false; labelSelectionMode = true;
260     currentInput = input; currentOutput = output;
261     TaskbarManager.Instance.CloseMenu();
262     TaskbarManager.Instance.OpenLabelMenu(input != null);
263
264     /// <summary>
265     /// Successfully completes the label selection process for the current
266     /// input or output.
267     /// </summary>
268     /// <param name="inputField">The text box to extract the label name
269     /// from.</param>
270     public void ConfirmIOPress(TMP_InputField inputField)
271     {
272         string labelName = inputField.text;
273
274         // Implies the current labeling was done for an input
275         if (currentInput != null)
276         {
277             // Moves the current input to the ordered list and out of the
278             // 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 ↵
284         // empty list.
285         outputCount++;
286         emptyOutputs.Remove(currentOutput);
287         orderedOutputs.Add(currentOutput);
288         outputLabels.Add(labelName);
289     }
290
291     inputField.text = "";
292     currentHover.GetComponent<MeshRenderer>().material =
293         currentHover.layer == 9 ?
294             CircuitVisualizer.Instance.InputMaterial :
295             CircuitVisualizer.Instance.OutputMaterial;
296     currentHover = null;
297     SetHoverText(TextMode.NONE);
298     labelSelectionMode = false;
299     TaskbarManager.Instance.CloseMenu();
300     TaskbarManager.Instance.NullState();
301
302     // All inputs/outputs have been labeled and/or ordered
303     if (inputCount == targetInputCount && outputCount ==
304         targetOutputCount)
305     {
306         hoverText.gameObject.SetActive(false);
307         enabled = false;
308         PreviewStructureManager.Instance.CreateCustomCircuit
309             (orderedInputs, orderedOutputs, inputLabels,
310             outputLabels); // Finally creates the custom circuit.
311     }
312
313     else movementOverride = true;
314 }
```

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

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5
6  /// <summary>
7  /// PreviewStructureManager controls the primary logic involved with      ↵
8  /// creating custom circuits within editor scenes.
9  /// </summary>
10 public class PreviewStructureManager : MonoBehaviour
11 {
12     // Singleton state reference
13     private static PreviewStructureManager instance;
14
15     /// <summary>
16     /// Denotes whether each internal circuit within the custom circuit      ↵
17     /// has been reached.<br/><br/>
18     /// Functionally, this list is used to run the depth-first search      ↵
19     /// (DFS) algorithm to determine whether all circuits in an editor scene      ↵
20     /// are connected.
21     /// </summary>
22     private bool[] reachedCircuits;
23
24     /// <summary>
25     /// List of inputs with no connections and all inputs respectively.
26     /// </summary>
27     private List<Circuit.Input> emptyInputs,
28         inputs;
29
30     /// <summary>
31     /// List of outputs with no connections and all outputs respectively.
32     /// </summary>
33     private List<Circuit.Output> emptyOutputs,
34         outputs;
35
36     /// <summary>
37     /// The number of circuits that have been reached.<br/><br/>
38     /// Functionally, circuitCount is utilized alongside <seealso      ↵
39     /// cref="reachedCircuits"/> to determine whether all circuits in an      ↵
40     /// editor scene are connected.
41     /// </summary>
42     private int circuitCount;
43
44     /// <summary>
45     /// The prospective name for the current custom circuit.<br/><br/>
46     /// If all validation tests succeed, it will be utilized as the name      ↵
47     /// of the custom circuit.
48     /// </summary>
49     private string currentName;
```

```
43
44     // Enforces a singleton state pattern
45     private void Awake()
46     {
47         if (instance != null)
48         {
49             Destroy(this);
50             throw new Exception("PreviewStructureManager instance already ↵
51                         established; terminating.");
52         }
53
54         instance = this;
55     }
56
57     /// <summary>
58     /// Calls the coroutine that begins the circuit creation process,      ↵
59     /// namely its validation tests.
60     /// </summary>
61     /// <param name="name">The prospective name of the custom circuit to    ↵
62     /// use.</param>
63     public void VerifyPreviewStructure(string name) { StartCoroutine      ↵
64         (VerifyPreviewStructureCoroutine(name)); }
65
66     /// <summary>
67     /// Performs a series of tests to verify the validity of a prospective ↵
68     /// custom circuit based on the current editor scene.
69     /// </summary>
70     /// <param name="name">The prospective name of the custom circuit to    ↵
71     /// use.</param>
72     private IEnumerator VerifyPreviewStructureCoroutine(string name)
73     {
74         // Skipping a frame ensures the UI dialog for verifying a custom    ↵
75         // circuit will show.
76         yield return null;
77
78         // Validation test #1: non-empty name
79         if (name == "")
80         {
81             TaskbarManager.Instance.CircuitSaveError("The custom circuit    ↵
82                         must not have an empty name.");
83             yield break;
84         }
85
86         // Validation test #2: unique name
87         foreach (PreviewStructure previewStructure in
88             MenuSetupManager.Instance.PreviewStructures)
89         {
90             if (previewStructure.Name == name)
91             {
92                 yield break;
93             }
94         }
95     }
96 }
```

```
...ets\Scripts\Editor Scripts\PreviewStructureManager.cs 3
83         TaskbarManager.Instance.CircuitSaveError("The custom      ↵
84             circuit must have a unique name.");
85     }
86 }
87
88 // Validation test #3: >= 1 circuits
89 if (EditorStructureManager.Instance.Circuits.Count == 0)
90 {
91     TaskbarManager.Instance.CircuitSaveError("The custom circuit      ↵
92         must consist of (1) or more circuits.");
93     yield break;
94 }
95
96 // Validation test #4: no input/display gates
97 foreach (Circuit circuit in
98     EditorStructureManager.Instance.Circuits)
99 {
100     Type type = circuit.GetType();
101
102     if (type == typeof(InputGate) || type == typeof(Display))
103     {
104         TaskbarManager.Instance.CircuitSaveError("The custom      ↵
105             circuit must not consist of any input gates or      ↵
106             displays.");
107         yield break;
108     }
109 }
110
111 // Validation test #5: all circuits are connected
112 reachedCircuits = new bool      ↵
113     [EditorStructureManager.Instance.Circuits.Count];
114 emptyInputs = new List<Circuit.Input>(); inputs = new      ↵
115     List<Circuit.Input>();
116 emptyOutputs = new List<Circuit.Output>(); outputs = new      ↵
117     List<Circuit.Output>();
118 circuitCount = 0;
119 CircuitConnectionTest(EditorStructureManager.Instance.Circuits      ↵
120     [0]); // Begins the DFS algorithm
121
122 if (circuitCount != reachedCircuits.Length)
123 {
124     TaskbarManager.Instance.CircuitSaveError("The custom circuit      ↵
125         must be entirely connected.");
126     yield break;
127 }
128
129 // Validation test #6: >= 1 empty outputs
130 if (emptyOutputs.Count == 0)
```

```
122         {
123             TaskbarManager.Instance.CircuitSaveError("The custom circuit      ↵
124             must have (1) or more empty outputs.");
125         }
126
127         /// All validation tests completed ///
128
129         currentName = name;
130         TaskbarManager.Instance.CloseMenu();
131         TaskbarManager.Instance.NullState();
132
133         // Begins the process in which the user assigns the order and      ↵
134             labels of all empty inputs and outputs.                         ↵
135         IOAssigner.Instance.Initialize(emptyInputs, emptyOutputs);
136
137     /// <summary>
138     /// Starts the coroutine involved in finally creating a custom      ↵
139             circuit.<br/><br/>
140     /// This method is specifically called by <see cref="IOAssigner"/>      ↵
141             after all empty inputs and outputs have been ordered by the user (as      ↵
142             well as any respective labelling).
143
144     /// </summary>
145     /// <param name="orderedInputs"></param>
146     /// <param name="orderedOutputs"></param>
147     /// <param name="inputLabels"></param>
148     /// <param name="outputLabels"></param>
149     public void CreateCustomCircuit(List<Circuit.Input> orderedInputs,      ↵
150         List<Circuit.Output> orderedOutputs, List<string> inputLabels,      ↵
151         List<string> outputLabels)
152     {
153         StartCoroutine(CreatePreviewStructure(orderedInputs,      ↵
154             orderedOutputs, inputLabels, outputLabels));
155     }
156
157     /// <summary>
158     /// Serializes a custom circuit as well as its corresponding preview      ↵
159             structure.
160     /// </summary>
161     /// <param name="orderedInputs">The list of empty inputs, ordered.</      ↵
162             param>
163     /// <param name="orderedOutputs">The list of empty outputs, ordered.</      ↵
164             param>
165     /// <param name="inputLabels">Labels associated with each ordered      ↵
166             input.</param>
167     /// <param name="outputLabels">Labels associated with each ordered      ↵
168             output.</param>
169     private IEnumerator CreatePreviewStructure(List<Circuit.Input>      ↵
```

...ets\Scripts\Editor Scripts\PreviewStructureManager.cs 5

```
    orderedInputs, List<Circuit.Output> orderedOutputs, List<string>
    inputLabels, List<string> outputLabels)
158  {
159      TaskbarManager.Instance.OnSuccessfulPreviewVerification();
160
161      // Skipping a frame ensures the UI dialog for creating a custom
162      // circuit will show.
163      yield return null;
164
165      List<CircuitIdentifier> circuitIdentifiers = new
166          List<CircuitIdentifier>();
167      List<int> inputOrders = new List<int>(), outputOrders = new
168          List<int>();
169      PreviewStructure previewStructure = new PreviewStructure
170          (currentName);
171
172      // Serializes each circuit by instantiating CircuitIdentifier
173      // references.
174      foreach (Circuit circuit in
175          EditorStructureManager.Instance.Circuits)
176      {
177          circuitIdentifiers.Add(new CircuitIdentifier(circuit));
178
179          foreach (Circuit.Input input in circuit.Inputs) { inputs.Add
179              (input); inputOrders.Add(orderedInputs.IndexOf(input)); }
180
181          foreach (Circuit.Output output in circuit.Outputs)
182              { outputs.Add(output); outputOrders.Add
183                  (orderedOutputs.IndexOf(output)); }
184      }
185
186      previewStructure.Circuits = circuitIdentifiers;
187      previewStructure.ID = UniqueID; // Assigns a unique ID to the
188      // preview structure.
189      previewStructure.InputOrders = inputOrders;
190      previewStructure.OutputOrders = outputOrders;
191      previewStructure.InputLabels = inputLabels;
192      previewStructure.OutputLabels = outputLabels;
193      previewStructure.CameraLocation =
194          CameraMovement.Instance.PlayerCamera.transform.position;
195
196      List<InternalConnection> internalConnections = new
197          List<InternalConnection>();
198
199      // Serializes each connection by assigning index values to each
200      // input/output pair within an InternalConnection instance.
201      foreach (CircuitConnector.Connection connection in
202          EditorStructureManager.Instance.Connections)
203      {
```

```
191         internalConnections.Add(new InternalConnection(
192             inputs.IndexOf(connection.Input),
193             outputs.IndexOf(connection.Output)
194         ));
195     }
196
197     previewStructure.Connections = internalConnections;
198
199     // Adds preview structure and its connections to the save
199     // directory and add menu.                                     ↵
200     MenuSetupManager.Instance.PreviewStructures.Add(previewStructure);
201     MenuSetupManager.Instance.GenerateConnections(false,
201         previewStructure.ID,
201         EditorStructureManager.Instance.Connections);
202     MenuSetupManager.Instance.UpdatePreviewStructure
202     (previewStructure);
203     TaskbarManager.Instance.AddCustomCircuitPanel(previewStructure.ID, ↵
203         false);
204     TaskbarManager.Instance.OnSuccessfulPreviewStructure();
205 }
206
207 /// <summary>
208 /// Performs a depth-first search starting at the first placed circuit
208 /// to determine whether the scene represents a complete graph. <br/>
209 /// At the same time, any circuit input or output without a connection
209 /// is stored for the next test.
210 /// </summary>
211 private void CircuitConnectionTest(Circuit currentCircuit)
212 {
213     while (currentCircuit.customCircuit != null)
214     {
215         currentCircuit = currentCircuit.customCircuit;
216     }
217
218     int index = EditorStructureManager.Instance.Circuits.IndexOf
218     (currentCircuit);
219
220     if (reachedCircuits[index]) return;
221
222     reachedCircuits[index] = true;
223     circuitCount++;
224
225     foreach (Circuit.Input input in currentCircuit.Inputs)
226     {
227         if (input.ParentOutput == null) { emptyInputs.Add(input);
227             continue; }
228
229         CircuitConnectionTest(input.ParentOutput.ParentCircuit);
230     }
```

```
231
232     foreach (Circuit.Output output in currentCircuit.Outputs)
233     {
234         if (output.ChildInputs.Count == 0) { emptyOutputs.Add(output); ↵
235             continue; }
236
237         foreach (Circuit.Input input in output.ChildInputs)
238         {
239             CircuitConnectionTest(input.ParentCircuit);
240         }
241     }
242
243 /// <summary>
244 /// Returns a new unique ID for a new preview structure.<br/><br/>
245 /// A unique ID starts from 0 and increments onward.
246 /// </summary>
247 private int UniqueID
248 {
249     get
250     {
251         int currentID = 0;
252
253         // Keeps incrementing the current ID until it is unique
254         // This system ensures that if an ID that is not the largest ↵
255             // is removed, it will be recycled in future custom circuit ↵
256             // creations.
257         while (true)
258         {
259             if (!
260                 MenuSetupManager.Instance.PreviewStructureIDs.Contains ↵
261                     (currentID))
262             {
263                 MenuSetupManager.Instance.PreviewStructureIDs.Add ↵
264                     (currentID);
265                 return currentID;
266             }
267
268             currentID++;
269         }
270     }
271
272     // Getter method
273     public static PreviewStructureManager Instance { get { return ↵
274         instance; } }
```

```
1  using System;
2  using UnityEngine;
3
4  /// <summary>
5  /// BackgroundParallax captures any mouse movement and proportionally moves ↵
6  /// an assigned background to emulate a parallax effect.
7  /// </summary>
8  public class BackgroundParallax : MonoBehaviour
9  {
10     // Singleton state reference
11     private static BackgroundParallax instance;
12
13     /// <summary>
14     /// Controls how much the captured mouse movement alters the ↵
15     /// background.
16     /// </summary>
17     [SerializeField]
18     float parallaxStrength;
19
20     /// <summary>
21     /// The background that the parallax is applied on.
22     /// </summary>
23     [SerializeField]
24     RectTransform backgroundTransform;
25
26     /// <summary>
27     /// Stores prior mouse positions to calculate the delta movement ↵
28     /// between frames.
29     /// </summary>
30     private Vector2 mousePos;
31
32     // Enforces a singleton state pattern
33     private void Awake()
34     {
35         if (instance != null)
36         {
37             Destroy(this);
38             throw new Exception("BackgroundParallax instance already ↵
39             established; terminating.");
40         }
41
42         instance = this;
43     }
44
45     private void Start() { mousePos = Input.mousePosition; }
46
47     private void Update()
48     {
49         // Captures the difference in mouse position between frames,
```

```
proportionally moving the background.  
46     Vector2 prevMousePos = mousePos, mouseDelta;  
47  
48     mousePos = Input.mousePosition;  
49     mouseDelta = (prevMousePos - mousePos) * parallaxStrength;  
50     backgroundTransform.offsetMin += mouseDelta;  
51 }  
52  
53 // Getter method  
54 public static BackgroundParallax Instance { get { return instance; } }  
55 }
```

```
1  using UnityEngine;
2  using UnityEngine.EventSystems;
3
4  ///<summary>
5  ///<summary>ButtonRightClick is assigned to the save slot buttons within the menu, ↵
6  ///</summary>enabling for the deletion of a scene.
7  ///</summary>
8  public class ButtonRightClick : MonoBehaviour, IPointerClickHandler
9  {
10     ///<summary>
11     ///<summary>Stores which editor scene this instance is assigned to.
12     ///</summary>
13     [SerializeField] int saveIndex;
14
15     ///<summary>
16     ///<summary>Begins the scene deletion process if the captured input was a right ↵
17     ///</summary>click.
18     public void OnPointerClick(PointerEventData eventData)
19     {
20         if (eventData.button == PointerEventData.InputButton.Right)      ↵
21             MenuInterfaceManager.Instance.BeginSceneDeletion(saveIndex);
22     }
23 }
```

```
1  using UnityEngine;
2  using UnityEngine.UI;
3
4  ///<summary>
5  ///<summary>CustomCircuitButtons is utilized to add delegate listeners to its
6  ///</summary>specified custom circuit within the menu for viewing and deleting.
7  ///</summary>
8  public class CustomCircuitButtons : MonoBehaviour
9  {
10     ///<summary>
11     ///<summary>The in-scene buttons pertaining to deletion and viewing a custom
12     ///</summary>
13     [SerializeField]
14     Button deleteButton, viewButton;
15
16     // Getter methods
17     public Button DeleteButton { get { return deleteButton; } }
18     public Button ViewButton { get { return viewButton; } }
19 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using TMPro;
4  using UnityEngine;
5
6  /// <summary>
7  /// MenuInterfaceManager handles all UI interactions and transitions
8  /// within the menu scene.
9  /// </summary>
10 public class MenuInterfaceManager : MonoBehaviour
11 {
12     // Singleton state reference
13     private static MenuInterfaceManager instance;
14
15     /// <summary>
16     /// The colors of save slots when uncreated and created. respectively.
17     /// </summary>
18     [SerializeField]
19     Color defaultColor,
20         saveColor;
21
22     /// <summary>
23     /// How much the height of <seealso cref="deleteErrorTransform"/>
24     /// should increase for each additional error message.
25     /// </summary>
26     [SerializeField] float deleteErrorMessageSize;
27
28     /// <summary>
29     /// Attempts to exit the <seealso cref="currentInterface"/> in
30     /// use.<br/><br/>
31     /// For some interfaces, this will not instantly occur (such as going
32     /// back to the options menu from the guide).
33     /// </summary>
34     [SerializeField] KeyCode cancelKey;
35
36     /// <summary>
37     /// Utilized to add and reference custom circuits from the directory.
38     /// </summary>
39     [SerializeField]
40     GameObject customCircuitPanel,
41         customCircuitPrefab;
42
43     /// <summary>
44     /// Set to visible when an interface is opened.<br/><br/>
45     /// Within the scene, this should be a semi-transparent background
46     /// overlayed onto everything except the currently opened interface.
47     /// </summary>
48     [SerializeField]
49     GameObject transparentBackground,
```

```
45     transparentBackgroundUpper; // Utilized for multi-level interfaces ↵
        such as the options interface.
46
47     [Space(10)]
48     /// <summary>
49     /// List of all interfaces within the scene.
50     /// </summary>
51     [SerializeField]
52     GameObject customCircuitsInterface, // Displays all currently open ↵
        custom circuits and allows the user to view and/or delete them; ↵
        opened from the options menu.
53     deleteErrorInterface, // The error log(s) displayed when a custom ↵
        circuit cannot be deleted for any reason(s).
54     guideInterface, // Displays the guide prefab; opened from the ↵
        options menu.
55     optionSelectionInterface, // Allows the user to activate ↵
        customCircuitsInterface as well as guideInterface.
56     optionsInterface, // The parent of optionSelectionInterface ↵
        indicating that the options menu is open.
57     sceneDeletionInterface, // Prompts the user to confirm deleting an ↵
        editor structure (save slot).
58     sceneNameInterface; // Prompts the user to compose a name for an ↵
        editor structure (save slot).
59
60     [Space(10)]
61     /// <summary>
62     /// Utilized to expand the vertical height of the error interface by ↵
        factors of <seealso cref="deleteErrorMessageSize"/>.
63     /// </summary>
64     [SerializeField]
65     RectTransform deleteErrorTransform;
66
67     /// <summary>
68     /// The text components for the delete and scene creation interfaces ↵
        utilized to display the cause(s) of error.
69     /// </summary>
70     [SerializeField]
71     TextMeshProUGUI deleteErrorText,
72     sceneNameError;
73
74     /// <summary>
75     /// The input field within <seealso cref="sceneNameInterface"/> that ↵
        the user uses to compose a scene name.
76     /// </summary>
77     [SerializeField]
78     TMP_InputField sceneNameInputField;
79
80     /// <summary>
81     /// Text components of all 3 save slots within the menu used for ↵
```

```
    setting and deleting their names.  
82     /// </summary>  
83     [SerializeField]  
84     TextMeshProUGUI save1,  
85         save2,  
86         save3;  
87  
88     /// <summary>  
89     /// The current UI interface in use.  
90     /// </summary>  
91     private GameObject currentInterface;  
92  
93     /// <summary>  
94     /// Index of the current <see cref="EditorStructure"/> loaded into the ↵  
         scene.  
95     /// </summary>  
96     private int currentSceneIndex = -1;  
97  
98     // Enforces a singleton state pattern  
99     private void Awake()  
100    {  
101        if (instance != null)  
102        {  
103            Destroy(this);  
104            throw new Exception("MenuInterfaceManager instance already      ↵  
                established; terminating.");  
105        }  
106  
107        instance = this;  
108    }  
109  
110    // Loads in all serialized editor scenes and preview structures.  
111    private void Start()  
112    {  
113        UpdateInterface();  
114        AddCustomBookmarks();  
115        CursorManager.SetMouseTexture(true);  
116        enabled = false;  
117    }  
118  
119    private void Update()  
120    {  
121        // Default exit controls for all interfaces except the options      ↵  
            interface.  
122        if (currentInterface != optionsInterface)  
        {  
            if (Input.GetKeyDown(cancelKey) || Input.GetMouseButtonUp(1)) CancelCurrentSubmission();  
        }  
125    }
```

```
126
127      // Otherwise, the options interface is opened and should utilize a >
128      // different transition scheme.
129
130      else if (Input.GetKeyDown(cancelKey) || Input.GetMouseButtonUp(1))
131      {
132          // If at the root, exit the interface.
133          if (optionSelectionInterface.activeSelf)
134              CancelCurrentSubmission();
135
136          // If within the custom circuits interface whilst the delete
137          // error interface is not open, return to the root.
138          else if (customCircuitsInterface.activeSelf && !
139                  deleteErrorInterface.activeSelf)
140          {
141              customCircuitsInterface.SetActive(false);
142              optionSelectionInterface.SetActive(true);
143          }
144
145          // If within the guide interface, return to the root.
146          else if (guideInterface.activeSelf)
147          {
148              guideInterface.SetActive(false);
149              optionSelectionInterface.SetActive(true);
150
151          // If within the guide delete error interface, re-adjust
152          // background layers to make the custom circuit interface
153          // accessible.
154          else if (deleteErrorInterface.activeSelf)
155          {
156              transparentBackgroundUpper.SetActive(false);
157              transparentBackground.SetActive(true);
158              deleteErrorInterface.SetActive(false);
159          }
160      }
161
162      /// <summary>
163      /// Opens an editor scene; called by pressing one of the valid save
164      /// buttons.
165      /// </summary>
166      /// <param name="sceneIndex"></param>
167      public void OpenScene(int sceneIndex)
168      {
169          MenuLogicManager.Instance.OpenScene(sceneIndex);
170
171      /// <summary>
172      /// Instantiates a <seealso cref="customCircuitPrefab"/> button for
173      // each preview structure.
```

```
165     /// </summary>
166     private void AddCustomBookmarks()
167     {
168         List<PreviewStructure> previewStructures =
169             MenuSetupManager.Instance.PreviewStructures;
170
171         foreach (PreviewStructure previewStructure in previewStructures)
172         {
173             GameObject current = Instantiate(customCircuitPrefab,
174                 customCircuitPanel.transform);
175
176             // Adds the relevant listeners to ensure the custom circuit
177             // can be deleted and viewed by their respective buttons.
178             current.GetComponent<CustomCircuitButtons>
179                 ().DeleteButton.onClick.AddListener(delegate { DeletePreview
180                 (current, previewStructure); });
181             current.GetComponent<CustomCircuitButtons>
182                 ().ViewButton.onClick.AddListener(delegate { PreviewScene
183                 (previewStructure); });
184         }
185     }
186
187     /// <summary>
188     /// Deletes a preview structure from the game and directory given it
189     /// is not in use.
190     /// </summary>
191     /// <param name="button">The button assigned this method by a
192     /// delegate.</param>
193     /// <param name="previewStructure">The preview structure to attempt
194     /// deletion on.</param>
195     public void DeletePreview(GameObject button, PreviewStructure
196     previewStructure)
197     {
198         // Obtains a list of all error messages in an attempt to delete
199         // the preview structure
200         List<string> errorMessages =
201             MenuLogicManager.CanDeleteCustomCircuit(previewStructure);
202
203         // If there are no errors, proceed with deletion.
204         if (errorMessages.Count == 0)
205         {
206             MenuSetupManager.Instance.DeletePreviewStructure
207                 (previewStructure);
208             Destroy(button);
209         }
210     }
```

```
199         // Otherwise, open the delete error interface and display errors.
200         else
201     {
202         transparentBackgroundUpper.SetActive(true);
203         transparentBackground.SetActive(false);
204         deleteErrorInterface.SetActive(true);
205         deleteErrorTransform.sizeDelta = new Vector2
206             (deleteErrorTransform.sizeDelta.x, deleteErrorMessageSize * errorMessages.Count);
207         deleteErrorText.text = "";
208
209         int index = 0;
210
211         // Adds each error
212         foreach (string errorMessage in errorMessages)
213         {
214             index++;
215             deleteErrorText.text += "- " + errorMessage + (index != errorMessages.Count ? "\n\n" : "");
216         }
217     }
218
219 /// <summary>
220 /// Referenced when the user acknowledges and closes the delete error interface; called by pressing an in-scene button.
221 /// </summary>
222 public void OnCircuitDeleteErrorConfirm()
223 {
224     transparentBackgroundUpper.SetActive(false);
225     transparentBackground.SetActive(true);
226     deleteErrorInterface.SetActive(false);
227 }
228
229 /// <summary>
230 /// Opens a preview structure whose internal components the user wishes to inspect.<br/><br/>
231 /// This method is assigned as a listener to the delete button within each instantiated <seealso cref="customCircuitPrefab"/>.
232 /// </summary>
233 /// <param name="previewStructure">The preview structure to open.</param>
234 public void PreviewScene(PreviewStructure previewStructure)
235     { MenuLogicManager.Instance.OpenPreview(previewStructure); }
236
237 /// <summary>
238 /// Sets the names of all save slots corresponding to serialized editor structures.
239 /// </summary>
```

```
239     public void UpdateInterface()
240     {
241         EditorStructure[] editorStructures =
242             MenuSetupManager.Instance.EditorStructures;
243
244         if (editorStructures[0] != null) { save1.text = editorStructures
245             [0].Name; save1.color = saveColor; }
246
247         if (editorStructures[1] != null) { save2.text = editorStructures
248             [1].Name; save2.color = saveColor; }
249
250         /// <summary>
251         /// Opens <seealso cref="sceneDeletionInterface"/> prompting a user to
252             acknowledge their action to delete a editor structure (save slot).
253         /// </summary>
254         /// <param name="sceneIndex">The index of the prospective editor
255             structure to delete</param>
256         public void BeginSceneDeletion(int sceneIndex)
257         {
258             if (MenuSetupManager.Instance.EditorStructures[sceneIndex] ==
259                 null) return;
260
261             currentSceneIndex = sceneIndex;
262             BeginInterface(sceneDeletionInterface);
263
264             /// <summary>
265             /// Opens <seealso cref="sceneNameInterface"/> prompting a user to
266                 compose a name to create an editor structure.
267             /// </summary>
268             /// <param name="sceneIndex">The index of the prospective editor
269                 structure to create.</param>
270             public void BeginSceneNameSubmission(int sceneIndex)
271             {
272                 currentSceneIndex = sceneIndex;
273                 BeginInterface(sceneNameInterface);
274
275             /// <summary>
276             /// Opens the options interface; called by pressing an in-scene
277                 button.
278             /// </summary>
279             public void OpenOptionsInterface()
280             {
281                 BeginInterface(optionsInterface);
```

