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

```
42      /// <summary>
43      /// Static constants representing folder names and file names used for
            serialization.
44      /// </summary>
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
            information.
56      private void Awake()
57      {
58          if (instance != null)
59          {
60              Destroy(this);
61              return;
62          }
63
64          instance = this;
65          DontDestroyOnLoad(this);
66          ImportJSONInformation();
67
68          foreach (Type type in componentsToAdd) gameObject.AddComponent
                (type);
69      }
70
71      /// <summary>
72      /// Deletes a requested editor scene within the running game as well
            as in the save directory.
73      /// </summary>
74      /// <param name="sceneIndex">The editor scene to delete (0-2).</param>
75      public void DeleteEditorStructure(int sceneIndex)
76      {
77          editorStructures[sceneIndex] = null;
78
79          // Obtains the save path pertaining to the requested editor scene.
80          string editorPath = Application.persistentDataPath + "/" +
                editorFolder + "/";
81          string prefabPath = Application.persistentDataPath + "/" +
                editorFolder + "/";
82
83          if (sceneIndex == 0) prefabPath += editorPrefab1Name; else if
                (sceneIndex == 1) prefabPath += editorPrefab2Name; else
```

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

```
120
121     /// <summary>
122     /// Serializes all connections pertaining to either a preview or editor structure.
123     /// </summary>
124     /// <param name="isEditor">Whether the referenced connections belong to an editor scene.</param>
125     /// <param name="generateIndex">The editor scene to update (0-2) if the referenced connections belong to an editor scene.</param>
126     /// <param name="connections">The connections to serialize.</param>
127     public void GenerateConnections(bool isEditor, int generateIndex, List<CircuitConnector.Connection> connections)
128     {
129         // Obtains the path to save all connections to.
130         string path = Application.persistentDataPath + "/" + (isEditor ? editorFolder : previewFolder) + "/";
131
132         if (isEditor)
133         {
134             if (generateIndex == 0) path += editorPrefab1Name; else if (generateIndex == 1) path += editorPrefab2Name; else path += editorPrefab3Name;
135         }
136
137         else
138         {
139             path += previewSubdirectory + generateIndex;
140
141             if (!Directory.Exists(path))
142             {
143                 Directory.CreateDirectory(Application.persistentDataPath + "/" + previewFolder + "/" + previewSubdirectory + generateIndex);
144             }
145         }
146
147         // In case the folder is already populated, all files are cleared from the obtained directory.
148         string[] filePaths = Directory.GetFiles(path);
149
150         foreach (string file in filePaths) File.Delete(file);
151
152         // No point in continuing if there are no connections.
153         if (connections.Count == 0) return;
154
155         int index = 0;
156
157         // Traverses through each connection and generates a corresponding 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 =
                      connection.Input.ParentCircuit.customCircuit;
176 
177                 // Obtains the top-most custom circuit
178                 while (actualCircuit.customCircuit != null) actualCircuit
                      = actualCircuit.customCircuit;
179 
180                 inputCircuitIndex =
                      EditorStructureManager.Instance.Circuits.IndexOf
                      (actualCircuit);
181                 inputIndex = Array.IndexOf(actualCircuit.Inputs,
                      connection.Input);
182             }
183 
184             // Runs if the input belongs to a non-custom circuit
185             else
186             {
187                 inputCircuitIndex =
                      EditorStructureManager.Instance.Circuits.IndexOf
                      (connection.Input.ParentCircuit);
188                 inputIndex = Array.IndexOf
                      (connection.Input.ParentCircuit.Inputs,
                      connection.Input);
189             }
190 
191             // Runs if the output belongs to a custom circuit
```

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

```
225         if (sceneIndex == 0) prefabPath += editorPrefab1Name; else if
              (sceneIndex == 1) prefabPath += editorPrefab2Name; else
              prefabPath += editorPrefab3Name;
226
227         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 =
```

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

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

```
                + editorFolder + "/" + editorPrefab2Name);
        }

        if (!Directory.Exists(Application.persistentDataPath + "/" +
          editorFolder + "/" + editorPrefab3Name))
        {
            Directory.CreateDirectory(Application.persistentDataPath + "/"
               + editorFolder + "/" + editorPrefab3Name);
        }

        string editorPath = Application.persistentDataPath + "/" +
          editorFolder + "/";

        // Ensures the relevant editor JSON save files are created if they
           were removed, otherwise they are loaded into the game.
        if (!File.Exists(editorPath + save1Name))
        {
            File.Create(editorPath + save1Name);
            File.SetAttributes(editorPath + save1Name, fileAttributes);
        }

        else
        {
            editorStructures[0] = JsonUtility.FromJson<EditorStructure>
              (File.ReadAllText(editorPath + save1Name));
        }

        if (!File.Exists(editorPath + save2Name))
        {
            File.Create(editorPath + save2Name);
            File.SetAttributes(editorPath + save2Name, fileAttributes);
        }

        else
        {
            editorStructures[1] = JsonUtility.FromJson<EditorStructure>
              (File.ReadAllText(editorPath + save2Name));
        }

        if (!File.Exists(editorPath + save3Name))
        {
            File.Create(editorPath + save3Name);
            File.SetAttributes(editorPath + save3Name, fileAttributes);
        }

        else
        {
            editorStructures[2] = JsonUtility.FromJson<EditorStructure>
              (File.ReadAllText(editorPath + save3Name));
```

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