

```
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
    have changed after calling <seealso cref="UpdateOutputs"/>.<br/><br/>
    >
45     /// Functionally, if an output does not have its power status change
    before and after an UpdateOutputs() call, it will short circuit and
    not call any circuits it is connected to.
46     /// </summary>
47     private List<Output> outputsToUpdate;
48
49     /// <summary>
50     /// The name of the circuit.
51     /// </summary>
52     private string circuitName;
53
54     /// <summary>
55     /// Input represents all required members of an input node that belong
    to a circuit.<br/><br/>
56     /// An input can only have one connection.
57     /// </summary>
58     public class Input
59     {
60         public Input(Circuit parentCircuit) { this.parentCircuit =
    parentCircuit; }
61
62         /// <summary>
63         /// Whether the input is powered.
64         /// </summary>
65         private bool powered;
66
67         /// <summary>
68         /// The circuit the input composes.
69         /// </summary>
70         private Circuit parentCircuit;
71
72         /// <summary>
73         /// The connection related to the input, if any.
74         /// </summary>
75         private CircuitConnector.Connection connection;
76
77         /// <summary>
78         /// Contains the material that visually displays whether the input
    is powered or not.
79         /// </summary>
80         private MeshRenderer statusRenderer;
81
82         /// <summary>
83         /// The output connecting to the input, if any.
84         /// </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  ➤
    belong to a circuit.<br/><br/>
108 /// An output can more than one connection.
109 /// </summary>
110 public class Output
111 {
112     public Output(Circuit parentCircuit) { this.parentCircuit =  ➤
        parentCircuit; }
113
114     /// <summary>
115     /// Whether the output is powered.
116     /// </summary>
117     private bool powered;
118
119     /// <summary>
120     /// The circuit the output composes.
121     /// </summary>
122     private Circuit parentCircuit;
123
124     /// <summary>
125     /// The connections related to the output, if any.
```

```

126     /// </summary>
127     private List<CircuitConnector.Connection> connections = new
        List<CircuitConnector.Connection>();
128
129     /// <summary>
130     /// The inputs connecting to the output, if any.
131     /// </summary>
132     private List<Input> childInputs = new List<Input>();
133
134     /// <summary>
135     /// Contains the material that visually displays whether the
        output is powered or not.
136     /// </summary>
137     private MeshRenderer statusRenderer;
138
139     /// <summary>
140     /// Transform of the GameObject representing the output, if any.
141     /// </summary>
142     private Transform transform;
143
144     // Getter and setter methods
145     public bool Powered { get { return powered; } set { powered =
        value; } }
146
147     public Circuit ParentCircuit { get { return parentCircuit; } set
        { parentCircuit = value; } }
148
149     public List<CircuitConnector.Connection> Connections { get
        { return connections; } set { connections = value; } }
150
151     public List<Input> ChildInputs { get { return childInputs; } set
        { childInputs = value; } }
152
153     public MeshRenderer StatusRenderer { get { return
        statusRenderer; } set { statusRenderer = value; } }
154
155     public Transform Transform { get { return transform; } set
        { transform = value; } }
156 }
157
158     /// <summary>
159     /// UpdateCall represents an attempt to alter an input node from a
        given output node.<br/><br/>
160     /// An update call does not occur instantly, rather after <seealso
        cref="clockSpeed"/> seconds have passed.<br/>
161     /// This prevents any potential stack overflows caused by loops within
        circuits.
162     /// </summary>
163     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 overridden 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.</
204     param>
205     public Circuit(string circuitName, Vector2 startingPosition) : this
206     (circuitName, 0, 0, startingPosition, false) { }
```

```

...lder\Unity Project\Assets\Scripts\Circuits\Circuit.cs 6
207    /// <param name="numInputs">Number of inputs associated with the  ↗
    circuit.</param>
208    /// <param name="numOutputs">Number of outputs associated with the  ↗
    circuit.</param>
209    /// <param name="startingPosition">Starting position of the circuit.</  ↗
    param>
210    public Circuit(string circuitName, int numInputs, int numOutputs,  ↗
        Vector2 startingPosition) : this(circuitName, numInputs, numOutputs, ↗
            startingPosition, true) { }
211
212    /// <summary>
213    /// Primary constructor that all other constructors reference.