...t\Assets\Scripts\Menu Scripts\MenuInterfaceManager.cs 8

```
278         optionSelectionInterface.SetActive(true);
279         customCircuitsInterface.SetActive(false);
280         guideInterface.SetActive(false);
281     }
282 
283     /// <summary>
284     /// Opens the guide interface; called by pressing an in-scene button.
285     /// </summary>
286     public void OpenGuide()
287     {
288         optionSelectionInterface.SetActive(false);
289         guideInterface.SetActive(true);
290     }
291 
292     /// <summary>
293     /// Opens the custom circuit interface; called by pressing an in-scene button.
294     /// </summary>
295     public void OpenCustomCircuits()
296     {
297         optionSelectionInterface.SetActive(false);
298         customCircuitsInterface.SetActive(true);
299     }
300 
301     /// <summary>
302     /// Restores the text of a save slot to its default values after a success editor scene deletion.
303     /// </summary>
304     public void SceneDeleteSubmission()
305     {
306         MenuSetupManager.Instance.DeleteEditorStructure
307             (currentSceneIndex);
308         TextMeshProUGUI currentText;
309 
310         if (currentSceneIndex == 0)
311         {
312             currentText = save1;
313         }
314 
315         else if (currentSceneIndex == 1)
316         {
317             currentText = save2;
318         }
319 
320         else
321         {
322             currentText = save3;
323         }
324     }
325 }
```

```
322         CancelCurrentSubmission();
323         currentText.text = "new save";
324         currentText.color = defaultColor;
325     }
326
327     /// <summary>
328     /// Checks the name in <seealso cref="sceneNameInputField"/> and
329     /// creates an editor scene if it is valid.      ↵
330     /// </summary>
331     public void SceneNameSubmission()
332     {
333         string submission = sceneNameInputField.text.ToLower().Trim();
334
335         // Cannot be empty
336         if (submission == string.Empty) sceneNameError.text = "scene name ↵
337             must be non-empty";
338
339         // Must be unique
340         else if (CurrentSceneNames.Contains(submission))      ↵
341             sceneNameError.text = "scene name must be unique";
342
343         // Creates and opens the editor structure.
344         else MenuLogicManager.Instance.CreateScene(currentSceneIndex,      ↵
345             submission);
346
347     /// <summary>
348     /// Closes the current interface in use.
349     /// </summary>
350     public void CancelCurrentSubmission()
351     {
352         if (currentInterface == sceneNameInterface)      ↵
353             sceneNameInputField.text = sceneNameError.text = "";
354
355         BackgroundParallax.Instance.enabled = true;
356         ToggleCurrentInterface(false);
357         currentSceneIndex = -1;
358         currentInterface = null;
359         enabled = false;
360     }
361
362     // Exits the game; called by pressing an in-scene button.
363     public void Quit() { Application.Quit(); }
364
365     /// <summary>
366     /// Opens an interface.
367     /// </summary>
368     /// <param name="newInterface">The interface to open.</param>
```

```
366     private void BeginInterface(GameObject newInterface)
367     {
368         if (currentInterface != null) return;
369
370         BackgroundParallax.Instance.enabled = false;
371         currentInterface = newInterface;
372         ToggleCurrentInterface(true);
373         enabled = true;
374     }
375
376     /// <summary>
377     /// Sets the visibility of the current interface.
378     /// </summary>
379     /// <param name="visible">Whether the current interface should be
380     /// visible.</param>
381     private void ToggleCurrentInterface(bool visible)
382     {
383         currentInterface.SetActive(visible);
384         transparentBackground.SetActive(visible);
385     }
386
387     // Getter methods
388     public static MenuInterfaceManager Instance { get { return
389         instance; } }
390
391     /// <summary>
392     /// Returns the named list of all editor structures.
393     /// </summary>
394     private List<string> CurrentSceneNames
395     {
396         get
397         {
398             EditorStructure[] editorStructures =
399                 MenuSetupManager.Instance.EditorStructures;
400             List<string> currentSceneNames = new List<string>();
401
402             foreach (EditorStructure editorStructure in editorStructures)
403                 if (editorStructure != null) currentSceneNames.Add
404                     (editorStructure.Name);
405
406             return currentSceneNames;
407         }
408     }
409 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  /// <summary>
7  /// MenuLogicManager handles all scene transitions as well as validation    ↵
8  /// tests for deleting custom circuits.
9  /// </summary>
10 public class MenuLogicManager : MonoBehaviour
11 {
12     // Singleton state reference
13     private static MenuLogicManager instance;
14
15     /// <summary>
16     /// Whether the current editor scene has just been created or not.<br/> ↵
17     /// ><br/>
18     /// If it is the first time, this value indicates that default values    ↵
19     /// must be initialized in the editor scene.
20     /// </summary>
21     private bool firstOpen;
22
23     /// <summary>
24     /// The index of the current editor scene to open.
25     /// </summary>
26     private int currentSceneIndex;
27
28     /// <summary>
29     /// The current preview structure in the preview scene.
30     /// </summary>
31     private PreviewStructure currentPreviewStructure;
32
33     /// <summary>
34     /// Utilized within <seealso cref="TraversalTest(PreviewStructure,    ↵
35     /// PreviewStructure)"/> to sort all preview structures into these    ↵
36     /// respective lists.<br/>
37     /// If a circuit is invalid, it means that it is being utilized    ↵
38     /// somewhere and cannot be deleted. A valid circuit can be deleted.
39     /// </summary>
40     private List<PreviewStructure> invalidCircuits, validCircuits;
41
42     // Enforces a singleton state pattern
43     private void Awake()
44     {
45         if (instance != null)
46         {
47             Destroy(this);
48             throw new Exception("MenuLogicManager instance already    ↵
49             established; terminating.");
50     }
```

```
43         }
44
45         instance = this;
46     }
47
48     /// <summary>
49     /// Attempts to open an editor scene by index.
50     /// </summary>
51     /// <param name="sceneIndex">Index of the editor structure to open.</param>
52     public void OpenScene(int sceneIndex)
53     {
54         bool isCreated = MenuSetupManager.Instance.EditorStructures[sceneIndex] != null;
55
56         firstOpen = !isCreated;
57
58         // Opens the scene only if it has not been just created; otherwise completes the setup process.
59         if (isCreated) ImportScene(sceneIndex); else
56             MenuInterfaceManager.Instance.BeginSceneNameSubmission
57             (sceneIndex);
58     }
59
60
61     /// <summary>
62     /// Creates an editor scene.
63     /// </summary>
64     /// <param name="sceneIndex">Index of the editor structure to create.</param>
65     /// <param name="name">Name of the editor structure to create.</param>
66     public void CreateScene(int sceneIndex, string name)
67     {
68         MenuSetupManager.Instance.EditorStructures[sceneIndex] = new
69             EditorStructure(name);
70         ImportScene(sceneIndex);
71     }
72
73     /// <summary>
74     /// Opens a preview structure.
75     /// </summary>
76     /// <param name="previewStructure">The preview structure to open.</param>
77     public void OpenPreview(PreviewStructure previewStructure)
78     {
79         currentPreviewStructure = previewStructure;
80         SceneManager.LoadScene(2);
81     }
82
83     /// <summary>
```

```
84     /// Opens a scene that has already been created beforehand.
85     /// </summary>
86     /// <param name="sceneIndex"></param>
87     private void ImportScene(int sceneIndex)
88     {
89         currentSceneIndex = sceneIndex;
90         SceneManager.LoadScene(1);
91     }
92
93     /// <summary>
94     /// Runs several validation tests that determine whether a custom      ↗
95     /// circuit can be deleted.<br/><br/>.
96     /// If there are no error messages returned, then the custom circuit      ↗
97     /// can be deleted.
98     /// </summary>
99     /// <param name="_previewStructure">The prospect custom circuit/      ↗
100    /// preview structure to delete.</param>
101    /// <returns></returns>
102    public static List<string> CanDeleteCustomCircuit(PreviewStructure      ↗
103        _previewStructure)
104    {
105        List<string> errorMessages = new List<string>();
106        instance.invalidCircuits = new List<PreviewStructure>();      ↗
107        instance.validCircuits = new List<PreviewStructure>();
108
109        // Marks each preview structure as either valid or invalid
110        foreach (PreviewStructure previewStructure in      ↗
111            MenuSetupManager.Instance.PreviewStructures)      ↗
112            instance.TraversalTest(previewStructure, _previewStructure);
113
114        // Tests #1/#2: not placed or part of a bookmark directly (top-      ↗
115        // most custom circuit) or indirectly (inside of another custom      ↗
116        // circuit) within an editor scene.
117        foreach (EditorStructure editorStructure in      ↗
118            MenuSetupManager.Instance.EditorStructures)
119        {
120            if (editorStructure == null) continue;
121
122            // Placed circuit test
123            foreach (CircuitIdentifier circuitIdentifier in      ↗
124                editorStructure.Circuits)
125            {
126                if (circuitIdentifier.previewStructureID == -1) continue;
127
128                PreviewStructure previewStructure =      ↗
129                    MenuSetupManager.Instance.PreviewStructures      ↗
130                    [MenuSetupManager.Instance.PreviewStructureIDs.IndexOf      ↗
131                        (circuitIdentifier.previewStructureID)];
```

```
...object\Assets\Scripts\Menu Scripts\MenuLogicManager.cs 4
119         if (instance.invalidCircuits.Contains(previewStructure))    ↗
120             { errorMessages.Add("Circuit is directly and/or      ↗
121               indirectly referenced by a placed circuit in an editor      ↗
122                 scene"); break; }
123
124     }
125     if (customCircuitID == -1) continue;
126
127     PreviewStructure previewStructure =      ↗
128         MenuSetupManager.Instance.PreviewStructures      ↗
129         [MenuSetupManager.Instance.PreviewStructureIDs.IndexOf      ↗
130           (customCircuitID)];
131
132     if (instance.invalidCircuits.Contains(previewStructure))    ↗
133         { errorMessages.Add("Circuit is directly and/or      ↗
134           indirectly referenced by a bookmark in an editor      ↗
135             scene"); break; }
136   }
137
138 // Test #3: not directly (top-most custom circuit) or indirectly      ↗
139 // (inside of another custom circuit) within a custom circuit.      ↗
140 foreach (PreviewStructure previewStructure in      ↗
141   MenuSetupManager.Instance.PreviewStructures)
142 {
143     if (previewStructure == _previewStructure) continue;
144
145     if (instance.invalidCircuits.Contains(previewStructure))    ↗
146         { errorMessages.Add("Circuit is directly and/or indirectly      ↗
147           referenced by another custom circuit"); break; }
148
149     // Returns the list of all obtained error messages, if any.
150     return errorMessages;
151   }
152
153 /// <summary>
154 /// Sorts the current preview structure as invalid (contains the      ↗
155 // target circuit) or valid by running a modified depth-first      ↗
156 // search.<br/><br/>
157 /// Essentially, the user inputs a starting preview structure. Until a      ↗
158 // starting circuit (e.g. AND) is reached or there is nothing else to      ↗
159 // explore, this method:<br/>
160 /// - Recursively calls this method on all of its circuit components      ↗
161 // that are custom circuits granted they have not already been sorted      ↗
162 // into valid or invalid circuits.
```

...object\Assets\Scripts\Menu Scripts\MenuLogicManager.cs 5

```
149     /// - Adds the current circuit as an invalid preview structure if any    ↵
      of its children contained the target custom circuit, and    ↵
      returns.<br/>
150     /// - Adds the current circuit as a valid preview structure if none of    ↵
      its child custom circuits contained the target custom circuit.
151     /// </summary>
152     /// <param name="current"></param>
153     /// <param name="target"></param>
154     private void TraversalTest(PreviewStructure current, PreviewStructure    ↵
      target)
155     {
156         // If the target preview structure was recursively called by    ↵
          TraversalTest, then add it.
157         if (current == target) { if (!invalidCircuits.Contains(current))    ↵
              { invalidCircuits.Add(current); } return; }
158
159         // If already designated as an invalid/valid circuit, return.
160         else if (invalidCircuits.Contains(current) ||    ↵
              validCircuits.Contains(current)) return;
161
162         // Traverse through each custom circuit and recursively call this    ↵
          method.
163         // Once done, check to see if the target preview structure was    ↵
          detected. If so, then also add this preview structure.
164         foreach (CircuitIdentifier circuitIdentifier in current.Circuits)
165         {
166             if (circuitIdentifier.previewStructureID != -1)
167             {
168                 PreviewStructure previewStructure =    ↵
                     MenuSetupManager.Instance.PreviewStructures    ↵
                     [MenuSetupManager.Instance.PreviewStructureIDs.IndexOf    ↵
                     (circuitIdentifier.previewStructureID)];
169
170                 TraversalTest(previewStructure, target);
171
172                 if (invalidCircuits.Contains(previewStructure))    ↵
                     { invalidCircuits.Add(current); return; }
173             }
174         }
175
176         // If not invalid, then must be valid.
177         validCircuits.Add(current);
178     }
179
180     // Getter methods
181     public static MenuLogicManager Instance { get { return instance; } }
182
183     public bool FirstOpen { get { return firstOpen; } }
184
```

```
185     public int CurrentSceneIndex { get { return currentSceneIndex; } }
```

```
186
```

```
187     public PreviewStructure CurrentPreviewStructure { get { return
```

```
           currentPreviewStructure; } }
```

```
188 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4  using System.Linq;
5  using UnityEngine;
6
7  /// <summary>
8  /// MenuSetupManager serves as the primary script for persistence and      ↵
9  /// communication between the menu and editor/preview scenes.
10 /// </summary>
11 public class MenuSetupManager : MonoBehaviour
12 {
13     // Singleton state reference
14     private static MenuSetupManager instance;
15
16     /// <summary>
17     /// The list of persistent scripts that should be loaded after      ↵
18     /// MenuSetupManager.
19     /// </summary>
20     private Type[] componentsToAdd = new Type[]
21     {
22         typeof(MenuLogicManager)
23     };
24
25     /// <summary>
26     /// The list of editor scenes that exist within the project.
27     /// </summary>
28     private EditorStructure[] editorStructures = new EditorStructure[3];
29
30     /// <summary>
31     /// List of custom circuit IDs corresponding to each element within      ↵
32     /// <seealso cref="previewStructures"/>.<br/><br/>
33     /// This list is primarily utilized to find a <see      ↵
34     /// cref="PreviewStructure"/> within <seealso cref="previewStructures"/>      ↵
35     /// through the IndexOf() method.
36     /// </summary>
37     private List<int> previewStructureIDs = new List<int>();
38
39     /// <summary>
40     /// The list of custom circuits created by the user.<br/><br/>
41     /// Note: a <see cref="PreviewStructure"/> is synonymous with a custom      ↵
42     /// circuit; however a preview structure tends to refer to the actual      ↵
43     /// scene where the custom circuit can be internally viewed.
44     /// </summary>
45     private List<PreviewStructure> previewStructures = new      ↵
46     List<PreviewStructure>();
```

```
42     /// <summary>
43     /// Static constants representing folder names and file names used for >
44     /// serialization.
45     private readonly string editorFolder = "EditorSaves",
46         editorPrefab1Name = "PREFABS_0",
47         editorPrefab2Name = "PREFABS_1",
48         editorPrefab3Name = "PREFABS_2",
49         previewFolder = "PreviewSaves",
50         previewSubdirectory = "CUSTOM_",
51         save1Name = "SAVE_0.json",
52         save2Name = "SAVE_1.json",
53         save3Name = "SAVE_2.json";
54
55     // Enforces a singleton state pattern and imports all serialized      ↵
56     // information.
57     private void Awake()
58     {
59         if (instance != null)
60         {
61             Destroy(this);
62             return;
63         }
64
65         instance = this;
66         DontDestroyOnLoad(this);
67         ImportJSONInformation();
68
69         foreach (Type type in componentsToAdd) gameObject.AddComponent      ↵
70             (type);
71     }
72
73     /// <summary>
74     /// Deletes a requested editor scene within the running game as well      ↵
75     /// as in the save directory.
76     /// </summary>
77     /// <param name="sceneIndex">The editor scene to delete (0-2).</param>
78     public void DeleteEditorStructure(int sceneIndex)
79     {
80         editorStructures[sceneIndex] = null;
81
82         // Obtains the save path pertaining to the requested editor scene.
83         string editorPath = Application.persistentDataPath + "/" +
84             editorFolder + "/";
85         string prefabPath = Application.persistentDataPath + "/" +
86             editorFolder + "/";
87
88         if (sceneIndex == 0) prefabPath += editorPrefab1Name; else if      ↵
89             (sceneIndex == 1) prefabPath += editorPrefab2Name; else      ↵
```

```
    prefabPath += editorPrefab3Name;
84
85    if (sceneIndex == 0) editorPath += save1Name; else if (sceneIndex >
86        == 1) editorPath += save2Name; else editorPath += save3Name;
87
88    File.WriteAllText(editorPath, ""); // Deletes the editor structure ↵
89        JSON file
90    prefabPath += "/";
91
92    // Deletes all connection JSON files, if any
93    string[] filePaths = Directory.GetFiles(prefabPath);
94
95    foreach (string file in filePaths) File.Delete(file);
96 }
97
98 /// <summary>
99 /// Overrides a requested editor scene as a consequence of a new save.
100 /// </summary>
101 /// <param name="sceneIndex">The editor scene to update (0-2).</param>
102 /// <param name="editorStructure">The editor scene object to update.</> ↵
103     param>
104 public void UpdateEditorStructure(int sceneIndex, EditorStructure      ↵
105     editorStructure)
106 {
107     string editorPath = Application.persistentDataPath + "/" +
108         editorFolder + "/";
109
110    if (sceneIndex == 0) editorPath += save1Name; else if (sceneIndex >
111        == 1) editorPath += save2Name; else editorPath += save3Name;
112
113    File.WriteAllText(editorPath, JsonUtility.ToJson
114        (editorStructure));
115 }
116
117 /// <summary>
118 /// Overrides a requested preview structure as a consequence of a new ↵
119     save.
120 /// </summary>
121 /// <param name="previewStructure">The preview structure object to      ↵
122     update.</param>
123 public void UpdatePreviewStructure(PreviewStructure previewStructure)
124 {
125     string previewPath = Application.persistentDataPath + "/" +
126         previewFolder + "/" + previewSubdirectory + previewStructure.ID      ↵
127         + "/SAVE.json";
128
129     File.WriteAllText(previewPath, JsonUtility.ToJson
130         (previewStructure));
131 }
```

```
120
121     /// <summary>
122     /// Serializes all connections pertaining to either a preview or      ↵
123     /// editor structure.
124     /// </summary>
125     /// <param name="isEditor">Whether the referenced connections belong      ↵
126     /// to an editor scene.</param>
127     /// <param name="generateIndex">The editor scene to update (0-2) if      ↵
128     /// the referenced connections belong to an editor scene.</param>
129     /// <param name="connections">The connections to serialize.</param>
130     public void GenerateConnections(bool isEditor, int generateIndex,      ↵
131         List<CircuitConnector.Connection> connections)
132     {
133         // Obtains the path to save all connections to.
134         string path = Application.persistentDataPath + "/" + (isEditor ?      ↵
135             editorFolder : previewFolder) + "/";
136
137         if (isEditor)
138         {
139             if (generateIndex == 0) path += editorPrefab1Name; else if      ↵
140                 (generateIndex == 1) path += editorPrefab2Name; else path +=      ↵
141                 editorPrefab3Name;
142         }
143
144         else
145         {
146             path += previewSubdirectory + generateIndex;
147
148             if (!Directory.Exists(path))
149             {
150                 Directory.CreateDirectory(Application.persistentDataPath +      ↵
151                     "/" + previewFolder + "/" + previewSubdirectory +      ↵
152                     generateIndex);
153             }
154         }
155
156         // In case the folder is already populated, all files are cleared      ↵
157         // from the obtained directory.
158         string[] filePaths = Directory.GetFiles(path);
159
160         foreach (string file in filePaths) File.Delete(file);
161
162         // No point in continuing if there are no connections.
163         if (connections.Count == 0) return;
164
165         int index = 0;
166
167         // Traverses through each connection and generates a corresponding      ↵
168         // JSON file.
```

```
158     foreach (CircuitConnector.Connection connection in connections)
159     {
160         int inputCircuitIndex;
161         int outputCircuitIndex;
162         int inputIndex;
163         int outputIndex;
164
165         // Runs if the input belongs to a custom circuit
166         // customCircuit == null --> a non-custom circuit
167         if (connection.Input.ParentCircuit.customCircuit != null)
168         {
169             /* A custom circuit can be put inside of another custom circuit recursively.
170              * Therefore, to obtain the top-most (actual) custom circuit located in the scene, some calculations must occur.
171              * The primary condition for this is to keep accessing the custom circuit of the parent until it is null.
172              * If it is null, that means the current custom circuit is at the top-most level.
173              * This essentially emulates a linked-list property, where the head is the node with no parent.
174             */
175             Circuit actualCircuit =
176                 connection.Input.ParentCircuit.customCircuit;
177
178             // Obtains the top-most custom circuit
179             while (actualCircuit.customCircuit != null) actualCircuit =
180                 actualCircuit.customCircuit;
181
182             inputCircuitIndex =
183                 EditorStructureManager.Instance.Circuits.IndexOf
184                     (actualCircuit);
185             inputIndex = Array.IndexOf(actualCircuit.Inputs,
186                     connection.Input);
187         }
188
189         // Runs if the input belongs to a non-custom circuit
190         else
191         {
192             inputCircuitIndex =
193                 EditorStructureManager.Instance.Circuits.IndexOf
194                     (connection.Input.ParentCircuit);
195             inputIndex = Array.IndexOf
196                     (connection.Input.ParentCircuit.Inputs,
197                     connection.Input);
198         }
199
200         // Runs if the output belongs to a custom circuit
```

