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

    /// <summary>
    /// Wrapper class allowing for the discovery of an <see cref="Circuit.Input"/> through raycasting.
    /// </summary>
    public class InputReference : MonoBehaviour
    {
        /// <summary>
        /// The wrapped input.
        /// </summary>
        private Circuit.Input input;

        // Getter and setter method
        public Circuit.Input Input { get { return input; } set { input = value; } }
    }

    /// <summary>
    /// Wrapper class allowing for the discovery of an <see cref="Circuit.Output"/> through raycasting.
    /// </summary>
    public class OutputReference : MonoBehaviour
    {
        /// <summary>
        /// The wrapped output.
        /// </summary>
        private Circuit.Output output;

        // Getter and setter method
        public Circuit.Output Output { get { return output; } set { output = value; } }
    }

    // Enforces a singleton state pattern
    private void Awake()
    {
        if (instance != null)
        {
            Destroy(this);
            throw new Exception("CircuitVisualizer instance already established; terminating.");
        }

        instance = this;
    }

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

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

```
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
                  (connection.CircuitConnectorIdentifier, startingWire,
                  endingWire, parentObj);
233         }
234     }
235
236     /// <summary>
237     /// Generates a circuit GameObject corresponding to its specific
        properties.
238     /// </summary>
239     /// <param name="circuit">The circuit to reference.</param>
240     /// <param name="startingPosition">The starting position of the
        circuit.</param>
241     public void VisualizeCircuit(Circuit circuit, Vector2
        startingPosition)
242     {
243         // Target circuit is a display; run alternate code
244         if (circuit.GetType() == typeof(Display))
245         {
246             Display display = (Display)circuit;
247             GameObject displayObj = Instantiate(displayRef);
248             DisplayReference displayVals =
                  displayObj.GetComponent<DisplayReference>();
249             Circuit.Input[] inputs = display.Inputs;
250
251             displayObj.name = display.CircuitName;
252             displayObj.transform.position = new Vector3
                  (startingPosition.x, GridMaintenance.Instance.GridHeight,
                  startingPosition.y);
253             display.PhysicalObject = displayObj;
254             display.Pins = displayVals.Pins;
255             display.PreviewPins = displayVals.PreviewPins;
256
257             for (int i = 0; i < 8; i++)
258             {
259                 GameObject currentInput = displayVals.Inputs[i];
260                 InputReference inputReference =
                      currentInput.AddComponent<InputReference>();
```

```
261
262                 inputReference.Input = inputs[i];
263                 inputs[i].Transform = currentInput.transform;
264                 inputs[i].StatusRenderer = displayVals.InputStatuses[i];
265             }
266
267             Destroy(displayVals); // Reference script no longer needed
                  after extracing relevant values
268
269             CircuitReference circuitRef =
                  displayObj.AddComponent<CircuitReference>();
270
271             circuitRef.Circuit = circuit;
272             return;
273         }
274
275         // Setting dimensions
276         int numInputMargins = circuit.Inputs.Length + 1, numOutputMargins
              = circuit.Outputs.Length + 1;
277         float inputHeight = numInputMargins * heightMargins +
              circuit.Inputs.Length * inputSize;
278         float outputHeight = numOutputMargins * heightMargins +
              circuit.Outputs.Length * outputSize;
279         Vector2 dimensions = new Vector2(width, Mathf.Max(inputHeight,
              outputHeight));
280
281         // Creating circuit base
282         GameObject physicalObject = new GameObject("\"" +
              circuit.CircuitName + "\" Gate");
283         GameObject baseQuad = new GameObject("Base");
284
285         physicalObject.transform.position = new Vector3
              (startingPosition.x, GridMaintenance.Instance.GridHeight,
              startingPosition.y);
286         baseQuad.layer = 8;
287         baseQuad.transform.parent = physicalObject.transform;
