```
1  using System;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  /// <summary>
6  /// CircuitConnector facilitates the connection process between circuits
     in the scene editor.
7  /// </summary>
8  public class CircuitConnector : MonoBehaviour
9  {
10     // Singleton state reference
11     private static CircuitConnector instance;
12
13     /// <summary>
14     /// Reference to the wire prefab.
15     /// </summary>
16     [SerializeField]
17     GameObject wireReference;
18
19     /// <summary>
20     /// The material for powered and unpowered statuses respectively.
21     /// </summary>
22     [SerializeField]
23     Material poweredMaterial, unpoweredMaterial;
24
25     private bool cancelled;
26
27     private Connection currentConnection;
28
29     private GameObject currentWire;
30
31     private int count;
32
33     private Vector3 startingWirePos, currentPos;
34
35     /// <summary>
36     /// Represents a connection from the output circuit to the input
       circuit.
37     /// </summary>
38     public class Connection : MonoBehaviour
39     {
40         /// <summary>
41         /// The input associated with the connection.
42         /// </summary>
43         private Circuit.Input input;
44
45         /// <summary>
46         /// The output associated with the connection.
47         /// </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 completing the connection.
92          bool staringAtIO = Physics.Raycast
              (CameraMovement.Instance.PlayerCamera.ScreenPointToRay
              (Input.mousePosition), out RaycastHit hitInfo) &&
              hitInfo.transform.gameObject.layer ==
              BehaviorManager.Instance.IOLayerCheck;
93
94          // The position to move the end of the wire to.
95          // If hovered onto a valid input or output for completing the connection, it will snap to its position.
96          Vector3 pos = staringAtIO ? hitInfo.transform.position :
              Coordinates.Instance.ModePos;
97
98          pos.y = GridMaintenance.Instance.GridHeight;
99          UpdatePosition(currentWire, currentPos, pos); // Updates the position of the wire.
100
101         // Creates a new pivot as long as the wire is active (has a length >= 0).
102         if (Input.GetMouseButtonDown(0) && currentWire.activeSelf)
103         {
104             count++;
105
106             if (count == 2) startingWirePos = currentPos;
107
108             currentConnection.EndingWire = currentWire;
109
110             // Places a new wire at the current pivot
111             InstantiateWire(currentConnection,
                  Coordinates.Instance.ModePos);
112         }
113     }
114
115     /// <summary>
116     /// Final step in restoring the logic of a serialized connection by initializing a <seealso cref="Connection"/> instance and assigning all of its values.
117     /// </summary>
118     /// <param name="prefab">The base GameObject of the connection.</param>
119     /// <param name="input">The input of the connection.</param>
120     /// <param name="output">The output of the connection.</param>
121     /// <param name="endingWire">The ending wire of the connection.</param>
122     /// <param name="startingWire">The starting wire of the connection.</param>
```

```
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
              (connection); // Re-adds connection for potential serialization
138
139         // If the circuit is an input gate, do not pursue an update call.
140         if (output.ParentCircuit.GetType() == typeof(InputGate)) return;
141
142         Circuit.UpdateCircuit(input, output);
143     }
144
145     // Finalizes the current connection in progress.
146     public static void Connect(Circuit.Input input, Circuit.Output output)
147     {
148         Instance.count = -1;
149         Instance.currentConnection.Input = input;
150         Instance.currentConnection.Output = output;
151         Instance.currentConnection.Input.Connection =
              Instance.currentConnection;
152         Instance.currentConnection.Output.Connections.Add
              (Instance.currentConnection);
153         Instance.currentConnection.Output.ChildInputs.Add(input);
154         Instance.currentConnection.EndingWire.name = "Ending Wire";
155         Instance.OptimizeMeshes();
156         EditorStructureManager.Instance.Connections.Add
              (Instance.currentConnection); // Adds connection for potential
              serialization
157         Instance.currentConnection = null; Instance.currentWire = null;
158         Instance.cancelled = true;
159         Circuit.UpdateCircuit(input, output);
160     }
161
162     /// <summary>
163     /// Removes a connection from the editor scene.
```

```
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); // Removes connection for potential serialization
170         Circuit.UpdateCircuit(false, connection.Input, null);
171         connection.Input.Connection = null;
172         connection.Output.Connections.Remove(connection);
173         connection.Output.ChildInputs.Remove(connection.Input);
174         Destroy(connection.gameObject);
175     }
176 
177     /// <summary>
178     /// Updates all wire materials pertaining to a connection, if applicable.
179     /// </summary>
180     /// <param name="connection"></param>
181     /// <param name="powered"></param>
182     public static void UpdateConnectionMaterial(Connection connection, bool powered)
183     {
184         // If there is an optimized mesh in the wire, update it.
185         if (connection.GetComponent<MeshRenderer>() != null)
              connection.GetComponent<MeshRenderer>().material = powered ? Instance.poweredMaterial : Instance.unpoweredMaterial;
186 
187         // If there is a starting mesh in the wire, update it.
188         if (connection.StartingWire != null)
              connection.StartingWire.GetComponentInChildren<MeshRenderer>().material = powered ? Instance.poweredMaterial : Instance.unpoweredMaterial;
189 
190         // If there is an ending mesh in the wire, update it.
191         if (connection.EndingWire != null)
              connection.EndingWire.GetComponentInChildren<MeshRenderer>().material = powered ? Instance.poweredMaterial : Instance.unpoweredMaterial;
192     }
193 
194     /// <summary>
195     /// Begins the connection process at the specified position.
196     /// </summary>
197     /// <param name="pos"></param>
198     public void BeginConnectionProcess(Vector3 pos)
199     {
200         count = 0;
201         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
              connection.
221         /// </summary>
222         /// <param name="connection">The connection this wire is a part of.</
              param>
223         /// <param name="a">The startring position to instantiate this wire
              at.</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,
                  a.z);
228             currentWire.transform.position = currentPos;
229             currentWire.SetActive(false);
230         }
231
232         /// <summary>
233         /// Specific signature of <seealso cref="UpdatePosition(GameObject,
              Vector3, Vector3, bool)"/> under which isCentered is always false.
234         /// </summary>
235         /// <param name="wire">The wire to move.</param>
236         /// <param name="a">The starting position.</param>
237         /// <param name="b">The ending position.</param>
238         public static void UpdatePosition(GameObject wire, Vector3 a, Vector3
              b) { UpdatePosition(wire, a, b, false); }
239
240         /// <summary>
241         /// Updates the start and end positions of the specified wire.
242         /// </summary>
243         /// <param name="wire">The wire to move.</param>
244         /// <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 =
                  (currentConnection.Input.Transform.position +
                  currentConnection.Output.Transform.position) / 2;
275
276             // Furthermore, the pivot must be altered.
277             currentConnection.StartingWire.transform.GetChild
                  (0).transform.localPosition = Vector3.back * 0.5f;
278
279             // Ensures the height of the wire does not exceed the global
                  grid height
280             Vector3 temp =
                  currentConnection.StartingWire.transform.position;
281
```

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

```
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 ==
                  currentConnection.EndingWire) continue;
327 
328             Destroy(childObj);
329         }
330 
331         MeshFilter combinedMeshFilter =
              currentConnection.gameObject.AddComponent<MeshFilter>();
332 
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
        GameObject.</returns>
343     private Connection InstantiateConnection() { return new GameObject
        ("Connection").AddComponent<Connection>(); }
344 
345     // Getter methods
346     public static CircuitConnector Instance { get { return instance; } }
347 
348     public Connection CurrentConnection { get { return
        currentConnection; } }
349 }
```