...object\Assets\Scripts\Menu Scripts\MenuSetupManager.cs 6

```
192         if (connection.Output.ParentCircuit.customCircuit != null)
193     {
194         Circuit actualCircuit =
195             connection.Output.ParentCircuit.customCircuit;
196
197         while (actualCircuit.customCircuit != null) actualCircuit =
198             = actualCircuit.customCircuit;
199
200         outputCircuitIndex =
201             EditorStructureManager.Instance.Circuits.IndexOf
202                 (actualCircuit);
203         outputIndex = Array.IndexOf(actualCircuit.Outputs,
204             connection.Output);
205     }
206
207     // Runs if the output belongs to a non-custom circuit
208     else
209     {
210         outputCircuitIndex =
211             EditorStructureManager.Instance.Circuits.IndexOf
212                 (connection.Output.ParentCircuit);
213         outputIndex = Array.IndexOf
214             (connection.Output.ParentCircuit.Outputs,
215                 connection.Output);
216
217         // Creates a corresponding connection identifier from the
218         // obtained indeces and saves to the obtained directory.
219         CircuitConnectorIdentifier circuitConnectionIdentifier = new
220             CircuitConnectorIdentifier(inputCircuitIndex,
221                 outputCircuitIndex, inputIndex, outputIndex);
222         ConnectionSerializer.SerializeConnection(connection,
223             circuitConnectionIdentifier, path + "/CONNECTION_" + index +
224                 ".json");
225         index++;
226     }
227 }
228
229 /// <summary>
230 /// Editor structure variation of the RestoreConnections() method
231 /// utilized to restore serialized connections.
232 /// </summary>
233 /// <param name="sceneIndex">The editor scene to delete (0-2).</param>
234 public void RestoreConnections(int sceneIndex)
235 {
236     // Obtains the path to access the connection JSON files from.
237     string prefabPath = Application.persistentDataPath + "/" +
238         editorFolder + "/";
```

...object\Assets\Scripts\Menu Scripts\MenuSetupManager.cs 7

```
225     if (sceneIndex == 0) prefabPath += editorPrefab1Name; else if      ↵
        (sceneIndex == 1) prefabPath += editorPrefab2Name; else           ↵
        prefabPath += editorPrefab3Name;
226     prefabPath += "/";
228
229     // Calls the primary method with the obtained directory.
230     RestoreConnections(prefabPath, true);
231 }
232
233 /// <summary>
234 /// Preview structure variation of the RestoreConnections() method      ↵
    utilized to restore serialized connections.
235 /// </summary>
236 /// <param name="previewStructure">The preview structure object to      ↵
    access.</param>
237 public void RestoreConnections(PreviewStructure previewStructure)      ↵
{
    RestoreConnections(Application.persistentDataPath + "/" +
    previewFolder + "/" + previewSubdirectory + previewStructure.ID +
    "/", false); }
238
239 /// <summary>
240 /// Deserializes saved connections and restores them to the relevant      ↵
    scene.
241 /// </summary>
242 /// <param name="prefabPath">The path to access the serialized      ↵
    connection files from.</param>
243 /// <param name="isEditor">Whether the path points to an editor      ↵
    structure.</param>
244 private void RestoreConnections(string prefabPath, bool isEditor)
245 {
246     string[] filePaths = Directory.GetFiles(prefabPath);
247
248     // Ensures only JSON files are being accessed.
249     List<string> prefabFilePaths = filePaths.Where(filePath =>      ↵
        filePath.EndsWith(".json")).ToList();
250
251     // Iterates through each connection file and restores it to the      ↵
    scene.
252     foreach (string prefabFilePath in prefabFilePaths)
253     {
254         // Implies the current JSON is a save file rather than a      ↵
            connection file, and should therefore be skipped.
255         if (prefabFilePath.EndsWith("SAVE.json")) continue;
256
257         // Simultaneously creates the connection mesh as well as all      ↵
            relevant values in the form of a
            ConnectionSerializerRestorer object
258         ConnectionSerializerRestorer connectionParent =
```

...object\Assets\Scripts\Menu Scripts\MenuSetupManager.cs 8

```
259     CircuitVisualizer.Instance.VisualizeConnection      ↵
260     (JsonUtility.FromJson<ConnectionSerializer>(File.ReadAllText      ↵
261     (prefabFilePath)));
262
263     // Depending on whether the scene is in the editor or not, the      ↵
264     // method of obtaining the connection's input and output will      ↵
265     // differ.
266     Circuit.Input input = isEditor ?      ↵
267         EditorStructureManager.Instance.Circuits      ↵
268         [connectionParent.circuitConnectorIdentifier.InputCircui      ↵
269         tIndex].Inputs      ↵
270         [connectionParent.circuitConnectorIdentifier.InputIndex]      ↵
271         :      ↵
272         PreviewManager.Instance.Circuits      ↵
273         [connectionParent.circuitConnectorIdentifier.InputCircui      ↵
274         tIndex].Inputs      ↵
275         [connectionParent.circuitConnectorIdentifier.InputIndex]      ↵
276         ;      ↵
277
278     Circuit.Output output = isEditor ?      ↵
279         EditorStructureManager.Instance.Circuits      ↵
280         [connectionParent.circuitConnectorIdentifier.OutputCircu      ↵
281         itIndex].Outputs      ↵
282         [connectionParent.circuitConnectorIdentifier.OutputIndex]      ↵
283         ] :      ↵
284         PreviewManager.Instance.Circuits      ↵
285         [connectionParent.circuitConnectorIdentifier.OutputCircu      ↵
286         itIndex].Outputs      ↵
287         [connectionParent.circuitConnectorIdentifier.OutputIndex]      ↵
288         ];
289
290     // Names and restores the connection within the scene by      ↵
291     // setting the parent circuits of the input and output.
292     connectionParent.parentObject.name = "Connection";
293     CircuitConnector.ConnectRestoration      ↵
294     (connectionParent.parentObject, input, output,      ↵
295     connectionParent.endingWire, connectionParent.startingWire,      ↵
296     isEditor);
297
298 }
299 }
300
301 /// <summary>
302 /// Deletes a requested preview structure within the running game as      ↵
303     // well as in the save directory.
304 /// </summary>
305 /// <param name="previewStructure">The preview structure object to      ↵
306     // delete.</param>
307 public void DeletePreviewStructure(PreviewStructure previewStructure)
308 {
309     int index = previewStructures.IndexOf(previewStructure);
```

...object\Assets\Scripts\Menu Scripts\MenuSetupManager.cs 9

```
281     string folderPath = Application.persistentDataPath + "/" +      ↵
282         previewFolder + "/" + previewSubdirectory + previewStructure.ID;
283
284     previewStructures.Remove(previewStructure);
285     previewStructureIDs.Remove(previewStructureIDs[index]);
286
287     string[] filePaths = Directory.GetFiles(folderPath + "/");
288
289     foreach (string file in filePaths) File.Delete(file);
290
291     Directory.Delete(folderPath);
292 }
293
294 /// <summary>
295 /// Extracts existing JSON data from the game directory to populate      ↵
296     all editor and preview structures.<br/><br/>
297 /// This method is called on startup before anything else.
298 /// </summary>
299 private void ImportJSONInformation()
300 {
301     // Ensures the base editor and preview save folders are created if      ↵
302     // they were removed
303     if (!Directory.Exists(Application.persistentDataPath + "/" +      ↵
304         editorFolder))
305     {
306         Directory.CreateDirectory(Application.persistentDataPath + "/" +      ↵
307             + editorFolder);
308     }
309
310     // Ensures the editor subdirectory save folders are created if      ↵
311     // they were removed
312     if (!Directory.Exists(Application.persistentDataPath + "/" +      ↵
313         editorFolder + "/" + editorPrefab1Name))
314     {
315         Directory.CreateDirectory(Application.persistentDataPath + "/" +      ↵
316             + editorFolder + "/" + editorPrefab1Name);
317     }
318
319     if (!Directory.Exists(Application.persistentDataPath + "/" +      ↵
320         editorFolder + "/" + editorPrefab2Name))
321     {
322         Directory.CreateDirectory(Application.persistentDataPath + "/" +      ↵
323             + editorFolder + "/" + editorPrefab2Name);
324     }
325 }
```

```
...object\Assets\Scripts\Menu Scripts\MenuSetupManager.cs 10
    + editorFolder + "/" + editorPrefab2Name);
319    }
320
321    if (!Directory.Exists(Application.persistentDataPath + "/" +
322        editorFolder + "/" + editorPrefab3Name))
323    {
324        Directory.CreateDirectory(Application.persistentDataPath + "/" +
325            editorFolder + "/" + editorPrefab3Name);
326    }
327
328    // Ensures the relevant editor JSON save files are created if they
329    // were removed, otherwise they are loaded into the game.
330    if (!File.Exists(editorPath + save1Name))
331    {
332        File.Create(editorPath + save1Name);
333        File.SetAttributes(editorPath + save1Name, fileAttributes);
334    }
335    else
336    {
337        editorStructures[0] = JsonUtility.FromJson<EditorStructure>(
338            (File.ReadAllText(editorPath + save1Name)));
339
340        if (!File.Exists(editorPath + save2Name))
341        {
342            File.Create(editorPath + save2Name);
343            File.SetAttributes(editorPath + save2Name, fileAttributes);
344        }
345
346        else
347        {
348            editorStructures[1] = JsonUtility.FromJson<EditorStructure>(
349                (File.ReadAllText(editorPath + save2Name)));
350
351        if (!File.Exists(editorPath + save3Name))
352        {
353            File.Create(editorPath + save3Name);
354            File.SetAttributes(editorPath + save3Name, fileAttributes);
355        }
356
357        else
358        {
359            editorStructures[2] = JsonUtility.FromJson<EditorStructure>(
360                (File.ReadAllText(editorPath + save3Name)));
```

```
360     }
361
362     string[] previewFilePaths = Directory.GetDirectories
363         (Application.persistentDataPath + "/" + previewFolder);
364
365     // Traverses through all valid preview save files and loads them
366     // into the game.
367     foreach (string filePath in previewFilePaths)
368     {
369         string[] previewFiles = Directory.GetFiles(filePath);
370         string jsonFile = previewFiles.FirstOrDefault(s => s.EndsWith
371             ("SAVE.json"));
372
373         if (jsonFile == null) throw new Exception("Preview structure
374             JSON modified outside the script; terminating.");
375
376         try
377         {
378             PreviewStructure previewStructure =
379                 JsonUtility.FromJson<PreviewStructure>(File.ReadAllText
380                     (jsonFile));
381
382             previewStructures.Add(previewStructure);
383             PreviewStructureIDs.Add(previewStructure.ID);
384         }
385     }
386
387     // Getter methods
388     public static MenuSetupManager Instance { get { return instance; } }
389
390     public EditorStructure[] EditorStructures { get { return
391         editorStructures; } }
392
393     public List<int> PreviewStructureIDs { get { return
394         previewStructureIDs; } }
395     public List<PreviewStructure> PreviewStructures { get { return
396         previewStructures; } }
```

```
1  using System;
2  using TMPro;
3  using UnityEngine;
4  using UnityEngine.EventSystems;
5
6  /// <summary>
7  /// BehaviorManagerPreview handles game state actions transitions within a preview scene.
8  /// </summary>
9  public class BehaviorManagerPreview : MonoBehaviour
10 {
11     // Singleton state reference
12     private static BehaviorManagerPreview instance;
13
14     /// <summary>
15     /// The material utilized for empty inputs or outputs the user is currently hovered on.
16     /// </summary>
17     [SerializeField]
18     Material selectedMaterial;
19
20     /// <summary>
21     /// Displays the current world position.
22     /// </summary>
23     [SerializeField]
24     TextMeshProUGUI coordinatesText;
25
26     /// <summary>
27     /// Displays the current label of the empty input or output hovered on, if applicable.
28     /// </summary>
29     [SerializeField]
30     TextMeshProUGUI labelText;
31
32     /// <summary>
33     /// Whether the user is currently hovered onto an empty input or output.
34     /// </summary>
35     private bool isInput;
36
37     /// <summary>
38     /// Whether the user is currently hovered onto a UI element.
39     /// </summary>
40     private bool isUILocked;
41
42     /// <summary>
43     /// The current GameObject raycasted to; guaranteed to be an input or output.
44     /// </summary>
```

```
45     private GameObject currentHitObject;
46
47     /// <summary>
48     /// The current index of the empty input or output that the user is      ↵
49     /// hovered on.
50     /// </summary>
51     private int labelIndex;
52
53     /// <summary>
54     /// The global grid height that all raycasts are cast on.
55     /// </summary>
56     private Plane coordinatesPlane;
57
58     /// <summary>
59     /// Default text utilized when the user is not hovered onto an empty      ↵
60     /// input or output.
61     /// </summary>
62     private readonly string defaultHoverText = "hover over and select      ↵
63     /// inputs/outputs to view their order & label";
64
65     // Enforces a singleton state pattern and establishes the grid plane.
66     private void Awake()
67     {
68         if (instance != null)
69         {
70             Destroy(this);
71             throw new Exception("BehaviorManagerPreview instance already      ↵
72             established; terminating.");
73         }
74
75         instance = this;
76         coordinatesPlane = new Plane(Vector3.down,
77             GridMaintenance.Instance.GridHeight);
78     }
79
80     private void Start() { labelText.text = defaultHoverText; }
81
82     private void Update()
83     {
84         // If hovered onto UI, reset to default values
85         if (EventSystem.current.IsPointerOverGameObject()) { isUILocked =      ↵
86             true; State(null); return; }
87
88         isUILocked = false; // Otherwise, game is not paused.
89
90         bool raycastHit = Physics.Raycast
91             (CameraMovementPreview.Instance.PlayerCamera.ScreenPointToRay
92             (Input.mousePosition), out RaycastHit hitInfo);
93
94 }
```

```
86         // If raycast invalid, reset to default values
87         if (!raycastHit) { State(null); return; }
88
89         // If raycast GameObject is not an input or output, reset to      ↵
89         // default values
90         if (hitInfo.transform.gameObject.layer != 9 &&           ↵
90             hitInfo.transform.gameObject.layer != 10) { State(null);           ↵
90             return; }           ↵
91
92         State(hitInfo.transform.gameObject);
93     }
94
95     /// <summary>
96     /// Exhibits different text states based on hit object properties.<br/> ↵
96     ><br/>
97     /// This text is then written to <seealso cref="labelText"/>.
98     /// </summary>
99     /// <param name="hitObject"></param>
100    private void State(GameObject hitObject)
101    {
102        // Already completed calculations for the same hit object.
103        if (currentHitObject == hitObject) return;
104
105        // Otherwise, restore previous hit object to default values.
106        if (currentHitObject != null)           ↵
106            currentHitObject.GetComponent<MeshRenderer>().material =           ↵
106            currentHitObject.layer == 9 ?           ↵
106                PreviewManager.Instance.InputMaterial :           ↵
106                PreviewManager.Instance.OutputMaterial;
107
108        // UpdateLabelIndex(hitObject) != -1 implies it is an empty input ↵
108        // or output.
109        if (hitObject != null && UpdateLabelIndex(hitObject) != -1)
110        {
111            // Obtains the label text
112            string newLabelText = isInput ?
112                MenuLogicManager.Instance.CurrentPreviewStructure.InputLabel ↵
112                s[labelIndex] :
112                MenuLogicManager.Instance.CurrentPreviewStructure.OutputLabe ↵
112                ls[labelIndex];
113
114            // Sets text values
115            hitObject.GetComponent<MeshRenderer>().material =           ↵
115                selectedMaterial;
116            labelText.text = (isInput ? "input" : "output") + " #" +           ↵
116                (labelIndex + 1) + (newLabelText != "" ? " - " +           ↵
116                    newLabelText : "");
117            currentHitObject = hitObject;
118        }
```

```
119          // Is null and/or invalid input/output; restore default values.
120      else
121      {
122          labelText.text = defaultHoverText;
123
124          if (hitObject == null) currentHitObject = null;
125      }
126  }
127 }
128
129 /// <summary>
130 /// Obtains the current mouse to world position.
131 /// </summary>
132 /// <returns>The current world position.</returns>
133 public Vector3 UpdateCoordinates()
134 {
135     Ray raycastRay =
136         CameraMovementPreview.Instance.PlayerCamera.ScreenPointToRay
137         (Input.mousePosition);
138
139     if (coordinatesPlane.Raycast(raycastRay, out float distance))
140     {
141         Vector3 point = raycastRay.GetPoint(distance);
142
143         // If the UI is not locked, also update the coordinates UI
144         // text.
145         if (!isUILocked) coordinatesText.text = "(" + point.x.ToString("0.0") +
146             ", " + point.z.ToString("0.0") + ")";
147
148         return new Vector3(point.x,
149             GridMaintenance.Instance.GridHeight, point.z);
150     }
151
152     throw new Exception("Unable to obtain new mouse position --
153     raycast failed.");
154 }
155
156 /// <summary>
157 /// Obtains the index of the empty input or output belonging to the
158 /// given hit object.
159 /// </summary>
160 /// <param name="hitObject">The raycasted game object.</param>
161 /// <returns>The index of the empty input or output.</returns>
162 private int UpdateLabelIndex(GameObject hitObject)
163 {
164     if (hitObject.layer == 9)
165     {
166         isInput = true;
167         labelIndex =
```

...ets\Scripts\Preview Scripts\BehaviorManagerPreview.cs 5

```
    MenuLogicManager.Instance.CurrentPreviewStructure.InputOrder >
    s[PreviewManager.Instance.Inputs.IndexOf
    (hitObject.GetComponent<CircuitVisualizer.InputReference>
    () .Input)];
161    }
162
163    else
164    {
165        isInput = false;
166        labelIndex =
            MenuLogicManager.Instance.CurrentPreviewStructure.OutputOrder >
            rs[PreviewManager.Instance.Outputs.IndexOf
            (hitObject.GetComponent<CircuitVisualizer.OutputReference>
            () .Output)];
167    }
168
169    return labelIndex;
170 }
171
172 // Getter methods
173 public static BehaviorManagerPreview Instance { get { return
    instance; } }
174
175 public bool IsUILocked { get { return isUILocked; } }
176 }
```

```
1  using System;
2  using UnityEngine;
3
4  /// <summary>
5  /// CameraMovementPreview handles all player movement within the preview ↵
6  /// scene.<br/><br/>
7  /// Some behaviors (e.g. scrolling) will be enabled or disabled based on ↵
8  /// the state of several other scripts.
9  /// </summary>
10 public class CameraMovementPreview : MonoBehaviour
11 {
12     // Singleton state reference
13     private static CameraMovementPreview instance;
14
15     /// <summary>
16     /// The primary camera utilized by the player.
17     /// </summary>
18     [SerializeField]
19     Camera playerCamera;
20
21     /// <summary>
22     /// The speed under which the player can move around scenes.
23     /// </summary>
24     [SerializeField]
25     float movementSpeed;
26
27     /// <summary>
28     /// The speed under which the player can scroll around scenes.
29     /// </summary>
30     [SerializeField]
31     float scrollSpeed;
32
33     /// <summary>
34     /// How low and high the player can vertically go.
35     /// </summary>
36     [SerializeField]
37     float minHeight, maxHeight;
38
39     /// <summary>
40     /// Moves the player up and down respectively.
41     /// </summary>
42     [SerializeField]
43     KeyCode upKey, downKey;
44
45     /// <summary>
46     /// Keeps track of the mouse position in the current frame.
47     /// </summary>
48     private Vector3 mousePosCurrent;
```

```
48     // Enforces a singleton state pattern and initializes camera values.
49     private void Awake()
50     {
51         if (instance != null)
52         {
53             Destroy(this);
54             throw new Exception("CameraMovementPreview instance already      ↵
55             established; terminating.");
56         }
57         instance = this;
58         ClampPos();
59     }
60
61     private void Start() { mousePosCurrent =
62         BehaviorManagerPreview.Instance.UpdateCoordinates(); }
63
64     // Listens to key inputs and updates movements each frame.
65     private void Update()
66     {
67         float x, y, z;
68
69         Vector3 mousePosPrev = mousePosCurrent;
70
71         mousePosCurrent =
72             BehaviorManagerPreview.Instance.UpdateCoordinates();      ↵
73
74         // X-Z movement via mouse drag
75         if (Input.GetMouseButton(0) && !
76             BehaviorManagerPreview.Instance.IsUILocked)
77         {
78             Vector3 mousePosDelta = mousePosPrev - mousePosCurrent;
79
80             x = mousePosDelta.x;
81             z = mousePosDelta.z;
82         }
83
84         // X-Z movement via WASD
85         else
86         {
87             // Obtains x and z axis values based on input
88             x = Input.GetAxisRaw("Horizontal") * movementSpeed *
89                 Time.deltaTime;
90             z = Input.GetAxisRaw("Vertical") * movementSpeed *
91                 Time.deltaTime;
92         }
93
94         // Y movement via scroll wheel
95         if (Mathf.Abs(Input.mouseScrollDelta.y) > 0)
```

```
91         {
92             y = -Input.mousePosition.y * scrollSpeed * Time.deltaTime;
93         }
94
95         // Y movement via upKey and/or downKey
96         else
97     {
98             y = 0;
99
100            // Determines y axis values (holding both "upKey" and
101            // "downKey" will negate one another)
102            if (Input.GetKey(upKey))
103            {
104                y += movementSpeed * Time.deltaTime;
105            }
106            if (Input.GetKey(downKey))
107            {
108                y -= movementSpeed * Time.deltaTime;
109            }
110        }
111
112        // Adds obtained values and updates position
113        transform.position += x * Vector3.right + y * -CameraRay.direction ↵
114            + z * Vector3.forward;
115        ClampPos();
116        mousePosCurrent =
117            BehaviorManagerPreview.Instance.UpdateCoordinates();
118
119        /// <summary>
120        /// Clamps values to ensure the user cannot traverse out of bounds.
121        /// </summary>
122        private void ClampPos()
123        {
124            Vector3 pos = transform.position;
125
126            pos.y = Mathf.Clamp(pos.y, minHeight, maxHeight);
127            transform.position = pos;
128        }
129
130        // Getter methods
131        public static CameraMovementPreview Instance { get { return
132            instance; } }
133
134        public Camera PlayerCamera { get { return playerCamera; } }
135
136        private Ray CameraRay { get { return playerCamera.ScreenPointToRay
137            (Input.mousePosition); } }
```