288         baseQuad.transform.localPosition = Vector3.up * 0.005f;
289
290         Vector3[] vertices = new Vector3[]
291         {
292             new Vector3(-dimensions.x / 2, 0, -dimensions.y / 2),
293             new Vector3(-dimensions.x / 2, 0, dimensions.y / 2),
294             new Vector3(dimensions.x / 2, 0, dimensions.y / 2),
295             new Vector3(dimensions.x / 2, 0, -dimensions.y / 2)
296         };
297
298         CreateQuad(baseQuad, vertices, baseMaterial);
299
300         // 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, -
                dimensions.y / 2 - borderThickness),
309             new Vector3(-dimensions.x / 2 - borderThickness, 0,
                dimensions.y / 2 + borderThickness),
310             new Vector3(dimensions.x / 2 + borderThickness, 0,
                dimensions.y / 2 + borderThickness),
311             new Vector3(dimensions.x / 2 + borderThickness, 0, -
                dimensions.y / 2 - borderThickness)
312         };
313         CreateQuad(borderQuad, vertices, borderMaterial, false);
314
315         // Power on/off vertices
316         Vector3[] powerVertices = new Vector3[]
317         {
318             new Vector3(-powerSize / 2, 0, -powerSize / 2),
319             new Vector3(-powerSize / 2, 0, powerSize / 2),
320             new Vector3(powerSize / 2, 0, powerSize / 2),
321             new Vector3(powerSize / 2, 0, -powerSize / 2)
322         };
323
324         // Creating input nodes
325         float inputStepSize = (dimensions.y - circuit.Inputs.Length *
              inputSize) / numInputMargins;
326         int index = 0;
327
328         vertices = new Vector3[]
329         {
330                 new Vector3(-inputSize / 2, 0, -inputSize / 2),
331                 new Vector3(-inputSize / 2, 0, inputSize / 2),
332                 new Vector3(inputSize / 2, 0, inputSize / 2),
333                 new Vector3(inputSize / 2, 0, -inputSize / 2)
334         };
335
336         for (float currentHeight = inputStepSize + inputSize / 2; index <
              circuit.Inputs.Length; currentHeight += inputStepSize +
              inputSize)
337         {
338             GameObject inputQuad = new GameObject("Input " + (index + 1));
339             GameObject inputQuadPower = new GameObject("Input Status " +
                (index + 1));
340             Vector3 pos = new Vector3(-dimensions.x / 2, 0.01f,
                currentHeight - dimensions.y / 2);
```

```
341
342             inputQuad.layer = 9;
343             inputQuad.transform.parent = inputQuadPower.transform.parent =
                    physicalObject.transform;
344             inputQuad.transform.localPosition =
                  inputQuadPower.transform.localPosition = pos;
345             CreateQuad(inputQuad, vertices, inputMaterial);
346             CreateQuad(inputQuadPower, powerVertices, powerOffMaterial,
                  false);
347
348             Vector3 temp = inputQuadPower.transform.localPosition;
349
350             temp.y = 0.015f;
351             inputQuadPower.transform.localPosition = temp;
352             circuit.Inputs[index].Transform = inputQuad.transform;
353             circuit.Inputs[index].StatusRenderer =
                  inputQuadPower.GetComponent<MeshRenderer>();
354
355             InputReference inputReference =
                  inputQuad.AddComponent<InputReference>();
356
357             inputReference.Input = circuit.Inputs[index];
358             index++;
359         }
360
361         // Creating output nodes
362         float outputStepSize = (dimensions.y - circuit.Outputs.Length *
              outputSize) / numOutputMargins;
363
364         index = 0;
365         vertices = new Vector3[]
366         {
367                 new Vector3(-outputSize / 2, 0, -outputSize / 2),
368                 new Vector3(-outputSize / 2, 0, outputSize / 2),
369                 new Vector3(outputSize / 2, 0, outputSize / 2),
370                 new Vector3(outputSize / 2, 0, -outputSize / 2)
371         };
372
373         for (float currentHeight = outputStepSize + outputSize / 2; index
              < circuit.Outputs.Length; currentHeight += outputStepSize +
              outputSize)
374         {
375             GameObject outputQuad = new GameObject("Output " + (index +
                  1));
376             GameObject outputQuadPower = new GameObject("Output Status " +
                    (index + 1));
377             Vector3 pos = new Vector3(dimensions.x / 2, 0.01f,
                  currentHeight - dimensions.y / 2);
378
```

