```
1  using System;
2  using System.IO;
3  using UnityEngine;
4
5  [Serializable]
6  public class NeuralNetwork
7  {
8      [SerializeField]
9      private Matrix[] Layers;
10     [SerializeField]
11     private Matrix Bias;
12     [SerializeField]
13     private int NumberOfLayers = 4;
14     [SerializeField]
15     private int NodesPerLayer = 12;
16     [SerializeField]
17     private int Inputs = 8;
18     [SerializeField]
19     private int Outputs = 3;
20     [SerializeField]
21     private int Generation;
22
23     public NeuralNetwork(int seed)
24     {
25         Generation = 0;
26         CreateLayers(seed);
27         CreateBias(seed);
28     }
29
30     public NeuralNetwork(string path)
31     {
32         NeuralNetwork net = Load(path);
33         Layers = net.Layers;
34         Bias = net.Bias;
35         NumberOfLayers = net.NumberOfLayers;
36         NodesPerLayer = net.NodesPerLayer;
37         Inputs = net.Inputs;
38         Outputs = net.Outputs;
39         Generation = net.Generation;
40     }
41
42     // The key values that influence the changes made are the generation and ↵
       the placement of the parent network
43     // The higher the generation, the less drastic changes will be made and  ↵
       the better the place,
44     // the less drastic changes will be made
45     public void Modify(int seed, int place)
46     {
47         Generation++;
48         foreach(Matrix layer in Layers)
49         {
50             layer.Modify(seed + Stats.Modifications, place, Generation);
51             Stats.Modifications++;
```

```
52          }
53          Bias.Modify(seed + Stats.Modifications, place, Generation);
54          Stats.Modifications++;
55      }
56
57      public void IncreaseGeneration()
58      {
59          Generation++;
60      }
61
62      private void CreateLayers(int seed)
63      {
64          Layers = new Matrix[NumberOfLayers + 1];
65          // Creates each of the layers. The first and last layers must be
              different sizes in order for the input and
66          // output to work correctly
67          Layers[0] = new Matrix(NodesPerLayer, Inputs);
68          Layers[0].Randomise(seed);
69          Layers[NumberOfLayers] = new Matrix(Outputs, NodesPerLayer);
70          Layers[NumberOfLayers].Randomise(seed);
71          for (int counter = 1; counter < NumberOfLayers; counter++)
72          {
73              Layers[counter] = new Matrix(NodesPerLayer, NodesPerLayer);
74              Layers[counter].Randomise(seed);
75          }
76      }
77
78      private void CreateBias(int seed)
79      {
80          Bias = new Matrix(NodesPerLayer, NumberOfLayers);
81          Bias.Randomise(seed);
82      }
83
84      public int[] Decision(double[] input)
85      {
86          try
87          {
88              // Processes the inputs to get the final matrix
89              Matrix End = RecurseNodes(new Matrix(input,
                    Matrix.VectorType.Column), NumberOfLayers + 1);
90
91              // Turns the matrix into 1's and 0's
92              End.Standardise();
93
94              // Converts the final output into an array of doubles
95              double[] dOutput = End.GetColumn(0);
96              int[] output = new int[dOutput.Length];
97              for (int counter = 0; counter < dOutput.Length; counter++)
98              {
99                  output[counter] = Convert.ToInt32(dOutput[counter]);
100             }
101
102             return output;
```

```
103         }
104         catch (InvalidSize e)
105         {
106             // Resizes the neural network
107             Debug.Log(e);
108             Inputs = input.Length;
109             CreateLayers(Stats.BotsMade);
110             CreateBias(Stats.BotsMade);
111             Stats.BotsMade++;
112             return Decision(input);
113         }
114     }
115
116     private Matrix RecurseNodes(Matrix currentNodes, int recursionsLeft)
117     {
118         // Recurses until 0 left
119         if (recursionsLeft == 0)
120         {
121             // Returns the original inputs (this is the start of processing)
122             return currentNodes;
123         }
124         else if (recursionsLeft == NumberOfLayers + 1)
125         {
126             return CreateFinalNodes(RecurseNodes(currentNodes,
                  recursionsLeft - 1), Layers[recursionsLeft - 1]);
127         }
128         else
129         {
130             // Creates the new nodes using the previous nodes
131             return CreateNodes(RecurseNodes(currentNodes, recursionsLeft -
                  1), Layers[recursionsLeft - 1], Bias.GetColumn(recursionsLeft
                  - 1));
132         }
133     }
134
135     private Matrix CreateNodes(Matrix currentNodes, Matrix weights, double[]
        biases)
136     {
137         // Multiplies the previous nodes by the weights and adds any bias
138         return Matrix.Add(Matrix.Multiply(weights, currentNodes), new Matrix
            (biases, Matrix.VectorType.Column));
139     }
140
141     private Matrix CreateFinalNodes(Matrix currentNodes, Matrix weights)
142     {
143         // Multiplies the previous nodes by the weights
144         return Matrix.Multiply(weights, currentNodes);
145     }
146
147     public int GetGeneration()
148     {
149         return Generation;
150     }
```

```
151
152     public void Save(string path)
153     {
154         // Prepares each matrix for saving
155         foreach (Matrix Layer in Layers)
156         {
157             Layer.PrepareJSON();
158         }
159         Bias.PrepareJSON();
160         string jsonString = JsonUtility.ToJson(this);
161
162         try
163         {
164             StreamWriter sw = new StreamWriter
                  (Application.persistentDataPath + path);
165             sw.Write(jsonString);
166             sw.Close();
167         }
168         catch
169         {
170             Debug.Log("Saving failed");
171         }
172     }
173
174     public static NeuralNetwork Load(string path)
175     {
176         try
177         {
178             StreamReader sr = new StreamReader
                  (Application.persistentDataPath + path);
179             string jsonString = sr.ReadLine();
180             sr.Close();
181             NeuralNetwork net = JsonUtility.FromJson<NeuralNetwork>
                  (jsonString);
182             // Converts each matrix back to normal after loading
183             foreach (Matrix layer in net.Layers)
184             {
185                 layer.ConvertFromJSON();
186             }
187             net.Bias.ConvertFromJSON();
188             return net;
189         }
190         catch
191         {
192             Debug.Log("Loading Failed");
193             try
194             {
195                 // Tries another location
196                 var textFile = Resources.Load<TextAsset>(path);
197                 NeuralNetwork net = JsonUtility.FromJson<NeuralNetwork>
                      (textFile.ToString());
198                 foreach (Matrix layer in net.Layers)
199                 {
```

```
200                     layer.ConvertFromJSON();
201                 }
202                 net.Bias.ConvertFromJSON();
203                 return net;
204             }
205             catch
206             {
207                 // Creates a random neural network
208                 Stats.BotsMade++;
209                 return new NeuralNetwork(Stats.BotsMade - 1);
210             }
211         }
212     }
213 }
```