135 }

```
1  using System;
2  using System.Collections.Generic;
3  using TMPro;
4  using UnityEngine;
5
6  /// <summary>
7  /// PreviewManager deserializes the current preview structure within the    ↵
8  /// preview scene.
9  /// </summary>
10 public class PreviewManager : MonoBehaviour
11 {
12     // Singleton state reference
13     private static PreviewManager instance;
14
15     /// <summary>
16     /// Material used for empty inputs and outputs respectively.
17     /// </summary>
18     [SerializeField]
19     Material inputMaterial,
20         outputMaterial;
21
22     /// <summary>
23     /// Displays the current input or output label being hovered on, if    ↵
24     /// any.
25     /// </summary>
26     [SerializeField]
27     TextMeshProUGUI nameText;
28
29     /// <summary>
30     /// List of instantiated circuits in the scene.
31     /// </summary>
32     private List<Circuit> circuits = new List<Circuit>();
33
34     /// <summary>
35     /// List of all inputs from circuits in the scene.
36     /// </summary>
37     private List<Circuit.Input> inputs = new List<Circuit.Input>();
38
39     /// <summary>
40     /// List of all outputs from circuits in the scene.
41     /// </summary>
42     private List<Circuit.Output> outputs = new List<Circuit.Output>();
43
44     /// <summary>
45     /// The preview structure to deserialize and load into the preview    ↵
46     /// scene.
47     /// </summary>
48     private PreviewStructure previewStructure;
```

```
47     // Enforces a singleton state pattern
48     private void Awake()
49     {
50         if (instance != null)
51         {
52             Destroy(this);
53             throw new Exception("PreviewManager instance already
54                         established; terminating.");
55         }
56         instance = this;
57     }
58
59     // Begins the deserialization process
60     private void Start()
61     {
62         CursorManager.SetMouseTexture(true);
63         previewStructure =
64             MenuLogicManager.Instance.CurrentPreviewStructure;
65         nameText.text = previewStructure.Name;
66         Deserialize();
67         UpdateIOMaterials();
68     }
69
70     /// <summary>
71     /// Deserializes the current preview structure.<br/><br/>
72     /// The restored values include the circuits, connections, and camera
73     /// position.
74     /// </summary>
75     private void Deserialize()
76     {
77         foreach (CircuitIdentifier circuitIdentifier in
78             previewStructure.Circuits) circuits.Add
79             (CircuitIdentifier.RestoreCircuit(circuitIdentifier));
80
81         MenuSetupManager.Instance.RestoreConnections(previewStructure);
82         CameraMovementPreview.Instance.PlayerCamera.transform.position =
83             previewStructure.CameraLocation;
84     }
85
86     /// <summary>
87     /// Iterates through each circuit in the scene to change the material
88     /// of empty inputs and outputs.
89     /// </summary>
90     private void UpdateIOMaterials()
91     {
92         int inputIndex = 0, outputIndex = 0;
93
94         foreach (Circuit circuit in circuits)
```

```
89         {
90             foreach (Circuit.Input input in circuit.Inputs)
91             {
92                 // If this current input exists in InputOrders, it is     ↵
93                 // guaranteed to be an empty input.
94                 if (previewStructure.InputOrders[inputIndex] != -1)      ↵
95                     input.Transform.GetComponent<MeshRenderer>().material =    ↵
96                     inputMaterial;
97
98
99             foreach (Circuit.Output output in circuit.Outputs)
100            {
101                // If this current output exists in OutputOrders, it is    ↵
102                // guaranteed to be an empty output.
103                if (previewStructure.OutputOrders[outputIndex] != -1)      ↵
104                    output.Transform.GetComponent<MeshRenderer>().material =    ↵
105                    outputMaterial;
106
107            }
108        }
109
110        // Getter methods
111        public static PreviewManager Instance { get { return instance; } }
112
113        public List<Circuit> Circuits { get { return circuits; } }
114
115        public List<Circuit.Input> Inputs { get { return inputs; } }
116
117        public List<Circuit.Output> Outputs { get { return outputs; } }
118
119        public Material InputMaterial { get { return inputMaterial; } }
120
121        public Material OutputMaterial { get { return outputMaterial; } }
122    }
```

```
1  using System;
2  using UnityEngine;
3
4  /// <summary>
5  /// CircuitConnectorIdentifier serializes a connection by storing relevant ↵
6  /// index values.
7  /// </summary>
8  [Serializable]
9  public class CircuitConnectorIdentifier
10 {
11     // Serialized values that are provided within the primary constructor.
12     [SerializeField]
13     int inputCircuitIndex,
14     outputCircuitIndex,
15     inputIndex,
16     outputIndex;
17
18     /// <param name="inputCircuitIndex">The ordered index of the circuit ↵
19     /// that this input is a part of.</param>
20     /// <param name="outputCircuitIndex">The ordered index of the circuit ↵
21     /// that this output is a part of.</param>
22     /// <param name="inputIndex">The ordered index of the circuit's inputs ↵
23     /// that this input is a part of.</param>
24     /// <param name="outputIndex">The ordered index of the circuit's ↵
25     /// outputs that this output is a part of.</param>
26     public CircuitConnectorIdentifier(int inputCircuitIndex, int ↵
27         outputCircuitIndex, int inputIndex, int outputIndex)
28     {
29         this.inputCircuitIndex = inputCircuitIndex;
30         this.outputCircuitIndex = outputCircuitIndex;
31         this.inputIndex = inputIndex;
32         this.outputIndex = outputIndex;
33     }
34
35     // Getter methods
36     public int InputCircuitIndex { get { return inputCircuitIndex; } }
37     public int OutputCircuitIndex { get { return outputCircuitIndex; } }
38     public int InputIndex { get { return inputIndex; } }
39     public int OutputIndex { get { return outputIndex; } }
```

```
1  using System;
2  using UnityEngine;
3
4  /// <summary>
5  /// CircuitIdentifier provides circuit serialization and deserialization by storing relevant enum values.
6  /// </summary>
7  [Serializable]
8  public class CircuitIdentifier
9  {
10     /// <summary>
11     /// CircuitType is the serialized representation of any circuit located within a scene.
12     /// </summary>
13     public enum CircuitType { CUSTOM_CIRCUIT, INPUT_GATE, DISPLAY, BUFFER, AND_GATE, NAND_GATE, NOR_GATE, NOT_GATE, OR_GATE, XOR_GATE }
14
15     /// <summary>
16     /// The circuit type to be serialized by this CircuitIdentifier instance.
17     /// </summary>
18     [SerializeField]
19     public CircuitType circuitType;
20
21     /// <summary>
22     /// Contains a valid custom circuit ID if the referenced CircuitType value is <seealso cref="CircuitType.CUSTOM_CIRCUIT"/>.
23     /// </summary>
24     [SerializeField]
25     public int previewStructureID = -1;
26
27     /// <summary>
28     /// The location of the circuit within the scene.
29     /// </summary>
30     [SerializeField]
31     Vector2 circuitLocation;
32
33     /// <param name="circuit">The circuit to serialize.</param>
34     public CircuitIdentifier(Circuit circuit)
35     {
36         Vector3 pos = circuit.PhysicalObject.transform.position;
37
38         circuitType = CircuitToCircuitType(circuit);
39         circuitLocation = new Vector2(pos.x, pos.z);
40
41         // If the circuit is a custom one, store its ID
42         if (circuitType == CircuitType.CUSTOM_CIRCUIT) previewStructureID = ((CustomCircuit)circuit).PreviewStructure.ID;
43     }
}
```

```
44
45     /// <summary>
46     /// Instantiates and returns a <see cref="Circuit"/> based on the      ↵
47     /// provided CircuitIdentifier (using default visibility settings).
48     /// </summary>
49     /// <param name="circuitIdentifier">The CircuitIdentifier to access      ↵
50     /// and reference.</param>
51     /// <returns>The instantiated <seealso cref="Circuit"/>.</returns>
52     public static Circuit RestoreCircuit(CircuitIdentifier      ↵
53         circuitIdentifier)
54     {
55         return RestoreCircuit(circuitIdentifier, true);
56     }
57
58     /// <summary>
59     /// Instantiates and returns a <see cref="Circuit"/> based on the      ↵
60     /// provided CircuitIdentifier.
61     /// </summary>
62     /// <param name="circuitIdentifier">The CircuitIdentifier to access      ↵
63     /// and reference.</param>
64     /// <param name="visible">False if the circuit is located inside of a      ↵
65     /// custom circuit, otherwise true.</param>
66     /// <returns>The instantiated <seealso cref="Circuit"/>.</returns>
67     public static Circuit RestoreCircuit(CircuitIdentifier      ↵
68         circuitIdentifier, bool visible)
69     {
70         Vector2 pos = visible ? circuitIdentifier.circuitLocation :      ↵
71             Vector2.positiveInfinity;
72
73         switch (circuitIdentifier.circuitType)
74         {
75             case CircuitType.CUSTOM_CIRCUIT:
76                 return new CustomCircuit
77                     (MenuSetupManager.Instance.PreviewStructures
78                      [MenuSetupManager.Instance.PreviewStructureIDs.IndexOf
79                          (circuitIdentifier.previewStructureID)], pos, visible);
80             case CircuitType.INPUT_GATE:
81                 return new InputGate(pos);
82             case CircuitType.DISPLAY:
83                 return new Display(pos);
84             case CircuitType.BUFFER:
85                 return new Buffer(pos);
86             case CircuitType.AND_GATE:
87                 return new AndGate(pos);
88             case CircuitType.NAND_GATE:
89                 return new NAndGate(pos);
90             case CircuitType.NOR_GATE:
91                 return new NOrGate(pos);
92             case CircuitType.NOT_GATE:
```

```
82             return new NotGate(pos);
83         case CircuitType.OR_GATE:
84             return new OrGate(pos);
85         case CircuitType.XOR_GATE:
86             return new XOrGate(pos);
87         default:
88             throw new Exception("Invalid circuit type.");
89     }
90 }
91
92 /// <summary>
93 /// Converts the provided <see cref="Circuit"/> to a valid <seealso href="CircuitType"/> for serialization.
94 /// </summary>
95 /// <param name="circuit">The circuit to convert.</param>
96 /// <returns>The converted <seealso cref="CircuitType"/>.</returns>
97 private static CircuitType CircuitToCircuitType(Circuit circuit)
98 {
99     Type type = circuit.GetType();
100
101    if (type == typeof(CustomCircuit)) return CircuitType.CUSTOM_CIRCUIT;
102
103    if (type == typeof(InputGate)) return CircuitType.INPUT_GATE;
104
105    if (type == typeof(Display)) return CircuitType.DISPLAY;
106
107    if (type == typeof(Buffer)) return CircuitType.BUFFER;
108
109    if (type == typeof(AndGate)) return CircuitType.AND_GATE;
110
111    if (type == typeof(NAndGate)) return CircuitType.NAND_GATE;
112
113    if (type == typeof(NOrGate)) return CircuitType.NOR_GATE;
114
115    if (type == typeof(NotGate)) return CircuitType.NOT_GATE;
116
117    if (type == typeof(OrGate)) return CircuitType.OR_GATE;
118
119    if (type == typeof(XOrGate)) return CircuitType.XOR_GATE;
120
121    throw new Exception("Invalid circuit type.");
122 }
123 }
```

```
1  using System;
2  using System.IO;
3  using UnityEngine;
4
5  /// <summary>
6  /// ConnectionSerializer stores mesh and index information pertaining to the assigned connection for serialization. ↵
7  /// </summary>
8  [Serializable]
9  public class ConnectionSerializer
10 {
11     /// <summary>
12     /// Whether <seealso cref="startingMesh"/> is equal to <seealso cref="endingMesh"/>. ↵
13     /// </summary>
14     [SerializeField]
15     bool singleWired;
16
17     /// <summary>
18     /// Contains relevant index information used to identify the connection's input and output circuit(s). ↵
19     /// </summary>
20     [SerializeField]
21     CircuitConnectorIdentifier circuitConnectorIdentifier;
22
23     /// <summary>
24     /// Serialized mesh data for the starting wire. ↵
25     /// </summary>
26     [SerializeField]
27     MeshSerializer startingMesh;
28
29     /// <summary>
30     /// Serialized mesh data for the ending wire. ↵
31     /// </summary>
32     [SerializeField]
33     MeshSerializer endingMesh;
34
35     /// <summary>
36     /// Serialized mesh data for the non-starting/ending wires.<br/><br/> ↵
37     /// If there is only a starting and ending wire, its value will be null. ↵
38     /// </summary>
39     [SerializeField]
40     MeshSerializer parentMesh;
41
42     // Private constructor; a ConnectionSerializer can only be instantiated through its primary constructor. ↵
43     private ConnectionSerializer() { }
44
```

```
45     /// <summary>
46     /// Instantiates and populates a <seealso cref="ConnectionSerializer"/> with the assigned values; saves to the provided path.
47     /// </summary>
48     /// <param name="connection">The connection to serialize.</param>
49     /// <param name="circuitConnectorIdentifier">The obtained <seealso href="CircuitConnectorIdentifier"/> representing this connection.</param>
50     /// <param name="path">The directory to save the serialized information.</param>
51     public static void SerializeConnection(CircuitConnector.Connection connection, CircuitConnectorIdentifier circuitConnectorIdentifier, string path)
52     {
53         ConnectionSerializer connectionSerializer = new ConnectionSerializer();
54
55         // Assigns relevant values
56         connectionSerializer.circuitConnectorIdentifier =
57             circuitConnectorIdentifier;
58         connectionSerializer.startingMesh = new MeshSerializer
59             (connection.StartingWire.transform.GetChild(0).GetChild
60                 (0).GetComponent<MeshFilter>().mesh,
61                 connection.StartingWire.transform);
62         connectionSerializer.endingMesh = new MeshSerializer
63             (connection.EndingWire.transform.GetChild(0).GetChild
64                 (0).GetComponent<MeshFilter>().mesh,
65                 connection.EndingWire.transform);
66         connectionSerializer.singleWired = connection.StartingWire ==
67             connection.EndingWire;
68
69         // If the connection has a parent mesh, serialize its information and save to parentMesh.
70         if (connection.GetComponent<MeshFilter>() != null)
71             connectionSerializer.parentMesh = new MeshSerializer
72                 (connection.GetComponent<MeshFilter>().mesh,
73                 connection.transform);
74
75         // Writes object to directory
76         File.WriteAllText(path, JsonUtility.ToJson(connectionSerializer));
77     }
78
79     // Getter methods
80     public bool SingleWired { get { return singleWired; } }
81
82     public CircuitConnectorIdentifier CircuitConnectorIdentifier { get
83         { return circuitConnectorIdentifier; } }
84
85     public MeshSerializer StartingMesh { get { return startingMesh; } }
```

```
74
75     public MeshSerializer EndingMesh { get { return endingMesh; } }
76
77     public MeshSerializer ParentMesh { get { return parentMesh; } }
78 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  /// <summary>
6  /// EditorStructure contains all serializable values to restore an editor scene.
7  /// </summary>
8  [Serializable]
9  public class EditorStructure
10 {
11     /// <summary>
12     /// Whether grid snapping is enabled.
13     /// </summary>
14     [SerializeField]
15     bool inGridMode;
16
17     /// <summary>
18     /// The list of circuits determined to be powered inputs.
19     /// </summary>
20     [SerializeField]
21     List<bool> isPoweredInput = new List<bool>();
22
23     /// <summary>
24     /// The list of circuit identifiers pertaining to each circuit within the scene.
25     /// </summary>
26     [SerializeField]
27     List<CircuitIdentifier> circuits = new List<CircuitIdentifier>();
28
29     /// <summary>
30     /// The circuit index representation of all bookmarked circuits.<br/><br/>
31     /// All indeces except that of custom circuits are unique.
32     /// </summary>
33     [SerializeField]
34     List<int> bookmarks = new List<int>();
35
36     /// <summary>
37     /// The custom circuit ID representation of all bookmarked circuits.<br/><br/>
38     /// If a bookmarked circuit is not a custom circuit, -1 is utilized as a corresponding identifier.
39     /// </summary>
40     [SerializeField]
41     List<int> bookmarkIDs = new List<int>();
42
43     /// <summary>
44     /// Name of the editor scene assigned when a save slot is first used.
```

```
45     /// <summary>
46     [SerializeField]
47     string name;
48
49     /// <summary>
50     /// Location of the camera within the editor scene.
51     /// </summary>
52     [SerializeField]
53     Vector3 cameraLocation;
54
55     public EditorStructure(string name) { this.name = name; }
56
57     // Getter and setter methods
58     public bool InEditMode { get { return inEditMode; } set { inEditMode = value; } }
59
60     public List<bool> IsPoweredInput { get { return isPoweredInput; } set { isPoweredInput = value; } }
61
62     public List<CircuitIdentifier> Circuits { get { return circuits; } set { circuits = value; } }
63
64     public List<int> Bookmarks { get { return bookmarks; } set { bookmarks = value; } }
65
66     public List<int> BookmarkIDs { get { return bookmarkIDs; } set { bookmarkIDs = value; } }
67
68     public Vector3 CameraLocation { get { return cameraLocation; } set { cameraLocation = value; } }
69
70     // Getter method
71     public string Name { get { return name; } }
72 }
```

```
1  using System;
2  using UnityEngine;
3
4  /// <summary>
5  /// InternalConnection serializes all connections occurring in a custom      ↵
6  /// circuit.
7  /// </summary>
8  [Serializable]
9  public class InternalConnection
10 {
11     /// <summary>
12     /// The input and output indeces of the connection.<br/><br/>
13     /// These values are referenced once the <see cref="Circuit.Input"/>      ↵
14     /// and <see cref="Circuit.Output"/> lists of the <see      ↵
15     /// cref="CustomCircuit"/> are established.
16     /// </summary>
17     [SerializeField] int inputIndex, outputIndex;
18
19     public InternalConnection(int inputIndex, int outputIndex)
20     {
21         this.inputIndex = inputIndex;
22         this.outputIndex = outputIndex;
23     }
24
25     // Getter methods
26     public int InputIndex { get { return inputIndex; } }
27
28     public int OutputIndex { get { return outputIndex; } }
29 }
```

```
1  using System;
2  using UnityEngine;
3
4  /// <summary>
5  /// MeshSerializer serves as a wrapper class for containing all relevant    ↵
6  /// mesh and transform values for serialization and deserialization of wires.
7  /// </summary>
8  [Serializable]
9  public class MeshSerializer
10 {
11     [SerializeField]
12     int[] triangles;
13
14     [SerializeField]
15     Vector2[] uv;
16
17     [SerializeField]
18     Vector3 position, rotation, scale;
19
20     [SerializeField]
21     Vector3[] normals, vertices;
22
23     /// <summary>
24     /// Extracts mesh and transform values pertaining to a wire.
25     /// </summary>
26     /// <param name="mesh">The mesh to extract relevant values from.</param>    ↵
27     /// <param name="parentTransform">The parent transform of the    ↵
28     /// GameObject containing the mesh (could be itself).</param>
29     public MeshSerializer(Mesh mesh, Transform parentTransform)
30     {
31         triangles = mesh.triangles;
32         uv = mesh.uv;
33         position = parentTransform.position;
34         rotation = parentTransform.eulerAngles;
35         scale = parentTransform.localScale;
36         normals = mesh.normals;
37         vertices = mesh.vertices;
38
39         // Getter methods
40         public int[] Triangles { get { return triangles; } }
41         public Vector2[] UV { get { return uv; } }
42         public Vector3 Position { get { return position; } }
43         public Vector3 Rotation { get { return rotation; } }
44
45
46 }
```

```
47     public Vector3 Scale { get { return scale; } }
```

```
48
```

```
49     public Vector3[] Normals { get { return normals; } }
```

```
50
```

```
51     public Vector3[] Vertices { get { return vertices; } }
```

```
52 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  /// <summary>
6  /// PreviewStructure contains all serializable values to restore a preview ↵
7  /// scene.
8  /// </summary>
9  [Serializable]
10 public class PreviewStructure
11 {
12     /// <summary>
13     /// The list of circuit identifiers pertaining to each circuit within ↵
14     /// the scene.
15     /// </summary>
16     [SerializeField]
17     List<CircuitIdentifier> circuits = new List<CircuitIdentifier>();
18
19     /// <summary>
20     /// The unique ID assigned to this preview structure.<br/><br/>
21     /// Functionally, this ID is utilized to access the specific folder ↵
22     /// under which the connection and save information of the preview ↵
23     /// structure is.
24     /// </summary>
25     [SerializeField]
26     int id;
27
28     /// <summary>
29     /// The order in which empty inputs and outputs were selected by the ↵
30     /// user.<br/><br/>
31     /// Functionally, a visualized custom circuit will output these inputs ↵
32     /// and outputs in their selected order (bottom to top).
33     /// </summary>
34     [SerializeField]
35     List<int> inputOrders,
36         outputOrders;
37
38     /// <summary>
39     /// Identifying list of connections that exist within the custom ↵
40     /// circuit.
41     /// </summary>
42     [SerializeField]
43     List<InternalConnection> connections;
44
45     /// <summary>
46     /// Name of the custom circuit.
47     /// </summary>
48     [SerializeField]
49     string name;
```

```
43
44     /// <summary>
45     /// The corresponding user-assigned label for each empty input/output.
46     /// </summary>
47     [SerializeField]
48     List<string> inputLabels,
49     outputLabels;
50
51     /// <summary>
52     /// Location of the camera within the editor scene.
53     /// </summary>
54     [SerializeField]
55     Vector3 cameraLocation;
56
57     public PreviewStructure(string name) { this.name = name; }
58
59     /// <summary>
60     /// Internal class utilized to obtain the custom circuit ID via in-      ↵
61     /// scene raycasting.
62     /// </summary>
63     public class PreviewStructureReference : MonoBehaviour
64     {
65         private int id;
66
67         public int ID { get { return id; } set { id = value; } }
68
69         // Getter and setter methods
70         public List<CircuitIdentifier> Circuits { get { return circuits; } set      ↵
71             { circuits = value; } }
72
73         public int ID { get { return id; } set { id = value; } }
74
75         public List<int> InputOrders { get { return inputOrders; } set      ↵
76             { inputOrders = value; } }
77
78         public List<int> OutputOrders { get { return outputOrders; } set      ↵
79             { outputOrders = value; } }
80
81         public List<InternalConnection> Connections { get { return      ↵
82             connections; } set { connections = value; } }
83
84         public List<string> InputLabels { get { return inputLabels; } set      ↵
85             { inputLabels = value; } }
86
87         public List<string> OutputLabels { get { return outputLabels; } set      ↵
88             { outputLabels = value; } }
89
90         public Vector3 CameraLocation { get { return cameraLocation; } set      ↵
```

```
    { cameraLocation = value; } }  
85  
86     // Getter method  
87     public string Name { get { return name; } }  
88 }
```