214    /// </summary>
215    /// <param name="circuitName">Name of the circuit.</param>
216    /// <param name="numInputs">Number of inputs associated with the  ↗
    circuit.</param>
217    /// <param name="numOutputs">Number of outputs associated with the  ↗
    circuit.</param>
218    /// <param name="startingPosition">Starting position of the circuit.</  ↗
    param>
219    /// <param name="createIO">Whether each input and output should be  ↗
    initialized.</param>
220    public Circuit(string circuitName, int numInputs, int numOutputs,  ↗
        Vector2 startingPosition, bool createIO)
221    {
222        this.circuitName = circuitName;
223
224        // Initializes inputs and outputs if specified
225        if (createIO)
226        {
227            inputs = new Input[numInputs];
228            outputs = new Output[numOutputs];
229
230            for (int i = 0; i < numInputs; i++) { inputs[i] = new Input  ↗
                (this); }
231
232            for (int i = 0; i < numOutputs; i++) { outputs[i] = new Output ↗
                (this); }
233        }
234
235        /* Determines whether this circuit is meant to be visible.
236        * Within this project, Vector2.PositiveInfinity implicitly  ↗
        defines an invisible circuit.
237        * The only circuits that are invisible are ones that are part of  ↗
        custom circuits.
238        */
239        visible = startingPosition.x != float.PositiveInfinity &&  ↗
            startingPosition.y != float.PositiveInfinity;
240

```

```

...lder\Unity Project\Assets\Scripts\Circuits\Circuit.cs 7
241 // Creates a corresponding mesh if the circuit is visible.
242 if (visible) CircuitVisualizer.Instance.VisualizeCircuit(this, 7
    startingPosition);
243 }
244
245 /// <summary>
246 /// Alternate signature of UpdateCircuit() that assumes the specified 7
    output is not null.
247 /// </summary>
248 /// <param name="input">The input to update.</param>
249 /// <param name="output">The output that caused the update.</param>
250 public static void UpdateCircuit(Input input, Output output) 7
    { UpdateCircuit(output.Powered, input, output); }
251
252 /// <summary>
253 /// Updates the circuit belonging to the specified input based on the 7
    given power status.<br/><br/>
254 /// Afterward, the circuit belonging to the specified input will 7
    update all circuits connected to its output(s).
255 /// </summary>
256 /// <param name="powered">Whether the specified input should be 7
    powered.</param>
257 /// <param name="input">The input to update.</param>
258 /// <param name="output">The output that caused the update.</param>
259 public static void UpdateCircuit(bool powered, Input input, Output 7
    output)
260 {
261     input.Powered = powered;
262
263     // Updates input power status material, if applicable
264     if (input.StatusRenderer != null) input.StatusRenderer.material = 7
        powered ? CircuitVisualizer.Instance.PowerOnMaterial : 7
        CircuitVisualizer.Instance.PowerOffMaterial;
265
266     // Updates the connection wire material associated with the input, 7
        if applicable
267     if (input.Connection != null) 7
        CircuitConnector.UpdateConnectionMaterial(input.Connection, 7
        powered);
268
269     input.ParentOutput = output;
270     input.ParentCircuit.Update();
271     input.ParentCircuit.UpdateChildren();
272 }
273
274 /// <summary>
275 /// Obtains the outputs that should be accessed by <seealso 7
    cref="UpdateChildren"/> as well as updating their <seealso 7
    cref="Output.statusRenderer"/> materials.

```

```
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     /// <summary>
291     /// Calls and updates all connections associated to each valid ↗
292     /// output.<br/><br/>
293     /// This method can be called recursively, i.e. trigger a chain ↗
294     /// reaction.
295     /// </summary>
296     public void UpdateChildren()
297     {
298         List<UpdateCall> updateList = new List<UpdateCall>();
299
300         foreach (Output output in outputsToUpdate)
301         {
302             if (customCircuit != null && ↗
303                 customCircuit.finalOutputs.Contains(output)) ↗
304                 customCircuit.finalOutputs.Remove(output);
305
306             foreach (Input input in output.ChildInputs) updateList.Add(new ↗
307                 UpdateCall(output.Powered, input, output));
308         }
309
310         CircuitCaller.InitiateUpdateCalls(updateList);
311     }
312     /// <summary>
313     /// Updates the materials of each valid output.
314     /// </summary>
315     private void UpdateStatuses()
316     {
317         foreach (Output output in outputsToUpdate)
318         {
319             if (output.StatusRenderer == null) continue;
320
321             output.StatusRenderer.material = output.Powered ? ↗
322                 CircuitVisualizer.Instance.PowerOnMaterial : ↗
```



```
CircuitVisualizer.Instance.PowerOffMaterial;

317     }
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 }
```