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

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

```
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, -
                dimensions.y / 2 - borderThickness),
465             new Vector3(-dimensions.x / 2 - borderThickness, 0,
                dimensions.y / 2 + borderThickness),
466             new Vector3(dimensions.x / 2 + borderThickness, 0,
                dimensions.y / 2 + borderThickness),
467             new Vector3(dimensions.x / 2 + borderThickness, 0, -
                dimensions.y / 2 - borderThickness)
468         };
469         CreateQuad(borderQuad, vertices, borderMaterial, false);
470
471         // Power on/off vertices
472         Vector3[] powerVertices = new Vector3[]
473         {
474             new Vector3(-powerSize / 2, 0, -powerSize / 2),
475             new Vector3(-powerSize / 2, 0, powerSize / 2),
476             new Vector3(powerSize / 2, 0, powerSize / 2),
477             new Vector3(powerSize / 2, 0, -powerSize / 2)
478         };
479
480         // Creating input nodes
481         float inputStepSize = (dimensions.y - customCircuit.Inputs.Length
              * inputSize) / numInputMargins;
482         int index = 0;
483
484         vertices = new Vector3[]
485         {
486                 new Vector3(-inputSize / 2, 0, -inputSize / 2),
487                 new Vector3(-inputSize / 2, 0, inputSize / 2),
488                 new Vector3(inputSize / 2, 0, inputSize / 2),
489                 new Vector3(inputSize / 2, 0, -inputSize / 2)
490         };
491
492         for (float currentHeight = inputStepSize + inputSize / 2; index <
              customCircuit.Inputs.Length; currentHeight += inputStepSize +
              inputSize)
493         {
494             GameObject inputQuad = new GameObject("Input " + (index + 1));
495             GameObject inputQuadPower = new GameObject("Input Status " +
                (index + 1));
496             Vector3 pos = new Vector3(-dimensions.x / 2, 0.01f,
```

```
                currentHeight - dimensions.y / 2);
497
498             inputQuad.layer = 9;
499             inputQuad.transform.parent = inputQuadPower.transform.parent =
                    physicalObject.transform;
500             inputQuad.transform.localPosition =
                  inputQuadPower.transform.localPosition = pos;
501             CreateQuad(inputQuad, vertices, inputMaterial);
502             CreateQuad(inputQuadPower, powerVertices, powerOffMaterial,
                  false);
503
504             Vector3 temp = inputQuadPower.transform.localPosition;
505
506             temp.y = 0.015f;
507             inputQuadPower.transform.localPosition = temp;
508             customCircuit.Inputs[index].Transform = inputQuad.transform;
509             customCircuit.Inputs[index].StatusRenderer =
                  inputQuadPower.GetComponent<MeshRenderer>();
510
511             InputReference inputReference =
                  inputQuad.AddComponent<InputReference>();
512
513             inputReference.Input = customCircuit.Inputs[index];
514             index++;
515         }
516
517         // Creating output nodes
518         float outputStepSize = (dimensions.y -
              customCircuit.Outputs.Length * outputSize) / numOutputMargins;
519
520         index = 0;
521         vertices = new Vector3[]
522         {
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                 new Vector3(outputSize / 2, 0, -outputSize / 2)
527         };
528
529         for (float currentHeight = outputStepSize + outputSize / 2; index
              < 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
                    = physicalObject.transform;
537             outputQuad.transform.localPosition =
                  outputQuadPower.transform.localPosition = pos;
538             CreateQuad(outputQuad, vertices, outputMaterial);
539             CreateQuad(outputQuadPower, powerVertices, powerOffMaterial,
                  false);
540
541             Vector3 temp = outputQuadPower.transform.localPosition;
542
543             temp.y = 0.015f;
544             outputQuadPower.transform.localPosition = temp;
545             customCircuit.Outputs[index].Transform = outputQuad.transform;
546             customCircuit.Outputs[index].StatusRenderer =
                  outputQuadPower.GetComponent<MeshRenderer>();
547
548             OutputReference outputReference =
                  outputQuad.AddComponent<OutputReference>();
549
550             outputReference.Output = customCircuit.Outputs[index];
551             index++;
552         }
553
554         // Adding text component
555         GameObject name = new GameObject("Name");
556
557         name.transform.parent = physicalObject.transform;
558         name.transform.localPosition = Vector3.up * 0.01f + Vector3.right
              * (inputSize - outputSize) / 4;
559         name.transform.eulerAngles = Vector3.right * 90;
560
561         Vector2 nameDimensions = new Vector2(dimensions.x - (inputSize +
              outputSize) / 2 - 2 * textPadding.x, dimensions.y - 2 *
              textPadding.y);
562         TextMeshPro text = name.AddComponent<TextMeshPro>();
563
564         text.text = customCircuit.CircuitName.ToUpper();
565         text.rectTransform.sizeDelta = nameDimensions;
566         text.alignment = TextAlignmentOptions.Center;
567         text.enableAutoSizing = true;
568         text.fontSizeMin = 0;
569         text.font = font;
570         text.color = customCircuitColor;
571
572         customCircuit.PhysicalObject = physicalObject; // Connects new
              GameObject to its circuit for future reference.
573
```

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