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5
6  /// <summary>
7  /// CircuitCaller handles every circuit call after a short delay defined in <see cref="Circuit"/>.
8  /// </summary>
9  public class CircuitCaller : MonoBehaviour
10 {
11     private static CircuitCaller instance; // Singleton state reference
12
13     // Enforces a singleton state pattern
14     private void Awake()
15     {
16         if (instance != null)
17         {
18             Destroy(this);
19             throw new Exception("CircuitCaller instance already established; terminating.");
20         }
21
22         instance = this;
23     }
24
25     /// <summary>
26     /// Starts a coroutine that shortly accesses the list of provided update calls.
27     /// </summary>
28     /// <param name="updateCalls">The list of update calls to pursue.</param>
29     public static void InitiateUpdateCalls(List<Circuit.UpdateCall> updateCalls) { instance.StartCoroutine(UpdateCalls(updateCalls)); }
30
31     /// <summary>
32     /// Attempts to access the list of provided update calls.
33     /// </summary>
34     /// <param name="updateCalls">The list of update calls to call.</param>
35     private static IEnumerator UpdateCalls(List<Circuit.UpdateCall> updateCalls)
36     {
37         yield return new WaitForSeconds(Circuit.clockSpeed);
38
39         foreach (Circuit.UpdateCall updateCall in updateCalls)
40         {
41             // Sometime between the call initiation and now, the referenced output was destroyed and should no longer be
```

```
    pursued.  
42        if (updateCall.Input.ParentOutput == null) continue;  
43  
44        if (!CustomCircuitTest(updateCall)) continue;  
45  
46        // Otherwise, the update call is accessed to update the  
        // relevant circuits.  
47        Circuit.UpdateCircuit(updateCall.Powered, updateCall.Input,  
                                updateCall.Output);  
48    }  
49}  
50  
51 /// <summary>  
52 /// Ensures that an update call pertaining to a custom circuit only  
/// runs if its custom circuit is not deleted.  
53 /// </summary>  
54 /// <param name="updateCall">The update call to test.</param>  
55 /// <returns>Whether this update call should be utilized.</returns>  
56 private static bool CustomCircuitTest(Circuit.UpdateCall updateCall)  
57 {  
    // In preview scene, therefore not necessary to run the test  
    if (EditorStructureManager.Instance == null) return true;  
60  
    // If the input of an update call is under a parent circuit, it is  
    // guaranteed that its output is as well.  
61    bool isInternalConnection =  
        updateCall.Input.ParentCircuit.customCircuit != null;  
62  
    // Connection does not pertain to the inside of a custom circuit.  
63    if (!isInternalConnection) return true;  
64  
    // Otherwise, obtain the top-most custom circuit and check to see  
    // if it is still within the scene.  
65    CustomCircuit customCircuitParent =  
        updateCall.Input.ParentCircuit.customCircuit;  
66  
    while (customCircuitParent.customCircuit != null)  
        customCircuitParent = customCircuitParent.customCircuit;  
67  
    return !customCircuitParent.ShouldDereference;  
73}  
74  
75 /// <summary>  
76 /// Deletes the specified circuit from the scene.  
77 /// </summary>  
78 /// <param name="circuit">The circuit to destroy.</param>  
79 public static void Destroy(Circuit circuit)  
80 {  
    // First disconnects any potential input connections
```

```
82         foreach (Circuit.Input input in circuit.Inputs)
83     {
84         if (input.Connection != null)
85         {
86             CircuitConnector.Disconnect(input.Connection);
87         }
88     }
89
90     // Then disconnects any potential output connections
91     foreach (Circuit.Output output in circuit.Outputs)
92     {
93         foreach (CircuitConnector.Connection connection in new
94             List<CircuitConnector.Connection>(output.Connections))
95         {
96             CircuitConnector.Disconnect(connection);
97         }
98
99         // Ensures all remaining calls within the custom circuit are
100        // skipped
101        if (circuit.GetType() == typeof(CustomCircuit)) ((CustomCircuit)
102            circuit).ShouldDereference = true;
103
104        EditorStructureManager.Instance.DisplaySavePrompt = true; // -->
105        // Destroying a circuit triggers the save prompt
106        EditorStructureManager.Instance.Circuits.Remove(circuit); // -->
107        // Removes circuit for potential serialization
108        Destroy(circuit.PhysicalObject);
109    }
110 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  /// <summary>
6  /// CircuitConnector facilitates the connection process between circuits ↵
7  /// in the scene editor.
8  /// </summary>
9  public class CircuitConnector : MonoBehaviour
10 {
11     // Singleton state reference
12     private static CircuitConnector instance;
13
14     /// <summary>
15     /// Reference to the wire prefab.
16     /// </summary>
17     [SerializeField]
18     GameObject wireReference;
19
20     /// <summary>
21     /// The material for powered and unpowered statuses respectively.
22     /// </summary>
23     [SerializeField]
24     Material poweredMaterial, unpoweredMaterial;
25
26     private bool cancelled;
27
28     private Connection currentConnection;
29
30     private GameObject currentWire;
31
32     private int count;
33
34     private Vector3 startingWirePos, currentPos;
35
36     /// <summary>
37     /// Represents a connection from the output circuit to the input ↵
38     /// circuit.
39     /// </summary>
40     public class Connection : MonoBehaviour
41     {
42         /// <summary>
43         /// The input associated with the connection.
44         /// </summary>
45         private Circuit.Input input;
46
47         /// <summary>
48         /// The output associated with the connection.
49         /// </summary>
```

```
48     private Circuit.Output output;
49
50     /// <summary>
51     /// The starting and ending wires associated with the connection.
52     /// </summary>
53     private GameObject endingWire, startingWire;
54
55     // Getter and setter methods
56     public Circuit.Input Input { get { return input; } set { input = value; } }
57
58     public Circuit.Output Output { get { return output; } set { output = value; } }
59
60     public GameObject EndingWire { get { return endingWire; } set { endingWire = value; } }
61
62     public GameObject StartingWire { get { return startingWire; } set { startingWire = value; } }
63 }
64
65 // Enforces a singleton state pattern and disables update calls.
66 private void Awake()
67 {
68     if (instance != null)
69     {
70         Destroy(this);
71         throw new Exception("CircuitConnector instance already established; terminating.");
72     }
73
74     instance = this;
75     enabled = false;
76 }
77
78 // While the connection has not been cancelled or completed, this method allows for creating pivots to organize the wire.
79 private void Update()
80 {
81     // If the connection process has been cancelled, disable update calls and return.
82     if (cancelled)
83     {
84         cancelled = enabled = false;
85         return;
86     }
87
88     // If the game is currently paused, skip frame.
89     if (BehaviorManager.Instance.CurrentStateType ==
```

```
    BehaviorManager.StateType.PAUSED) return;  
90  
91     // Whether the user is staring at a valid input or output for  
92     // completing the connection.  
93     bool staringAtIO = Physics.Raycast  
94         (CameraMovement.Instance.PlayerCamera.ScreenPointToRay  
95         (Input.mousePosition), out RaycastHit hitInfo) &&  
96         hitInfo.transform.gameObject.layer ==  
97         BehaviorManager.Instance.IOLayerCheck;  
98  
99     // The position to move the end of the wire to.  
100    // If hovered onto a valid input or output for completing the  
101    // connection, it will snap to its position.  
102    Vector3 pos = staringAtIO ? hitInfo.transform.position :  
103        Coordinates.Instance.ModePos;  
104  
105    pos.y = GridMaintenance.Instance.GridHeight;  
106    UpdatePosition(currentWire, currentPos, pos); // Updates the  
107    // position of the wire.  
108  
109    // Creates a new pivot as long as the wire is active (has a length  
110    // >= 0).  
111    if (Input.GetMouseButtonUp(0) && currentWire.activeSelf)  
112    {  
113        count++;  
114  
115        if (count == 2) startingWirePos = currentPos;  
116  
117        currentConnection.EndingWire = currentWire;  
118  
119        // Places a new wire at the current pivot  
120        InstantiateWire(currentConnection,  
121            Coordinates.Instance.ModePos);  
122    }  
123 }  
124  
125 /// <summary>  
126 /// Final step in restoring the logic of a serialized connection by  
127     // initializing a <seealso cref="Connection"/> instance and assigning  
128     // all of its values.  
129 /// </summary>  
130 /// <param name="prefab">The base GameObject of the connection.</  
131     param>  
132 /// <param name="input">The input of the connection.</param>  
133 /// <param name="output">The output of the connection.</param>  
134 /// <param name="endingWire">The ending wire of the connection.</  
135     param>  
136 /// <param name="startingWire">The starting wire of the connection.</  
137     param>
```

```
...ect\Assets\Scripts\Shared Scripts\CircuitConnector.cs 4
123     /// <param name="isEditor">Whether the connection is being restored in the editor.</param>
124     public static void ConnectRestoration(GameObject prefab, Circuit.Input input, Circuit.Output output, GameObject endingWire, GameObject startingWire, bool isEditor)
125     {
126         Connection connection = prefab.AddComponent<Connection>();
127
128         connection.Input = input;
129         connection.Output = output;
130         input.Connection = connection;
131         input.ParentOutput = output;
132         output.Connections.Add(connection);
133         output.ChildInputs.Add(input);
134         connection.EndingWire = endingWire;
135         connection.StartingWire = startingWire;
136
137         if (isEditor) EditorStructureManager.Instance.Connections.Add
138             (connection); // Re-adds connection for potential serialization
139
140         // If the circuit is an input gate, do not pursue an update call.
141         if (output.ParentCircuit.GetType() == typeof(InputGate)) return;
142
143         Circuit.UpdateCircuit(input, output);
144     }
145
146     // Finalizes the current connection in progress.
147     public static void Connect(Circuit.Input input, Circuit.Output output)
148     {
149         Instance.count = -1;
150         Instance.currentConnection.Input = input;
151         Instance.currentConnection.Output = output;
152         Instance.currentConnection.Input.Connection =
153             Instance.currentConnection;
154         Instance.currentConnection.Output.Connections.Add
155             (Instance.currentConnection);
156         Instance.currentConnection.Output.ChildInputs.Add(input);
157         Instance.currentConnection.EndingWire.name = "Ending Wire";
158         Instance.OptimizeMeshes();
159         EditorStructureManager.Instance.Connections.Add
160             (Instance.currentConnection); // Adds connection for potential serialization
161
162         Instance.currentConnection = null; Instance.currentWire = null;
163         Instance.cancelled = true;
164         Circuit.UpdateCircuit(input, output);
165     }
166
167     /// <summary>
168     /// Removes a connection from the editor scene.
169 }
```

```
164     /// </summary>
165     /// <param name="connection"></param>
166     public static void Disconnect(Connection connection)
167     {
168         EditorStructureManager.Instance.DisplaySavePrompt = true;
169         EditorStructureManager.Instance.Connections.Remove(connection); // ➔
170         // Removes connection for potential serialization
171         Circuit.UpdateCircuit(false, connection.Input, null);
172         connection.Input.Connection = null;
173         connection.Output.Connections.Remove(connection);
174         connection.Output.ChildInputs.Remove(connection.Input);
175         Destroy(connection.gameObject);
176     }
177     /// <summary>
178     /// Updates all wire materials pertaining to a connection, if
179     /// applicable.
180     /// </summary>
181     /// <param name="connection"></param>
182     public static void UpdateConnectionMaterial(Connection connection,
183         bool powered)
184     {
185         // If there is an optimized mesh in the wire, update it.
186         if (connection.GetComponent<MeshRenderer>() != null) ➔
187             connection.GetComponent<MeshRenderer>().material = powered ? ➔
188                 Instance.poweredMaterial : Instance.unpoweredMaterial;
189
190         // If there is a starting mesh in the wire, update it.
191         if (connection.StartingWire != null) ➔
192             connection.StartingWire.GetComponentInChildren<MeshRenderer>()
193                 ().material = powered ? Instance.poweredMaterial : ➔
194                 Instance.unpoweredMaterial;
195
196         // If there is an ending mesh in the wire, update it.
197         if (connection.EndingWire != null) ➔
198             connection.EndingWire.GetComponentInChildren<MeshRenderer>()
199                 ().material = powered ? Instance.poweredMaterial : ➔
200                 Instance.unpoweredMaterial;
201     }
202
203     /// <summary>
204     /// Begins the connection process at the specified position.
205     /// </summary>
206     /// <param name="pos"></param>
207     public void BeginConnectionProcess(Vector3 pos)
208     {
209         count = 0;
210         cancelled = false;
```

```
202         currentConnection = InstantiateConnection();
203         InstantiateWire(currentConnection, pos);
204         currentConnection.StartingWire = currentWire;
205         currentWire.name = "Starting Wire";
206         enabled = true; // Enables frame-by-frame update calls from Unity
207     }
208
209     /// <summary>
210     /// Cancels the current connection process.
211     /// </summary>
212     public void CancelConnectionProcess()
213     {
214         count = -1;
215         cancelled = true;
216         Destroy(currentConnection.gameObject);
217     }
218
219     /// <summary>
220     /// Creates a new wire at the specified position for the given
221     /// connection.
222     /// </summary>
223     /// <param name="connection">The connection this wire is a part of.</param>
224     private void InstantiateWire(Connection connection, Vector3 a)
225     {
226         currentWire = Instantiate(wireReference, connection.transform);
227         currentPos = new Vector3(a.x, GridMaintenance.Instance.GridHeight,
228                                 a.z);
229         currentWire.transform.position = currentPos;
230         currentWire.SetActive(false);
231     }
232
233     /// <summary>
234     /// Specific signature of <seealso cref="UpdatePosition(GameObject,
235     /// Vector3, Vector3, bool)"> under which isCentered is always false.
236     /// </summary>
237     /// <param name="wire">The wire to move.</param>
238     public static void UpdatePosition(GameObject wire, Vector3 a, Vector3
239                                         b) { UpdatePosition(wire, a, b, false); }
240
241     /// <summary>
242     /// Updates the start and end positions of the specified wire.
243     /// </summary>
244     /// <param name="wire">The wire to move.</param>
245     /// <param name="a">The starting position.</param>
```

```
245     /// <param name="b">The ending position.</param>
246     /// <param name="isCentered">Whether the wire should be centered.</param>
247     public static void UpdatePosition(GameObject wire, Vector3 a, Vector3 b, bool isCentered)
248     {
249         // If the wire is centered, then startingWire == endingWire and it must be positioned differently.
250         if (isCentered) wire.transform.position = (a + b) / 2;
251
252         wire.transform.localScale = new Vector3(1, 1, (a - b).magnitude);
253         wire.SetActive(wire.transform.localScale.z != 0);
254         wire.transform.LookAt(b);
255
256         // Ensures the height of the wire does not exceed the global grid height
257         Vector3 temp = wire.transform.position;
258
259         temp.y = GridMaintenance.Instance.GridHeight;
260         wire.transform.position = temp;
261     }
262
263     /// <summary>
264     /// Optimizes all non-starting and non-ending wire meshes by merging them together into a single mesh.
265     /// </summary>
266     private void OptimizeMeshes()
267     {
268         // There is a single wire in the connection
269         if (currentConnection.StartingWire == currentConnection.EndingWire)
270         {
271             Destroy(currentWire);
272
273             // If there is a single wire in the connection, it must be centered so UpdatePosition() can work properly.
274             currentConnection.StartingWire.transform.position =
275                 (currentConnection.Input.Transform.position +
276                  currentConnection.Output.Transform.position) / 2;
277
278             // Furthermore, the pivot must be altered.
279             currentConnection.StartingWire.transform.GetChild(0).transform.localPosition = Vector3.back * 0.5f;
280
281             // Ensures the height of the wire does not exceed the global grid height
282             Vector3 temp =
283                 currentConnection.StartingWire.transform.position;
```

```
282         temp.y = GridMaintenance.Instance.GridHeight;
283         currentConnection.StartingWire.transform.position = temp;
284         return;
285     }
286
287     // Ensures the starting wire behaves properly with the
288     // UpdatePosition() method
289     currentConnection.StartingWire.transform.position =
290         startingWirePos;
291     currentConnection.StartingWire.transform.eulerAngles += Vector3.up
292         * 180;
293
294     // Checks to see whether there are exactly 3 wires.
295     // 2 of them are the starting and ending wires (cannot be merged
296     // into a mesh) and the third wire is the placement wire, which
297     // will be deleted regardless.
298     if (currentConnection.transform.childCount == 3)
299     {
300         Destroy(currentWire);
301         return;
302     }
303
304     // Begins the mesh combination process
305     List<CombineInstance> combineInstances = new List<CombineInstance>()
306     {
307         foreach (Transform child in currentConnection.transform)
308         {
309             GameObject childObj = child.gameObject;
310
311             if (childObj == currentConnection.StartingWire || childObj == currentConnection.EndingWire || childObj == currentWire)
312                 continue;
313
314             MeshFilter meshFilter =
315                 childObj.GetComponentInChildren<MeshFilter>();
316             CombineInstance combineInstance = new CombineInstance();
317
318             combineInstance.mesh = meshFilter.mesh;
319             combineInstance.transform =
320                 meshFilter.transform.localToWorldMatrix;
321
322             combineInstances.Add(combineInstance);
323         }
324
325         Mesh combinedMesh = new Mesh();
326
327         combinedMesh.CombineMeshes(combineInstances.ToArray());
328     }
```

```
321         // Deletes the original instances of the unmerged meshes.
322         foreach (Transform child in currentConnection.transform)
323     {
324             GameObject childObj = child.gameObject;
325
326             if (childObj == currentConnection.StartingWire || childObj == ↵
327                 currentConnection.EndingWire) continue;
328             Destroy(childObj);
329         }
330
331         MeshFilter combinedMeshFilter = ↵
332             currentConnection.gameObject.AddComponent<MeshFilter>();
333         currentConnection.gameObject.AddComponent<MeshRenderer>();
334         combinedMeshFilter.mesh = combinedMesh;
335         currentConnection.gameObject.layer = 11;
336         currentConnection.gameObject.AddComponent<MeshCollider>();
337     }
338
339     /// <summary>
340     /// Creates a new connection GameObject.
341     /// </summary>
342     /// <returns>The connection component of a newly instantiated ↵
343     /// GameObject.</returns>
344     private Connection InstantiateConnection() { return new GameObject
345         ("Connection").AddComponent<Connection>(); }
346
347     // Getter methods
348     public static CircuitConnector Instance { get { return instance; } }
349     public Connection CurrentConnection { get { return ↵
350         currentConnection; } }
```

```
1  using UnityEngine;
2
3  ///<summary>
4  ///<see cref="Circuit"/> wrapper class attached to    ↵
5  /// all instantiated circuit GameObjects.<br/><br/>
6  /// This script is utilized to obtain the target circuit for any given    ↵
7  /// GameObject through raycasts.
8  ///</summary>
9  public class CircuitReference : MonoBehaviour
10 {
11     ///<summary>
12     /// The circuit that this
13     ///</summary>
14     private Circuit circuit;
15
16     // Getter and setter method
17     public Circuit Circuit { get { return circuit; } set { circuit =
18         value; } }
```

```
1  using System;
2  using TMPro;
3  using UnityEngine;
4
5  /// <summary>
6  /// CircuitVisualizer generates meshes for both circuits and custom
7  /// circuits.
8  /// </summary>
9  public class CircuitVisualizer : MonoBehaviour
10 {
11     // Singleton state reference
12     private static CircuitVisualizer instance;
13
14     /// <summary>
15     /// The color associated with starting and custom circuits.
16     /// </summary>
17     [SerializeField]
18     Color startingCircuitColor,
19         customCircuitColor;
20
21     /// <summary>
22     /// Thickness of the border surrounding the base of a circuit.
23     /// </summary>
24     [SerializeField]
25     float borderThickness; // Border surrounding the base of the circuit
26
27     /// <summary>
28     /// Square dimensions of an input node.
29     /// </summary>
30     [SerializeField]
31     float inputSize;
32
33     /// <summary>
34     /// Square dimensions of an output node.
35     /// </summary>
36     [SerializeField]
37     float outputSize;
38
39     /// <summary>
40     /// Square dimensions of the power indicator on input and output
41     /// nodes.
42     /// </summary>
43     [SerializeField]
44     float powerSize;
45
46     /// <summary>
47     /// The distance between each input and output node.
48     /// </summary>
49     [SerializeField]
```

```
48     float heightMargins;
49
50     /// <summary>
51     /// The width of all circuits.
52     /// </summary>
53     [SerializeField]
54     float width;
55
56     /// <summary>
57     /// Reference to the display prefab.
58     /// </summary>
59     [SerializeField]
60     GameObject displayRef;
61
62     /// <summary>
63     /// Various materials utilized in circuit creation.
64     /// </summary>
65     [SerializeField]
66     Material baseMaterial, borderMaterial, inputMaterial, outputMaterial, ↩
       powerOffMaterial, powerOnMaterial;
67
68     /// <summary>
69     /// The font to utilize for circuit names.
70     /// </summary>
71     [SerializeField]
72     TMP_FontAsset font;
73
74     /// <summary>
75     /// The padding that should be applied to the text component of a      ↩
       visualized circuit.
76     /// </summary>
77     [SerializeField]
78     Vector2 textPadding;
79
80     /// <summary>
81     /// Refers to the triangles of any generated quad.
82     /// </summary>
83     private readonly int[] triangles = new int[] { 0, 1, 3, 3, 1, 2 };
84
85     /// <summary>
86     /// Refers to the UV of any generated quad.
87     /// </summary>
88     private readonly Vector2[] uv = new Vector2[] { new Vector2(0, 0), new ↩
       Vector2(0, 1), new Vector2(1, 1), new Vector2(1, 0) };
89
90     /// <summary>
91     /// Refers to the normals of any generated quad.
92     /// </summary>
93     private readonly Vector3[] normals = new Vector3[] { Vector3.up,      ↩
```

```
        Vector3.up, Vector3.up, Vector3.up };
```

94 /// <summary>
95 /// Wrapper class allowing for the discovery of an <see
96 /// cref="Circuit.Input"/> through raycasting. ↵
97 /// </summary>
98 public class InputReference : MonoBehaviour
99 {
100 /// <summary>
101 /// The wrapped input.
102 /// </summary>
103 private Circuit.Input input;
104
105 // Getter and setter method
106 public Circuit.Input Input { get { return input; } set { input = ↵
107 value; } }
108
109 /// <summary>
110 /// Wrapper class allowing for the discovery of an <see
111 /// cref="Circuit.Output"/> through raycasting. ↵
112 /// </summary>
113 public class OutputReference : MonoBehaviour
114 {
115 /// <summary>
116 /// The wrapped output.
117 /// </summary>
118 private Circuit.Output output;
119
120 // Getter and setter method
121 public Circuit.Output Output { get { return output; } set { output = ↵
122 value; } }
123
124 // Enforces a singleton state pattern
125 private void Awake()
126 {
127 if (instance != null)
128 {
129 Destroy(this);
130 throw new Exception("CircuitVisualizer instance already ↵
131 established; terminating.");
132 }
133
134 instance = this;
135
136 /// <summary>
137 /// Restores a serialized connection back to the scene.

```
137     /// </summary>
138     /// <param name="connection">The serialized connection to restore.</param>
139     /// <returns>The instantiated GameObjects in the form of a <see href="ConnectionSerializerRestorer"/>.</returns>
140     public ConnectionSerializerRestorer VisualizeConnection
141         (ConnectionSerializer connection)
142     {
143         GameObject parentObj = new GameObject("Connection");
144         Vector3 normalOffset = new Vector3(0, -0.125f, 0.5f);
145         Vector3 pivotOffset = new Vector3(0, 0, -0.5f);
146
147         parentObj.transform.position = Vector3.zero;
148
149         // If there is an optimized parent mesh, create it.
150         if (connection.ParentMesh != null) CreateMesh(parentObj,
151             connection.ParentMesh);
152
153         // Runs if the starting wire and ending wire point to the same
154         // GameObject.
155         if (connection.SingleWired)
156         {
157             // Creates the single wire parent
158             GameObject singleWire = new GameObject("Ending Wire");
159
160             singleWire.transform.parent = parentObj.transform;
161             singleWire.transform.position =
162                 connection.EndingMesh.Position;
163             singleWire.transform.eulerAngles =
164                 connection.EndingMesh.Rotation;
165             singleWire.transform.localScale = connection.EndingMesh.Scale;
166
167             // Creates the single wire pivot
168             GameObject pivot = new GameObject("Pivot");
169
170             pivot.transform.parent = singleWire.transform;
171             pivot.transform.localPosition = pivotOffset;
172             pivot.transform.localEulerAngles = Vector3.zero;
173             pivot.transform.localScale = Vector3.one;
174
175             // Creates the single wire mesh
176             GameObject actual = new GameObject("GameObject");
177
178             actual.transform.parent = pivot.transform;
179             actual.transform.localPosition = normalOffset;
180             actual.transform.localEulerAngles = Vector3.zero;
181             actual.transform.localScale = Vector3.one;
182             CreateMesh(actual, connection.EndingMesh);
```

...ct\Assets\Scripts\Shared Scripts\CircuitVisualizer.cs 5

```
179         return new ConnectionSerializerRestorer
180             (connection.CircuitConnectorIdentifier, singleWire,
181              singleWire, parentObj);
182     }
183
184     else
185     {
186
187         // Creates the starting wire parent
188         GameObject startingWire = new GameObject("Starting Wire");
189
190         startingWire.transform.parent = parentObj.transform;
191         startingWire.transform.position =
192             connection.StartingMesh.Position;
193         startingWire.transform.eulerAngles =
194             connection.StartingMesh.Rotation;
195         startingWire.transform.localScale =
196             connection.StartingMesh.Scale;
197
198         // Creates the starting wire pivot
199         GameObject pivot = new GameObject("Pivot");
200
201         pivot.transform.parent = startingWire.transform;
202         pivot.transform.localPosition = Vector3.zero;
203         pivot.transform.localEulerAngles = Vector3.zero;
204         pivot.transform.localScale = Vector3.one;
205
206         // Creates the starting wire mesh
207         GameObject actual = new GameObject("GameObject");
208
209         actual.transform.parent = pivot.transform;
210         actual.transform.localPosition = normalOffset;
211         actual.transform.localEulerAngles = Vector3.zero;
212         actual.transform.localScale = Vector3.one;
213         CreateMesh(actual, connection.StartingMesh);
214
215         // Creates the ending wire parent
216         GameObject endingWire = new GameObject("Ending Wire");
217
218         endingWire.transform.parent = parentObj.transform;
219         endingWire.transform.position =
220             connection.EndingMesh.Position;
221         endingWire.transform.eulerAngles =
222             connection.EndingMesh.Rotation;
223         endingWire.transform.localScale = connection.EndingMesh.Scale;
```

```
221         pivot.transform.localEulerAngles = Vector3.zero;
222         pivot.transform.localScale = Vector3.one;
223
224         // Creates the ending wire mesh
225         actual = new GameObject("GameObject");
226         actual.transform.parent = pivot.transform;
227         actual.transform.localPosition = normalOffset;
228         actual.transform.localEulerAngles = Vector3.zero;
229         actual.transform.localScale = Vector3.one;
230         CreateMesh(actual, connection.EndingMesh);
231
232         return new ConnectionSerializerRestorer
233             (connection.CircuitConnectorIdentifier, startingWire,
234              endingWire, parentObj);
235     }
236
237     /// <summary>
238     /// Generates a circuit GameObject corresponding to its specific
239     /// properties.
240     /// </summary>
241     /// <param name="circuit">The circuit to reference.</param>
242     /// <param name="startingPosition">The starting position of the
243     /// circuit.</param>
244     public void VisualizeCircuit(Circuit circuit, Vector2
245         startingPosition)
246     {
247         // Target circuit is a display; run alternate code
248         if (circuit.GetType() == typeof(Display))
249         {
250             Display display = (Display)circuit;
251             GameObject displayObj = Instantiate(displayRef);
252             DisplayReference displayVals =
253                 displayObj.GetComponent<DisplayReference>();
254             Circuit.Input[] inputs = display.Inputs;
255
256             displayObj.name = display.CircuitName;
257             displayObj.transform.position = new Vector3
258                 (startingPosition.x, GridMaintenance.Instance.GridHeight,
259                  startingPosition.y);
260             display.PhysicalObject = displayObj;
261             display.Pins = displayVals.Pins;
262             display.PreviewPins = displayVals.PreviewPins;
263
264             for (int i = 0; i < 8; i++)
265             {
266                 GameObject currentInput = displayVals.Inputs[i];
267                 InputReference inputReference =
268                     currentInput.AddComponent<InputReference>();
```

```
261             inputReference.Input = inputs[i];
262             inputs[i].Transform = currentInput.transform;
263             inputs[i].StatusRenderer = displayVals.InputStatuses[i];
264         }
265
266         Destroy(displayVals); // Reference script no longer needed ↗
267             after extracting relevant values
268
269         CircuitReference circuitRef = ↗
270             displayObj.AddComponent<CircuitReference>();
271
272         circuitRef.Circuit = circuit;
273         return;
274     }
275
276     // Setting dimensions
277     int numInputMargins = circuit.Inputs.Length + 1, numOutputMargins ↗
278         = circuit.Outputs.Length + 1;
279     float inputHeight = numInputMargins * heightMargins +
280         circuit.Inputs.Length * inputSize;
281     float outputHeight = numOutputMargins * heightMargins +
282         circuit.Outputs.Length * outputSize;
283     Vector2 dimensions = new Vector2(width, Mathf.Max(inputHeight, ↗
284         outputHeight));
285
286     // Creating circuit base
287     GameObject physicalObject = new GameObject("\\" + ↗
288         circuit.CircuitName + "\\ Gate");
289     GameObject baseQuad = new GameObject("Base");
290
291     physicalObject.transform.position = new Vector3
292         (startingPosition.x, GridMaintenance.Instance.GridHeight, ↗
293             startingPosition.y);
294     baseQuad.layer = 8;
295     baseQuad.transform.parent = physicalObject.transform;
296     baseQuad.transform.localPosition = Vector3.up * 0.005f;
297
298     Vector3[] vertices = new Vector3[]
299     {
300         new Vector3(-dimensions.x / 2, 0, -dimensions.y / 2),
301         new Vector3(-dimensions.x / 2, 0, dimensions.y / 2),
302         new Vector3(dimensions.x / 2, 0, dimensions.y / 2),
303         new Vector3(dimensions.x / 2, 0, -dimensions.y / 2)
304     };
305
306     CreateQuad(baseQuad, vertices, baseMaterial);
307
308     // Creating circuit border
```

```
301     GameObject borderQuad = new GameObject("Border");
302
303     borderQuad.layer = 13;
304     borderQuad.transform.parent = physicalObject.transform;
305     borderQuad.transform.localPosition = Vector3.zero;
306     vertices = new Vector3[]
307     {
308         new Vector3(-dimensions.x / 2 - borderThickness, 0, -
309             dimensions.y / 2 - borderThickness),
310         new Vector3(-dimensions.x / 2 - borderThickness, 0,
311             dimensions.y / 2 + borderThickness),
312         new Vector3(dimensions.x / 2 + borderThickness, 0,
313             dimensions.y / 2 + borderThickness),
314         new Vector3(dimensions.x / 2 + borderThickness, 0, -
315             dimensions.y / 2 - borderThickness)
316     };
317     CreateQuad(borderQuad, vertices, borderMaterial, false);
318
319     // Power on/off vertices
320     Vector3[] powerVertices = new Vector3[]
321     {
322         new Vector3(-powerSize / 2, 0, -powerSize / 2),
323         new Vector3(-powerSize / 2, 0, powerSize / 2),
324         new Vector3(powerSize / 2, 0, powerSize / 2),
325         new Vector3(powerSize / 2, 0, -powerSize / 2)
326     };
327
328     // Creating input nodes
329     float inputStepSize = (dimensions.y - circuit.Inputs.Length *
330         inputSize) / numInputMargins;
331     int index = 0;
332
333     vertices = new Vector3[]
334     {
335         new Vector3(-inputSize / 2, 0, -inputSize / 2),
336         new Vector3(-inputSize / 2, 0, inputSize / 2),
337         new Vector3(inputSize / 2, 0, inputSize / 2),
338         new Vector3(inputSize / 2, 0, -inputSize / 2)
339     };
340
341     for (float currentHeight = inputStepSize + inputSize / 2; index < circuit.Inputs.Length; currentHeight += inputStepSize +
342         inputSize)
343     {
344         GameObject inputQuad = new GameObject("Input " + (index + 1));
345         GameObject inputQuadPower = new GameObject("Input Status " +
346             (index + 1));
347         Vector3 pos = new Vector3(-dimensions.x / 2, 0.01f,
348             currentHeight - dimensions.y / 2);
```

```
341             inputQuad.layer = 9;
342             inputQuad.transform.parent = inputQuadPower.transform.parent = ↵
343                 physicalObject.transform;
344             inputQuad.transform.localPosition =
345                 inputQuadPower.transform.localPosition = pos;
346             CreateQuad(inputQuad, vertices, inputMaterial);
347             CreateQuad(inputQuadPower, powerVertices, powerOffMaterial, ↵
348                 false);
349
350             Vector3 temp = inputQuadPower.transform.localPosition;
351
352             temp.y = 0.015f;
353             inputQuadPower.transform.localPosition = temp;
354             circuit.Inputs[index].Transform = inputQuad.transform;
355             circuit.Inputs[index].StatusRenderer =
356                 inputQuadPower.GetComponent<MeshRenderer>();
357
358             InputReference inputReference =
359                 inputQuad.AddComponent<InputReference>();
360
361             inputReference.Input = circuit.Inputs[index];
362             index++;
363         }
364
365         // Creating output nodes
366         float outputStepSize = (dimensions.y - circuit.Outputs.Length * ↵
367             outputSize) / numOutputMargins;
368
369         index = 0;
370         vertices = new Vector3[]
371         {
372             new Vector3(-outputSize / 2, 0, -outputSize / 2),
373             new Vector3(-outputSize / 2, 0, outputSize / 2),
374             new Vector3(outputSize / 2, 0, outputSize / 2),
375             new Vector3(outputSize / 2, 0, -outputSize / 2)
376         };
377
378         for (float currentHeight = outputStepSize + outputSize / 2; index > ↵
379             < circuit.Outputs.Length; currentHeight += outputStepSize + ↵
380             outputSize)
381         {
382             GameObject outputQuad = new GameObject("Output " + (index + ↵
383                 1));
384             GameObject outputQuadPower = new GameObject("Output Status " + ↵
385                 (index + 1));
386             Vector3 pos = new Vector3(dimensions.x / 2, 0.01f, ↵
387                 currentHeight - dimensions.y / 2);
388
389             outputQuad.transform.localPosition =
390                 outputQuadPower.transform.localPosition = pos;
391             CreateQuad(outputQuad, vertices, outputMaterial);
392             CreateQuad(outputQuadPower, powerVertices, powerOffMaterial, ↵
393                 false);
394
395             Vector3 temp = outputQuadPower.transform.localPosition;
396
397             temp.y = 0.015f;
398             outputQuadPower.transform.localPosition = temp;
399             circuit.Outputs[index].Transform = outputQuad.transform;
400             circuit.Outputs[index].StatusRenderer =
401                 outputQuadPower.GetComponent<MeshRenderer>();
402
403             InputReference inputReference =
404                 outputQuad.AddComponent<InputReference>();
405
406             inputReference.Input = circuit.Outputs[index];
407             index++;
408         }
```

```
379         outputQuad.layer = 10;
380         outputQuad.transform.parent = outputQuadPower.transform.parent =>
381             = physicalObject.transform;
382         outputQuad.transform.localPosition =
383             outputQuadPower.transform.localPosition = pos;
384         CreateQuad(outputQuad, vertices, outputMaterial);
385         CreateQuad(outputQuadPower, powerVertices, powerOffMaterial,
386             false);
387
388         Vector3 temp = outputQuadPower.transform.localPosition;
389
390         temp.y = 0.015f;
391         outputQuadPower.transform.localPosition = temp;
392         circuit.Outputs[index].Transform = outputQuad.transform;
393         circuit.Outputs[index].StatusRenderer =
394             outputQuadPower.GetComponent<MeshRenderer>();
395
396         OutputReference outputReference =
397             outputQuad.AddComponent<OutputReference>();
398
399         outputReference.Output = circuit.Outputs[index];
400         index++;
401     }
402
403     // Adding text component
404     GameObject name = new GameObject("Name");
405
406     name.transform.parent = physicalObject.transform;
407     name.transform.localPosition = Vector3.up * 0.01f + Vector3.right =>
408         * (inputSize - outputSize) / 4;
409     name.transform.eulerAngles = Vector3.right * 90;
410
411     Vector2 nameDimensions = new Vector2(dimensions.x - (inputSize +
412         outputSize) / 2 - 2 * textPadding.x, dimensions.y - 2 *
413         textPadding.y);
414     TextMeshPro text = name.AddComponent<TextMeshPro>();
415
416     text.text = circuit.CircuitName;
417     text.rectTransform.sizeDelta = nameDimensions;
418     text.alignment = TextAlignmentOptions.Center;
419     text.enableAutoSizing = true;
420     text.fontSizeMin = 0;
421     text.font = font;
422     text.color = startingCircuitColor;
423
424     circuit.PhysicalObject = physicalObject; // Connects new
425         GameObject to its circuit for future reference.
426
427     CircuitReference circuitReference =
```

```
419         physicalObject.AddComponent<CircuitReference>();
420         circuitReference.Circuit = circuit;
421         circuit.Update();
422     }
423
424     /// <summary>
425     /// Generates a custom circuit GameObject corresponding to its
426     /// specific properties.
427     /// </summary>
428     /// <param name="customCircuit">The custom circuit to reference.</
429     /// param>
430     /// <param name="startingPosition">The starting position of the custom
431     /// circuit.</param>
432     public void VisualizeCustomCircuit(CustomCircuit customCircuit,
433                                         Vector2 startingPosition)
434     {
435         // Setting dimensions
436         int numInputMargins = customCircuit.Inputs.Length + 1,
437             numOutputMargins = customCircuit.Outputs.Length + 1;
438         float inputHeight = numInputMargins * heightMargins +
439             customCircuit.Inputs.Length * inputSize;
440         float outputHeight = numOutputMargins * heightMargins +
441             customCircuit.Outputs.Length * outputSize;
442         Vector2 dimensions = new Vector2(width, Mathf.Max(inputHeight,
443                                               outputHeight));
444
445         // Creating circuit base
446         GameObject physicalObject = new GameObject("\\" +
447             customCircuit.CircuitName + "\\");
448         GameObject baseQuad = new GameObject("Base");
449
450         physicalObject.transform.position = new Vector3
451             (startingPosition.x, GridMaintenance.Instance.GridHeight,
452              startingPosition.y);
453         baseQuad.layer = 8;
454         baseQuad.transform.parent = physicalObject.transform;
455         baseQuad.transform.localPosition = Vector3.up * 0.005f;
456
457         Vector3[] vertices = new Vector3[]
458         {
459             new Vector3(-dimensions.x / 2, 0, -dimensions.y / 2),
460             new Vector3(-dimensions.x / 2, 0, dimensions.y / 2),
461             new Vector3(dimensions.x / 2, 0, dimensions.y / 2),
462             new Vector3(dimensions.x / 2, 0, -dimensions.y / 2)
463         };
464
465         CreateQuad(baseQuad, vertices, baseMaterial);
466     }
467 }
```

```
456     // Creating circuit border
457     GameObject borderQuad = new GameObject("Border");
458
459     borderQuad.layer = 13;
460     borderQuad.transform.parent = physicalObject.transform;
461     borderQuad.transform.localPosition = Vector3.zero;
462     vertices = new Vector3[]
463     {
464         new Vector3(-dimensions.x / 2 - borderThickness, 0, -
465             dimensions.y / 2 - borderThickness),
466         new Vector3(-dimensions.x / 2 - borderThickness, 0,
467             dimensions.y / 2 + borderThickness),
468         new Vector3(dimensions.x / 2 + borderThickness, 0,
469             dimensions.y / 2 + borderThickness),
470         new Vector3(dimensions.x / 2 + borderThickness, 0, -
471             dimensions.y / 2 - borderThickness)
472     };
473     CreateQuad(borderQuad, vertices, borderMaterial, false);
474
475     // Power on/off vertices
476     Vector3[] powerVertices = new Vector3[]
477     {
478         new Vector3(-powerSize / 2, 0, -powerSize / 2),
479         new Vector3(-powerSize / 2, 0, powerSize / 2),
480         new Vector3(powerSize / 2, 0, powerSize / 2),
481         new Vector3(powerSize / 2, 0, -powerSize / 2)
482     };
483
484     // Creating input nodes
485     float inputStepSize = (dimensions.y - customCircuit.Inputs.Length *
486         inputSize) / numInputMargins;
487     int index = 0;
488
489     vertices = new Vector3[]
490     {
491         new Vector3(-inputSize / 2, 0, -inputSize / 2),
492         new Vector3(-inputSize / 2, 0, inputSize / 2),
493         new Vector3(inputSize / 2, 0, inputSize / 2),
494         new Vector3(inputSize / 2, 0, -inputSize / 2)
495     };
496
497     for (float currentHeight = inputStepSize + inputSize / 2; index < customCircuit.Inputs.Length; currentHeight += inputStepSize +
498         inputSize)
499     {
500         GameObject inputQuad = new GameObject("Input " + (index + 1));
501         GameObject inputQuadPower = new GameObject("Input Status " +
502             (index + 1));
503         Vector3 pos = new Vector3(-dimensions.x / 2, 0.01f,
```

```
        currentHeight - dimensions.y / 2);

497     inputQuad.layer = 9;
498     inputQuad.transform.parent = inputQuadPower.transform.parent = ↵
        physicalObject.transform;
499     inputQuad.transform.localPosition =
500         inputQuadPower.transform.localPosition = pos;
501     CreateQuad(inputQuad, vertices, inputMaterial);
502     CreateQuad(inputQuadPower, powerVertices, powerOffMaterial, ↵
        false);

503     Vector3 temp = inputQuadPower.transform.localPosition;
504
505     temp.y = 0.015f;
506     inputQuadPower.transform.localPosition = temp;
507     customCircuit.Inputs[index].Transform = inputQuad.transform;
508     customCircuit.Inputs[index].StatusRenderer = ↵
        inputQuadPower.GetComponent<MeshRenderer>();

509     InputReference inputReference =
510         inputQuad.AddComponent<InputReference>();

511     inputReference.Input = customCircuit.Inputs[index];
512     index++;
513 }
514
515 // Creating output nodes
516 float outputStepSize = (dimensions.y -
517     customCircuit.Outputs.Length * outputSize) / numOutputMargins;
518
519 index = 0;
520 vertices = new Vector3[]
521 {
522     new Vector3(-outputSize / 2, 0, -outputSize / 2),
523     new Vector3(-outputSize / 2, 0, outputSize / 2),
524     new Vector3(outputSize / 2, 0, outputSize / 2),
525     new Vector3(outputSize / 2, 0, -outputSize / 2)
526 };
527
528 for (float currentHeight = outputStepSize + outputSize / 2; index < ↵
529     customCircuit.Outputs.Length; currentHeight += outputStepSize + ↵
        outputSize)
530 {
531     GameObject outputQuad = new GameObject("Output " + (index + ↵
        1));
532     GameObject outputQuadPower = new GameObject("Output Status " + ↵
        (index + 1));
533     Vector3 pos = new Vector3(dimensions.x / 2, 0.01f, ↵
        currentHeight - dimensions.y / 2);
```

```
534
535     outputQuad.layer = 10;
536     outputQuad.transform.parent = outputQuadPower.transform.parent =>
537         = physicalObject.transform;
538     outputQuad.transform.localPosition =
539         outputQuadPower.transform.localPosition = pos;
540     CreateQuad(outputQuad, vertices, outputMaterial);
541     CreateQuad(outputQuadPower, powerVertices, powerOffMaterial,
542         false);
543
544     Vector3 temp = outputQuadPower.transform.localPosition;
545
546     temp.y = 0.015f;
547     outputQuadPower.transform.localPosition = temp;
548     customCircuit.Outputs[index].Transform = outputQuad.transform;
549     customCircuit.Outputs[index].StatusRenderer =
550         outputQuadPower.GetComponent<MeshRenderer>();
551
552     OutputReference outputReference =
553         outputQuad.AddComponent<OutputReference>();
554
555     outputReference.Output = customCircuit.Outputs[index];
556     index++;
557 }
558
559 // Adding text component
560 GameObject name = new GameObject("Name");
561
562 name.transform.parent = physicalObject.transform;
563 name.transform.localPosition = Vector3.up * 0.01f + Vector3.right =>
564     * (inputSize - outputSize) / 4;
565 name.transform.eulerAngles = Vector3.right * 90;
566
567 Vector2 nameDimensions = new Vector2(dimensions.x - (inputSize +
568     outputSize) / 2 - 2 * textPadding.x, dimensions.y - 2 *
569     textPadding.y);
570 TextMeshPro text = name.AddComponent<TextMeshPro>();
571
572 text.text = customCircuit.CircuitName.ToUpper();
573 text.rectTransform.sizeDelta = nameDimensions;
574 text.alignment = TextAlignmentOptions.Center;
575 text.enableAutoSizing = true;
576 text.fontSizeMin = 0;
577 text.font = font;
578 text.color = customCircuitColor;
579
580 customCircuit.PhysicalObject = physicalObject; // Connects new
581     GameObject to its circuit for future reference.
```

```
574     CircuitReference circuitReference =
575         physicalObject.AddComponent<CircuitReference>();
576 
577     circuitReference.Circuit = customCircuit;
578     customCircuit.Connections.transform.parent =
579         customCircuit.PhysicalObject.transform;
580 }
581 
582 /// <summary>
583 /// Special signature of <seealso cref="CreateQuad(GameObject, Vector3 >
584 /// [], Material, bool)"> that always adds a mesh collider.
585 /// </summary>
586 private void CreateQuad(GameObject quad, Vector3[] vertices, Material >
587     material) { CreateQuad(quad, vertices, material, true); }
588 
589 /// <summary>
590 /// Creates a quad from the given mesh data.
591 /// </summary>
592 /// <param name="quad">The GameObject to save the mesh to.</param>
593 /// <param name="vertices">The vertices of the mesh.</param>
594 /// <param name="material">The material of the mesh.</param>
595 /// <param name="addMeshCollider">Whether the mesh should have a mesh >
596     collider for raycasting.</param>
597 private void CreateQuad(GameObject quad, Vector3[] vertices, Material >
598     material, bool addMeshCollider)
599 {
600     Mesh mesh = new Mesh();
601     MeshFilter meshFilter = quad.AddComponent<MeshFilter>();
602     MeshRenderer meshRenderer = quad.AddComponent<MeshRenderer>();
603 
604     mesh.vertices = vertices;
605     mesh.triangles = triangles;
606     mesh.uv = uv;
607     mesh.normals = normals;
608     meshFilter.mesh = mesh;
609     meshRenderer.material = material;
610 
611     if (addMeshCollider) quad.AddComponent<MeshCollider>();
612 }
613 
614 /// <summary>
615 /// Creates a mesh from a given mesh serializer.
616 /// </summary>
617 /// <param name="obj">The GameObject to add the mesh to.</param>
618 /// <param name="ms">The serialized mesh data.</param>
619 private void CreateMesh(GameObject obj, MeshSerializer ms)
```

```
617     {
618         Mesh mesh = new Mesh();
619         MeshFilter meshFilter = obj.AddComponent<MeshFilter>();
620         MeshRenderer meshRenderer = obj.AddComponent<MeshRenderer>();
621
622         // Restores mesh values and GameObject layer
623         meshFilter.mesh = mesh;
624         mesh.vertices = ms.Vertices;
625         mesh.triangles = ms.Triangles;
626         mesh.uv = ms.UV;
627         mesh.normals = msNormals;
628         mesh.RecalculateBounds();
629         meshRenderer.material = powerOffMaterial;
630         obj.AddComponent<MeshCollider>();
631         obj.layer = 11;
632     }
633
634     // Getter methods
635     public static CircuitVisualizer Instance { get { return instance; } }
636
637     public Material InputMaterial { get { return inputMaterial; ; } }
638
639     public Material OutputMaterial { get { return outputMaterial; } }
640
641     public Material PowerOffMaterial { get { return powerOffMaterial; } }
642
643     public Material PowerOnMaterial { get { return powerOnMaterial; } }
644 }
```

```
1  using UnityEngine;
2
3  ///<summary>
4  ///<ConnectionSerializerRestorer stores deserialized GameObject values from >
5  ///<an external <see cref="ConnectionSerializer"/>.<br/><br/>
6  ///<These stored values are then referenced by <see >
7  ///<ref="CircuitVisualizer"/> as the assigned connection is restored to the >
8  ///<scene.>
9  ///</summary>
10 public class ConnectionSerializerRestorer
11 {
12     ///<summary>
13     ///<The <see cref="CircuitConnectorIdentifier"/> pertaining to this >
14     ///<connection.>
15     ///</summary>
16     public CircuitConnectorIdentifier circuitConnectorIdentifier { get; >
17         private set; }
18
19     ///<summary>
20     ///<The GameObject that has been created from the assigned <see >
21     ///<ref="ConnectionSerializer.startingMesh"/>.
22     ///</summary>
23     public GameObject startingWire { get; private set; }
24
25     ///<summary>
26     ///<The GameObject that has been created from the assigned <see >
27     ///<ref="ConnectionSerializer.endingMesh"/>.
28     ///</summary>
29     public GameObject endingWire { get; private set; }
30
31     ///<summary>
32     ///<Instantiates and assigns all relevant values extracted from a <see >
33     ///<ref="ConnectionSerializer"/>.
34     public ConnectionSerializerRestorer(CircuitConnectorIdentifier >
35         circuitConnectorIdentifier, GameObject startingWire, GameObject >
36         endingWire, GameObject parentObject)
37     {
38         this.circuitConnectorIdentifier= circuitConnectorIdentifier;
39         this.startingWire = startingWire;
40         this.endingWire = endingWire;
41         this.parentObject = parentObject;
42     }
43 }
```

39 }

```
1  using System;
2  using UnityEngine;
3
4  /// <summary>
5  /// CursorManager handles the switching of mouse textures within each scene.
6  /// </summary>
7  public class CursorManager : MonoBehaviour
8  {
9      // Singleton state reference
10     private static CursorManager instance;
11
12     /// <summary>
13     /// Texture that is utilized when the cursor is in use.
14     /// </summary>
15     [SerializeField]
16     Texture2D cursorTexture;
17
18     /// <summary>
19     /// Texture that is utilized when the link is in use.
20     /// </summary>
21     [SerializeField]
22     Texture2D linkTexture;
23
24     /// <summary>
25     /// The pixel position the cursor texture is centered around.
26     /// </summary>
27     private static Vector2 cursorPosition = Vector2.zero;
28
29     /// <summary>
30     /// The pixel position the link texture is centered around.
31     /// </summary>
32     private static Vector2 linkPosition = new Vector2(12, 0);
33
34     // Enforces a singleton state pattern
35     private void Awake()
36     {
37         if (instance != null)
38         {
39             Destroy(this);
40             throw new Exception("CircuitManager instance already established; terminating.");
41         }
42
43         instance = this;
44     }
45
46     /// <summary>
47     /// Sets a new mouse texture based on the specified value.
```

```
48     /// </summary>
49     /// <param name="useCursorTexture">Whether the cursor texture should be ↵
50     public static void SetMouseTexture(bool useCursorTexture)
51     {
52         if (useCursorTexture) Cursor.SetCursor(instance.cursorTexture,           ↵
53             cursorPosition, CursorMode.ForceSoftware);
54         else Cursor.SetCursor(instance.linkTexture, linkPosition,           ↵
55             CursorMode.ForceSoftware);
56     }
```

```
1  using TMPro;
2  using UnityEngine;
3
4  ///<summary>
5  ///<summary> FPSCounter displays the current frames-per-second to an assigned text UI.
6  ///</summary>
7  public class FPSCounter : MonoBehaviour
8  {
9      ///<summary>
10     ///<summary> The text that displays the FPS.
11     ///</summary>
12     [SerializeField]
13     TextMeshProUGUI fpsText;
14
15     ///<summary>
16     ///<summary> The frames-per-second.
17     ///</summary>
18     private float fps;
19
20     // Calculates the FPS and writes it to fpsText.
21     private void Update()
22     {
23         fps = (int)(Time.timeScale / Time.smoothDeltaTime);
24         fpsText.text = "FPS: " + fps;
25     }
26 }
```

```
1  using System;
2  using UnityEngine;
3
4  /// <summary>
5  /// GridMaintenance instantiates and maintains the visible grid in the editor and preview scenes.
6  /// </summary>
7  public class GridMaintenance : MonoBehaviour
8  {
9      // Singleton state reference
10     private static GridMaintenance instance;
11
12     /// <summary>
13     /// The global height that all in-scene placements are placed at.<br/>><br/>
14     /// The game calculates the x and z positions by raycasting to an x-z plane located at this height.
15     /// </summary>
16     [SerializeField]
17     float gridHeight;
18
19     /// <summary>
20     /// The prefab that displays the in-scene grid.
21     /// </summary>
22     [SerializeField]
23     GameObject gridReference;
24
25     /// <summary>
26     /// The instantiated grid; a copy of <seealso cref="gridReference"/>.
27     /// </summary>
28     private GameObject grid;
29
30     /// <summary>
31     /// The material of the grid.
32     /// </summary>
33     private Material gridMaterial;
34
35     /// <summary>
36     /// The material texture offset of the grid.<br/><br/>
37     /// Because the grid follows the camera, its material must move opposite to the direction of camera movement to create an illusion of it standing still.
38     /// </summary>
39     private Vector2 materialOffset;
40
41     /// <summary>
42     /// Utilized to calculate the delta position between frames.
43     /// </summary>
44     private Vector3 currentPos;
```

```
45
46     // Enforces a singleton state pattern.
47     private void Awake()
48     {
49         if (instance != null)
50         {
51             Destroy(this);
52             throw new Exception("GridMaintenance instance already
53                             established; terminating.");
54         }
55         instance = this;
56     }
57
58     // Instantiates the grid and its respective values.
59     private void Start()
60     {
61         grid = Instantiate(gridReference);
62         grid.name = "Grid";
63         grid.transform.position = new Vector3(transform.position.x,
64                                         gridHeight, transform.position.z);
64         grid.transform.eulerAngles = Vector3.zero;
65         gridMaterial = grid.GetComponent<MeshRenderer>().material;
66         currentPos = transform.position;
67         materialOffset = gridMaterial.GetTextureOffset("_MainTex");
68     }
69
70     // Obtains the change in position from the last frame and alters
71     // materialOffset by an opposite value.
71     private void Update()
72     {
73         grid.transform.position = new Vector3(transform.position.x,
74                                         gridHeight, transform.position.z);
74
75         Vector3 oldPos = currentPos;
76
77         currentPos = transform.position;
78
79         Vector3 deltaPos = currentPos - oldPos;
80         Vector2 realDeltaPos = new Vector2(deltaPos.x, deltaPos.z);
81
82         materialOffset += realDeltaPos;
83         materialOffset = new Vector2(materialOffset.x % 1, materialOffset.y %
84                                     % 1); // The grid is 1x1, therefore it can be clamped.
84         gridMaterial.SetTextureOffset("_MainTex", materialOffset);
85     }
86
87     // Getter methods
88     public static GridMaintenance Instance { get { return instance; } }
```

```
89
90     public float GridHeight { get { return gridHeight; } }
91 }
```

```
1  using System;
2  using UnityEngine;
3  using UnityEngine.UI;
4
5  /// <summary>
6  /// GuideHandler is assigned to its respective prefab, allowing for      ↵
7  /// transitions between windows and tabs.
8  /// </summary>
9  public class GuideHandler : MonoBehaviour
10 {
11     /// <summary>
12     /// The button color of a selected and unselected tab respectively.
13     /// </summary>
14     [SerializeField]
15     Color selectedTabColor, unselectedTabColor;
16
17     /// <summary>
18     /// The button backgrounds of each window.
19     /// </summary>
20     [SerializeField]
21     Image welcomeWindow, logicGatesWindow, controlsWindow;
22
23     /// <summary>
24     /// The button backgrounds of the tabs within each window.
25     /// </summary>
26     [SerializeField]
27     Image[] welcomeTabs, logicGatesTabs, controlsTabs;
28
29     /// <summary>
30     /// The GameObjects that contains all tabs of each window      ↵
31     /// respectively.
32     /// </summary>
33     [SerializeField]
34     GameObject welcomeTabsParent, logicGatesTabsParent,
35         controlsTabsParent;
36
37     /// <summary>
38     /// The view areas of the tabs within each window.
39     /// </summary>
40     [SerializeField]
41     GameObject[] welcomeTabsViews, logicGatesTabsViews, controlsTabsViews;
42
43     /// <summary>
44     /// The background button of the current window that is viewable.
45     /// </summary>
46     private Image currentWindow;

    /// <summary>
    /// Displays the index of the currently opened tab for each window.
```

```
47     /// </summary>
48     private int currentWelcomeTab = 0, currentLogicGatesTab = 0,
49         currentControlsTab = 0;
50
51     // Initializes the default window as the welcome window.
52     private void Start() { currentWindow = welcomeWindow; }
53
54     /// <summary>
55     /// Switches to a new tab for the current window in use.
56     /// </summary>
57     /// <param name="newTab">The index of the tab to switch to.</param>
58     public void UpdateTab(int newTab)
59     {
60         GameObject currentView, newView; // The view areas to turn off and on respectively.
61         Image currentButton, newButton; // The button backgrounds to color unselected and selected respectively.
62
63         // Populates initialized variables based on current window.
64         if (currentWindow == welcomeWindow)
65         {
66             if (newTab == currentWelcomeTab) return;
67
68             currentButton = welcomeTabs[currentWelcomeTab];
69             newButton = welcomeTabs[newTab];
70             currentView = welcomeTabsViews[currentWelcomeTab];
71             newView = welcomeTabsViews[newTab];
72             currentWelcomeTab = newTab;
73         }
74
75         else if (currentWindow == logicGatesWindow)
76         {
77             if (newTab == currentLogicGatesTab) return;
78
79             currentButton = logicGatesTabs[currentLogicGatesTab];
80             newButton = logicGatesTabs[newTab];
81             currentView = logicGatesTabsViews[currentLogicGatesTab];
82             newView = logicGatesTabsViews[newTab];
83             currentLogicGatesTab = newTab;
84         }
85
86         else if (currentWindow == controlsWindow)
87         {
88             if (newTab == currentControlsTab) return;
89
90             currentButton = controlsTabs[currentControlsTab];
91             newButton = controlsTabs[newTab];
92             currentView = controlsTabsViews[currentControlsTab];
93             newView = controlsTabsViews[newTab];
```

```
93         currentControlsTab = newTab;
94     }
95
96     // Incorrect window currently in use
97     else throw new Exception("Invalid current window.");
98
99     // Updates the obtained values
100    currentButton.color = unselectedTabColor;
101    newButton.color = selectedTabColor;
102    currentView.SetActive(false);
103    newView.SetActive(true);
104}
105
106 /// <summary>
107 /// Switches to a new window.
108 /// </summary>
109 /// <param name="newWindow">The new window to switch to.</param>
110 public void UpdateWindow(Image newWindow)
111 {
112     if (currentWindow == newWindow) return;
113
114     // Updates the button colors.
115     currentWindow.color = unselectedTabColor;
116     newWindow.color = selectedTabColor;
117
118     // Makes the current window invisible.
119     if (currentWindow == welcomeWindow) welcomeTabsParent.SetActive(false); ➔
120
121     else if (currentWindow == logicGatesWindow) ➔
122         logicGatesTabsParent.SetActive(false);
123
124     else if (currentWindow == controlsWindow) ➔
125         controlsTabsParent.SetActive(false);
126
127     else throw new Exception("Invalid current window.");
128
129     // Makes the new window visible.
130     if (newWindow == welcomeWindow) welcomeTabsParent.SetActive(true); ➔
131
132     else if (newWindow == logicGatesWindow) ➔
133         logicGatesTabsParent.SetActive(true);
134
135     else if (newWindow == controlsWindow) controlsTabsParent.SetActive(true); ➔
136
137     else throw new Exception("Invalid new window.");
138
139     currentWindow = newWindow;
```

...Project\Assets\Scripts\Shared Scripts\GuideHandler.cs

```
137     }
138 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using TMPro;
4  using UnityEditor;
5  using UnityEngine;
6  using UnityEngine.Events;
7  using UnityEngine.SceneManagement;
8  using UnityEngine.UI;
9
10 public class TaskbarManager : MonoBehaviour
11 {
12     // Singleton state reference
13     private static TaskbarManager instance;
14
15     /// <summary>
16     /// The color associated with starting and custom circuits
17     /// respectively.
18     /// </summary>
19     [SerializeField]
20     Color startingCircuitColor,
21         customCircuitColor;
22
23     /// <summary>
24     /// Horizontal length of the scroll bar attached to the bookmarks
25     /// menu.
26     /// </summary>
27     [SerializeField]
28     float bookmarkScrollThickness;
29
30     /// <summary>
31     /// Set to visible when an interface is opened.<br/><br/>
32     /// Within the scene, this should be a semi-transparent background
33     /// overlayed onto everything except the currently opened interface.
34     /// </summary>
35     [SerializeField]
36     GameObject background;
37
38     /// <summary>
39     /// The scroll bar belonging to the bookmarks menu.<br/><br/>
40     /// If the size of bookmarks exceeds the view area of the bookmarks
41     /// menu, this scroll bar is set to visible.
42     /// </summary>
43     [SerializeField]
44     GameObject bookmarkScrollbar;
45
46     /// <summary>
47     /// GameObject that all bookmarks are instantiated under.
48     /// </summary>
49     [SerializeField]
```

```
46     GameObject bookmarksPanel;
47
48     /// <summary>
49     /// Referenced when moving the bookmarks menu to the current user      ↵
50     /// mouse position.
51     /// </summary>
52     [SerializeField]
53     GameObject bookmarksScroll;
54
55     /// <summary>
56     /// List of all menus that can be active in the editor scene.
57     /// </summary>
58     [SerializeField]
59     GameObject addMenu, // The menu in which the user can add/bookmark    ↵
60     // circuits.
61     bookmarksMenu, // The menu that displays bookmarked circuits, if    ↵
62     // any.
63     circuitSaveErrorMenu, // The menu indicating the reason a custom    ↵
64     // circuit has failed to create.
65     guide, // The guide prefab.
66     labelMenu, // The menu where the user labels empty inputs/outputs.
67     notifierPanel, // An empty menu without a user-driven exit scheme;    ↵
68     // has UI indicating why (e.g. saving).
69     nullState, // An empty menu without a user-driven exit scheme;    ↵
70     // does not have UI.
71     saveWarning, // The menu prompting the user to save.
72     sceneSaveMenu; // The menu in which the user can either save the    ↵
73     // editor scene or create a custom circuit.
74
75     /// <summary>
76     /// Prefab button for custom circuits within the add menu.
77     /// </summary>
78     [SerializeField]
79     GameObject customBookmarkRef;
80
81     /// <summary>
82     /// Prefab button for any circuit within the bookmarks menu.
83     /// </summary>
84     [SerializeField]
85     GameObject bookmarkRef;
86
87     /// <summary>
88     /// Exits out of <seealso cref="currentMenu"/>.<br/><br/>
89     /// More often than not, an alternate input to achieve the same effect    ↵
90     // is pressing the right mouse button.
91     /// </summary>
92     [SerializeField]
93     KeyCode cancelKey;
```

```
87     /// <summary>
88     /// Parent of all starting and custom circuits buttons within the add ➔
89     /// menu.
89     /// </summary>
90     [SerializeField]
91     RectTransform addCustomPanel,
92     addStartingPanel;
93
94     /// <summary>
95     /// Transform of the border behind the bookmarks menu.
96     /// </summary>
97     [SerializeField]
98     RectTransform bookmarksBorder;
99
100    /// <summary>
101    /// Displays the reason for a custom circuit failing to create.
102    /// </summary>
103    [SerializeField]
104    TextMeshProUGUI circuitErrorText;
105
106    /// <summary>
107    /// Prompts the user to compose a label for an empty input or output.
108    /// </summary>
109    [SerializeField]
110    TextMeshProUGUI labelText;
111
112    /// <summary>
113    /// Utilized with <seealso cref="notifierPanel"/> to display the ➔
113    /// reason why a game has disabled all input to the player.<br/><br/>
114    /// Its primary uses are for when the game is saving as well as when a ➔
114    /// custom circuit is being verified/created.
115    /// </summary>
116    [SerializeField]
117    TextMeshProUGUI notifierText;
118
119    /// <summary>
120    /// The name field with which the user specifies the name of a ➔
120    /// prospective custom circuit.
121    /// </summary>
122    [SerializeField]
123    TMP_InputField circuitNameField;
124
125    /// <summary>
126    /// Whether a custom circuit should be created.<br/>
127    /// If unchecked, then the current editor scene is saved instead.
128    /// </summary>
129    [SerializeField]
130    Toggle circuitToggle;
131
```

```
132     /// <summary>
133     /// The size of the bookmark view area and a bookmark respectively.
134     /// </summary>
135     [SerializeField]
136     Vector2 bookmarkMaskSize,
137     bookmarkSize;
138
139     /// <summary>
140     /// Whether the left mouse button is currently held down whilst in the bookmarks menu.<br/>
141     /// This helps discern whether the initial left mouse button press and release occurred when hovered on UI elements.
142     /// </summary>
143     private bool bookmarksDown;
144
145     /// <summary>
146     /// Whether the game is currently deserializing all bookmarks belonging to the current editor scene.
147     /// </summary>
148     private bool currentlyRestoring;
149
150     /// <summary>
151     /// Whether the bookmarks bar can be opened.<br/>
152     /// This value is false until the cooldown to open the bookmarks bar passes, enabling it again.
153     /// </summary>
154     private bool reopenBookmarks = true;
155
156     /// <summary>
157     /// The menu currently opened within the editor scene.
158     /// </summary>
159     private GameObject currentMenu;
160
161     /// <summary>
162     /// The ID list of bookmarks in the scene.<br/><br/>
163     /// Helps to differentiate whether the bookmark is a starting or custom circuit, since all starting circuits have an ID of -1.
164     /// </summary>
165     private List<int> bookmarkIDs = new List<int>();
166
167     /// <summary>
168     /// The typed list of bookmarks in the scene.
169     /// </summary>
170     private List<Type> bookmarks = new List<Type>();
171
172     // Enforces a singleton state pattern and disables frame-by-frame update calls.
173     private void Awake()
174     {
```

```
175         if (instance != null)
176     {
177         Destroy(this);
178         throw new Exception("TaskbarManager instance already
179             established; terminating.");
180     }
181
182     instance = this;
183     enabled = false;
184 }
185
186 // Contains input listening for each user-controllable control
187 // interface.
188 private void Update()
189 {
190     // No menu is currently opened, skips current frame
191     if (currentMenu == nullState) return;
192
193     // Bookmark control scheme
194     if (currentMenu == bookmarksMenu)
195     {
196         // Registers left mouse button down
197         if (Input.GetMouseButtonDown(0) && !
198             EventSystem.current.IsPointerOverGameObject())
199             { bookmarksDown = true; }
200
201         // Exit scheme (occurs if left mouse button is released, but
202             // not while hovered on UI.
203         else if (Input.GetMouseButtonUp(0) && bookmarksDown)
204             {
205                 if (EventSystem.current.IsPointerOverGameObject())
206                     bookmarksDown = false; else CloseMenu();
207             }
208
209         // Moves bookmark to new mouse position
210         else if (Input.GetMouseButtonDown(1) && !
211             EventSystem.current.IsPointerOverGameObject())
212             UpdateBookmarkPosition();
213
214         // Keyboard exit scheme
215         else if (Input.GetKeyDown(cancelKey)) CloseMenu();
216     }
217
218     // Default control scheme for all other menus allowed to be
219         // existed by the user.
220     else if (currentMenu == addMenu || currentMenu == sceneSaveMenu ||
221             currentMenu == circuitSaveErrorMenu || currentMenu ==
222             saveWarning || currentMenu == guide)
223     {
```

...object\Assets\Scripts\Shared Scripts\TaskbarManager.cs 6

```
213         if (Input.GetKeyDown(cancelKey) || Input.GetMouseButtonUp(1))
214     {
215         if (currentMenu == circuitSaveErrorMenu) ConfirmError();
216         else CloseMenu();
217     }
218 }
219
220 /// <summary>
221 /// Opens the <seealso cref="nullState"/> interface.<br/>
222 /// Essentially disables the taskbar from functioning; a locked state
223 /// that must be manually closed via script.
224 /// </summary>
225 public void NullState() { OpenMenu(false, nullState); }
226
227 /// <summary>
228 /// Updates <seealso cref="circuitSaveErrorMenu"/> after <seealso cref="circuitToggle"/> is pressed; called by pressing an in-scene
229 /// button.
230 /// </summary>
231 public void UpdateSaveToggle()
232 {
233     bool isOn = circuitToggle.isOn;
234
235     circuitNameField.interactable = isOn;
236
237     if (!isOn) circuitNameField.text = "";
238
239     /// <summary>
240     /// Goes back to the menu; called by pressing an in-scene button.
241     /// </summary>
242     public void OpenOptions()
243     {
244         // If the current scene is in the editor, check if the save prompt
245         // should first be displayed. Otherwise (including if in a preview
246         // scene), go back to the menu.
247         if (EditorStructureManager.Instance != null &&
248             EditorStructureManager.Instance.DisplaySavePrompt) OpenMenu(true, saveWarning);
249         else SceneManager.LoadScene(0);
250     }
251
252     /// <summary>
253     /// Goes back to the game menu; called by pressing an in-scene button.
254     /// </summary>
255     public void OpenMenuScene() { SceneManager.LoadScene(0); }
256
257     /// <summary>
```

...object\Assets\Scripts\Shared Scripts\TaskbarManager.cs 7

```
253     /// Opens <seealso cref="labelMenu"/> <see cref="IOAssigner"/> is ↵
        enabled and the user presses LMB on an incomplete empty input/ ↵
        output.
254     /// </summary>
255     /// <param name="isInput"></param>
256     public void OpenLabelMenu(bool isInput)
257     {
258         OpenMenu(true, labelMenu);
259         labelText.text = "compose a label for the selected " + (isInput ? ↵
            "input" : "output");
260     }
261
262     /// <summary>
263     /// Opens <seealso cref="guide"/>; called by pressing an in-scene ↵
        button.
264     /// </summary>
265     public void OpenGuide() { OpenMenu(true, guide); }
266
267     /// <summary>
268     /// Opens <seealso cref="sceneSaveMenu"/>; called by pressing an in- ↵
        scene button.
269     /// </summary>
270     public void OpenSave() { OpenMenu(true, sceneSaveMenu); }
271
272     /// <summary>
273     /// Displays an error message if saving a custom circuit fails.
274     /// </summary>
275     /// <param name="errorMessage">The error message to display.</param>
276     public void CircuitSaveError(string errorMessage)
277     {
278         CloseMenu();
279         circuitErrorText.text = errorMessage;
280         OpenMenu(true, circuitSaveErrorMenu);
281     }
282
283     /// <summary>
284     /// Closes <seealso cref="circuitSaveErrorMenu"/>; can be called by ↵
        pressing an in-scene button.
285     /// </summary>
286     public void ConfirmError()
287     {
288         CloseMenu();
289         OpenSave();
290     }
291
292     /// <summary>
293     /// Confirms <seealso cref="sceneSaveMenu"/> input and either saves ↵
        the editor scene or creates a custom circuit based on <seealso ↵
        cref="circuitToggle"/>.<br/>
```

```
294     /// Called by pressing an in-scene button.  
295     /// </summary>  
296     public void SaveConfirm()  
297     {  
298         CloseMenu();  
299  
300         // Should attempt to create custom circuit  
301         if (circuitToggle.isOn)  
302         {  
303             notifierText.text = "verifying...";  
304             OpenMenu(true, notifierPanel);  
305             PreviewStructureManager.Instance.VerifyPreviewStructure  
306             (circuitNameField.text.ToLower().Trim());  
307         }  
308         // Should save scene  
309         else  
310         {  
311             notifierText.text = "saving scene...";  
312             OpenMenu(true, notifierPanel);  
313             EditorStructureManager.Instance.Serialize();  
314         }  
315     }  
316  
317     /// <summary>  
318     /// Opens <seealso cref="addMenu"/>; called by pressing an in-scene  
319     /// button.  
320     /// </summary>  
321     public void OpenAdd() { OpenMenu(true, addMenu); }  
322  
323     /// <summary>  
324     /// Opens the bookmarks menu.  
325     /// </summary>  
326     public void OpenBookmarks() { OpenBookmarks(false); }  
327  
328     /// <summary>  
329     /// Called when a custom circuit is successfully created.  
330     /// </summary>  
331     public void OnSuccessfulPreviewStructure()  
332     {  
333         circuitNameField.text = "";  
334         CloseMenu();  
335     }  
336  
337     /// <summary>  
338     /// Called when a custom circuit successfully passes validation.  
339     /// </summary>  
340     public void OnSuccessfulPreviewVerification()  
341     {
```

```
341         CloseMenu();
342         OpenMenu(true, notifierPanel);
343         notifierText.text = "creating...";
344     }
345
346     /// <summary>
347     /// Opens the bookmarks menu.<br/>
348     /// If there are no bookmarks to display, this method does nothing.
349     /// </summary>
350     /// <param name="showBackground"></param>
351     public void OpenBookmarks(bool showBackground)
352     {
353         if (bookmarks.Count == 0) return;
354
355         bookmarksDown = false;
356         OpenMenu(showBackground, bookmarksMenu);
357     }
358
359     /// <summary>
360     /// Deserializes all bookmarks stored in the current editor structure.
361     /// </summary>
362     /// <param name="circuitIndeces">The serialized integer to circuit
363     /// identifiers.</param>
364     /// <param name="circuitIDs">The preview structure IDs of each circuit
365     /// (-1 if non-custom).</param>
366     public void RestoreBookmarks(List<int> circuitIndeces, List<int>
367         circuitIDs)
368     {
369         int index = 0;
370
371         currentlyRestoring = true;
372
373         foreach (int circuitIndex in new List<int>(circuitIndeces))
374         {
375             // Is a custom circuit
376             if (circuitIndex != -1)
377             {
378                 Toggle toggle = addStartingPanel.GetChild
379                     (circuitIndex).GetComponentInChildren<Toggle>();
380
381                 toggle.isOn = true;
382                 UpdateBookmarkAll(toggle.gameObject);
383             }
384
385             // Is a starting circuit
386             else AddCustomCircuitPanel(circuitIDs[index], true);
387
388             index++;
389         }
390     }
391 }
```

```
386
387         currentlyRestoring = false;
388     }
389
390     /// <summary>
391     /// Adds all non-bookmarked custom circuits belonging to the current preview structure back to <seealso cref="addMenu"/>.
392     /// </summary>
393     public void RestoreCustomCircuits()
394     {
395         foreach (PreviewStructure previewStructure in MenuSetupManager.Instance.PreviewStructures)
396         {
397             // Is bookmarked, continue.
398             if (bookmarkIDs.Contains(previewStructure.ID)) continue;
399
400             AddCustomCircuitPanel(previewStructure.ID, false);
401         }
402     }
403
404     /// <summary>
405     /// Adds a custom circuit to <seealso cref="addCustomPanel"/>.
406     /// </summary>
407     /// <param name="circuitID">The custom circuit ID.</param>
408     /// <param name="bookmarked">Whether this circuit is bookmarked in the current editor scene.</param>
409     public void AddCustomCircuitPanel(int circuitID, bool bookmarked)
410     {
411         GameObject current = Instantiate(customBookmarkRef,
412                                         addCustomPanel.transform); // Instantiates a prefab copy
412         Toggle toggle = current.GetComponentInChildren<Toggle>();
413         PreviewStructure.PreviewStructureReference reference =
414             current.AddComponent<PreviewStructure.PreviewStructureReference>();
415
415         current.GetComponentInChildren<TextMeshProUGUI>().text =
416             MenuSetupManager.Instance.PreviewStructures
417             [MenuSetupManager.Instance.PreviewStructureIDs.IndexOf
418             (circuitID)].Name;
416         reference.ID = circuitID;
417
418         // Adds listeners required to add and bookmark the custom circuit.
419         current.GetComponentInChildren<Button>().onClick.AddListener
420             (delegate { AddBookmarkCircuit(-1, reference.ID); });
420         toggle.onValueChanged.AddListener(delegate { UpdateBookmark
421             (reference); });
421
422         if (bookmarked)
423         {
```

```
424         toggle.isOn = true;
425         UpdateBookmarkCustom(reference);
426     }
427 }
428
429 /// <summary>
430 /// Identical to <seealso cref="UpdateBookmarkAll(GameObject)" />
431 /// except reserved specifically for <see cref="Toggle"/> calls.<br/>
432 /// <br/>
433 /// This is due to boolean changes within scripting also triggering
434 /// <seealso cref="Toggle.onValueChanged"/> events.
435 /// </summary>
436 /// <param name="obj">The bookmark to update.</param>
437 public void UpdateBookmark(GameObject obj)
438 {
439     // If toggles are being adjusted within the script, do not
440     // continue.
441     if (currentlyRestoring) return;
442
443     UpdateBookmarkAll(obj);
444 }
445
446 /// <summary>
447 /// Identical to <seealso cref="UpdateBookmarkCustom
448 /// (PreviewStructure.PreviewStructureReference)" /> except reserved
449 /// specifically for <see cref="Toggle"/> calls.<br/><br/>
450 /// This is due to boolean changes within scripting also triggering
451 /// <seealso cref="Toggle.onValueChanged"/> events.
452 /// </summary>
453 /// <param name="obj">The bookmark to update.</param>
454 public void UpdateBookmark(PreviewStructure.PreviewStructureReference
455     previewStructureReference)
456 {
457     // If toggles are being adjusted within the script, do not
458     // continue.
459     if (currentlyRestoring) return;
460
461     UpdateBookmarkCustom(previewStructureReference);
462 }
463
464 /// <summary>
465 /// Updates a starting bookmark after it has been bookmarked or
466 /// unbookmarked.
467 /// </summary>
468 /// <param name="obj">The starting bookmark to update.</param>
469 public void UpdateBookmarkAll(GameObject obj)
470 {
471     bool newStatus = obj.GetComponent<Toggle>().isOn;
472     Type type = CircuitType(obj.transform.parent.GetSiblingIndex());
```

```
463
464     // Adds bookmark
465     if (newStatus && !bookmarks.Contains(type))
466     {
467         if (!currentlyRestoring)
468             EditorStructureManager.Instance.DisplaySavePrompt = true;
469
470         EditorStructureManager.Instance.Bookmarks.Add
471             (StartingCircuitIndex(type));
472         bookmarks.Add(type);
473         bookmarkIDs.Add(-1);
474
475         GameObject bookmark = Instantiate(bookmarkRef,
476             bookmarksPanel.transform);
477         Button button = bookmark.GetComponentInChildren<Button>();
478         TextMeshProUGUI text =
479             bookmark.GetComponentInChildren<TextMeshProUGUI>();
480
481         bookmark.name = text.text = obj.transform.parent.name;
482         text.color = startingCircuitColor;
483
484         // Ensures pressing on the bookmark will add its
485         // representative circuit.
486         button.onClick.AddListener(delegate { AddBookmarkCircuit
487             (StartingCircuitIndex(type), -1); });
488     }
489
490     // Deletes bookmark
491     else if (!newStatus && bookmarks.Contains(type))
492     {
493         if (!currentlyRestoring)
494             EditorStructureManager.Instance.DisplaySavePrompt = true;
495
496         int index = bookmarks.IndexOf(type);
497
498         EditorStructureManager.Instance.Bookmarks.Remove
499             (StartingCircuitIndex(type));
500         bookmarks.Remove(type);
501         bookmarkIDs.RemoveAt(index);
502         Destroy(bookmarksPanel.transform.GetChild(index).gameObject);
503     }
504
505     /// <summary>
506     /// Updates a custom bookmark after it has been bookmarked or
507     /// unbookmarked.
508     /// </summary>
509     /// <param name="obj">The custom bookmark to update.</param>
510     public void UpdateBookmarkCustom
```

```
    (PreviewStructure.PreviewStructureReference reference)
503    {
504        bool newStatus = reference.GetComponentInChildren<Toggle>()..isOn;
505        int id =
506            reference.GetComponentInChildren<PreviewStructure.PreviewStructu
507            reReference>().ID;
508
509        // Adds bookmark
510        if (newStatus && !bookmarkIDs.Contains(id))
511        {
512            if (!currentlyRestoring)
513                EditorStructureManager.Instance.DisplaySavePrompt = true;
514
515            EditorStructureManager.Instance.Bookmarks.Add(-1);
516            bookmarks.Add(typeof(CustomCircuit));
517            bookmarkIDs.Add(id);
518
519            GameObject bookmark = Instantiate(bookmarkRef,
520                bookmarksPanel.transform);
521            Button button = bookmark.GetComponentInChildren<Button>();
522            TextMeshProUGUI text =
523                bookmark.GetComponentInChildren<TextMeshProUGUI>();
524
525            bookmark.name = text.text =
526                MenuSetupManager.Instance.PreviewStructures
527                [MenuSetupManager.Instance.PreviewStructureIDs.IndexOf
528                    (id)].Name;
529            text.color = customCircuitColor;
530
531            // Ensures pressing on the bookmark will add its
532            // representative circuit.
533            button.onClick.AddListener(delegate { AddBookmarkCircuit(-1,
534                id); });
535
536        }
537
538        // Deletes bookmark
539        else if (!newStatus && bookmarkIDs.Contains(id))
540        {
541            if (!currentlyRestoring)
542                EditorStructureManager.Instance.DisplaySavePrompt = true;
543
544            int index = bookmarkIDs.IndexOf(id);
545
546            EditorStructureManager.Instance.Bookmarks.RemoveAt(index);
547            bookmarks.RemoveAt(index);
548            bookmarkIDs.Remove(id);
549            Destroy(bookmarksPanel.transform.GetChild(index).gameObject);
550        }
551    }
552}
```

```
540
541     /// <summary>
542     /// Adds a bookmarked circuit based on its circuit type and custom      ↵
543     /// circuit ID (if applicable).
544     /// </summary>
545     /// <param name="circuitType">Index representing the circuit type.</      ↵
546     param>
547     /// <param name="circuitID">The custom circuit ID (-1 if custom).</      ↵
548     param>
549     private void AddBookmarkCircuit(int circuitType, int circuitID)
550     {
551         switch (circuitType)
552         {
553             // Custom circuit
554             case -1:
555                 AddCircuit(new CustomCircuit
556                     (MenuSetupManager.Instance.PreviewStructures
557                     [MenuSetupManager.Instance.PreviewStructureIDs.IndexOf
558                     (circuitID)]));
559                 return;
560             // Input
561             case 0:
562                 AddCircuit(new InputGate());
563                 return;
564             // Display
565             case 1:
566                 AddCircuit(new Display());
567                 return;
568             // Buffer
569             case 2:
570                 AddCircuit(new Buffer());
571                 return;
572             // And gate
573             case 3:
574                 AddCircuit(new AndGate());
575                 return;
576             // NAnd gate
577             case 4:
578                 AddCircuit(new NAndGate());
579                 return;
580             // NOR gate
581             case 5:
582                 AddCircuit(new NOrGate());
```

```
583         case 7:
584             AddCircuit(new OrGate());
585             return;
586             // XOr gate
587         case 8:
588             AddCircuit(new XOrGate());
589             return;
590         }
591     }
592
593     /// <summary>
594     /// Adds a starting circuit to the scene; called by pressing an in-    ↵
595     scene button.
596     /// </summary>
597     /// <param name="startingCircuitIndex">Representative index of the    ↵
598     starting circuit.</param>
599     public void AddStartingCircuit(int startingCircuitIndex) { AddCircuit    ↵
600         GetStartingCircuit(startingCircuitIndex); }
601
602     /// <summary>
603     /// Adds a circuit to the scene.
604     /// </summary>
605     /// <param name="newCircuit">The circuit to add.</param>
606     private void AddCircuit(Circuit newCircuit)
607     {
608         // Cancels any modes that would obstruct the placement process.
609         switch (BehaviorManager.Instance.UnpausedGameState)
610         {
611             case BehaviorManager.GameState.CIRCUIT_MOVEMENT:
612                 BehaviorManager.Instance.CancelCircuitMovement();
613                 break;
614
615             case BehaviorManager.GameState.IO_PRESS:
616                 BehaviorManager.Instance.CancelWirePlacement();
617                 break;
618
619             // Switches to placement mode
620             BehaviorManager.Instance.UnpausedGameState =
621                 BehaviorManager.GameState.CIRCUIT_PLACEMENT;
622             BehaviorManager.Instance.UnpausedStateType =
623                 BehaviorManager.StateType.LOCKED;
624             BehaviorManager.Instance.CircuitPlacement(newCircuit);
625             CloseMenu();
626         }
627
628         /// <summary>
629         /// Obtains the circuit type based on its circuit index.
630         /// </summary>
```

```
627     /// <param name="circuitIndex">The index of the circuit.</param>
628     /// <returns>The type of the circuit.</returns>
629     private Type CircuitType(int circuitIndex)
630     {
631         switch (circuitIndex)
632         {
633             case -1:
634                 return typeof(CustomCircuit);
635             case 0:
636                 return typeof(InputGate);
637             case 1:
638                 return typeof(Display);
639             case 2:
640                 return typeof(Buffer);
641             case 3:
642                 return typeof(AndGate);
643             case 4:
644                 return typeof(NAndGate);
645             case 5:
646                 return typeof(NOrGate);
647             case 6:
648                 return typeof(NotGate);
649             case 7:
650                 return typeof(OrGate);
651             case 8:
652                 return typeof(XOrGate);
653             default:
654                 throw new Exception("Invalid starting circuit index.");
655         }
656     }
657
658     /// <summary>
659     /// Obtains the index representation of a starting circuit.
660     /// </summary>
661     /// <param name="circuitType">The type of the starting circuit.</param>
662     /// <returns>The index representation of the circuit.</returns>
663     private int StartingCircuitIndex(Type circuitType)
664     {
665         if (circuitType == typeof(InputGate)) return 0;
666
667         else if (circuitType == typeof(Display)) return 1;
668
669         else if (circuitType == typeof(Buffer)) return 2;
670
671         else if (circuitType == typeof(AndGate)) return 3;
672
673         else if (circuitType == typeof(NAndGate)) return 4;
674     }
```

```
675         else if (circuitType == typeof(NOrGate)) return 5;
676
677         else if (circuitType == typeof(NotGate)) return 6;
678
679         else if (circuitType == typeof(OrGate)) return 7;
680
681         else if (circuitType == typeof(XOrGate)) return 8;
682
683         else throw new Exception("Invalid starting circuit type.");
684     }
685
686     /// <summary>
687     /// Opens a menu.
688     /// </summary>
689     /// <param name="showBackground">Whether <seealso cref="background"/> ↵
690     should be visible.</param>
691     /// <param name="newMenu">The menu to open.</param>
692     private void OpenMenu(bool showBackground, GameObject newMenu)
693     {
694         // If another menu is open, do nothing.
695         if (currentMenu != null && currentMenu != bookmarksMenu) return;
696
697         // Close the bookmarks menu if another menu is opened.
698         if (currentMenu == bookmarksMenu) CloseMenu();
699
700         currentMenu = newMenu;
701
702         // If applicable, the bookmarks menu should open around the user's ↵
703         // cursor.
704         if (newMenu == bookmarksMenu)
705         {
706             UpdateBookmarkPosition();
707             UpdateBookmarkScroll();
708
709             BehaviorManager.Instance.LockUI = true;
710             background.SetActive(showBackground); currentMenu.SetActive(true);
711             enabled = true; // Enables the frame-by-frame listener.
712         }
713
714         /// <summary>
715         /// Updates the size of the bookmarks menu and enables/disables the ↵
716         /// scroll bar.
717         /// </summary>
718         private void UpdateBookmarkScroll()
719         {
720             // If the bookmarks menu does not show all bookmarked circuits, the ↵
721             // vertical scroll bar should appear.
722             bool exceededViewport = bookmarkSize.y * bookmarks.Count > ↵
```

```
    bookmarkMaskSize.y;

720    if (exceededViewport)
721    {
722        // If large enough to scroll, always starts at top of options ↵
723        // list
724        bookmarksPanel.GetComponent<RectTransform>().anchoredPosition ↵
725        *= Vector2.right;
726        bookmarksPanel.GetComponent<RectTransform>().sizeDelta = ↵
727        Vector2.right * (bookmarkSize.x + bookmarkScrollThickness);
728        bookmarksBorder.sizeDelta = new Vector2(bookmarkSize.x + ↵
729        bookmarkScrollThickness, Mathf.Clamp(bookmarks.Count * ↵
730        bookmarkSize.y, 0, bookmarkMaskSize.y));
731    }
732    else
733    {
734        bookmarksPanel.GetComponent<RectTransform>().sizeDelta = ↵
735        Vector2.right * bookmarkSize.x;
736        bookmarksBorder.sizeDelta = new Vector2(bookmarkSize.x, ↵
737        Mathf.Clamp(bookmarks.Count * bookmarkSize.y, 0, ↵
738        bookmarkMaskSize.y));
739    }
740    bookmarkScrollbar.SetActive(exceededViewport);
741
742    /// <summary>
743    /// Moves the bookmarks menu to the current position of the mouse.
744    /// </summary>
745    private void UpdateBookmarkPosition()
746    {
747        RectTransform bottomLeftPos =
748            bookmarksScroll.GetComponent<RectTransform>();
749        Vector2 currentPosition = Input.mousePosition;
750
751        currentPosition.x -= bookmarkSize.x / 2;
752
753        float downVal = bookmarkMaskSize.y - (bookmarkSize.y / 2 * ↵
754            bookmarks.Count);
755
756        currentPosition.y -= Mathf.Clamp(downVal, bookmarkMaskSize.y / 2, ↵
757            bookmarkMaskSize.y);
758        bottomLeftPos.anchoredPosition = currentPosition; // Moves all ↵
759        // bookmarks to the new position
760
761        Vector2 borderPosition = currentPosition;
```

```
755
756         borderPosition.y += Mathf.Clamp(0, downVal - bookmarks.Count *      ↵
757             bookmarkSize.y / 2, bookmarkMaskSize.y);
758     bookmarksBorder.anchoredPosition = borderPosition; // Moves the      ↵
759     // bookmarks border to the new position
760 }
761 /// <summary>
762 /// Closes the currently opened menu.
763 /// </summary>
764 public void CloseMenu()
765 {
766     BehaviorManager.Instance.LockUI = false;
767     reopenBookmarks = false;
768     Invoke("UnlockUI", 0.1f);
769     background.SetActive(false); currentMenu.SetActive(false);
770
771     if (currentMenu == addMenu) addStartingPanel.anchoredPosition =      ↵
772         addCustomPanel.anchoredPosition = Vector2.zero;
773
774     currentMenu = null;
775     enabled = false;
776 }
777 /// <summary>
778 /// Allows the bookmarks menu to be opened; called by invokement      ↵
779 /// within this script.
780 /// </summary>
781 private void UnlockUI() { reopenBookmarks = true; }
782
783 /// <summary>
784 /// Creates a starting circuit from its index representation.
785 /// </summary>
786 /// <param name="startingCircuitIndex">Index of the starting      ↵
787 /// circuit.</param>
788 /// <returns>The newly created circuit.</returns>
789 private Circuit GetStartingCircuit(int startingCircuitIndex)
790 {
791     switch (startingCircuitIndex)
792     {
793         case 0:
794             return new InputGate();
795         case 1:
796             return new Display();
797         case 2:
798             return new Buffer();
```

```
799             return new NAndGate();  
800         case 5:  
801             return new NOrGate();  
802         case 6:  
803             return new NotGate();  
804         case 7:  
805             return new OrGate();  
806         case 8:  
807             return new XOrGate();  
808         default:  
809             throw new Exception("Invalid starting circuit index.");  
810     }  
811 }  
812  
813 /// <summary>  
814 /// Serializes the current editor scene; called by pressing an in-      ↵  
     scene button.  
815 /// </summary>  
816 public void Serialize() { EditorStructureManager.Instance.Serialize      ↵  
    (); }  
817  
818 // Getter methods  
819 public static TaskbarManager Instance { get { return instance; } }  
820  
821 public bool ReopenBookmarks { get { return reopenBookmarks; } }  
822  
823 public GameObject CurrentMenu { get { return currentMenu; } }  
824  
825 public List<int> BookmarkIDs { get { return bookmarkIDs; } }  
826 }
```