

```
1 using System;
2 using UnityEngine;
3
4 [Serializable]
5 public class Matrix
6 {
7     private double[,] Values;
8     [SerializeField]
9     private int Rows;
10    [SerializeField]
11    private int Columns;
12    [SerializeField]
13    private double[] ValuesJSON;
14
15    // Returns the appropriate value
16    public double this[int row, int col]
17    {
18        get
19        {
20            return Values[row, col];
21        }
22        set
23        {
24            Values[row, col] = value;
25        }
26    }
27    public enum VectorType
28    {
29        Row,
30        Column
31    }
32
33    public Matrix(int rows, int columns)
34    {
35        Values = new double[rows, columns];
36        Rows = rows;
37        Columns = columns;
38        EmptyMatrix();
39    }
40
41    public Matrix(double[,] values)
42    {
43        Rows = values.GetLength(0);
44        Columns = values.GetLength(1);
45        Values = values;
46    }
47
48    public Matrix(double[] values, VectorType type)
49    {
50        if (type == VectorType.Row)
51        {
52            Rows = 1;
53            Columns = values.Length;
```

```
54         Values = new double[1, Columns];
55         for (int i = 0; i < Columns; i++)
56         {
57             this[0, i] = values[i];
58         }
59     }
60     else if (type == VectorType.Column)
61     {
62         Columns = 1;
63         Rows = values.Length;
64         Values = new double[Rows, 1];
65         for (int i = 0; i < Rows; i++)
66         {
67             this[i, 0] = values[i];
68         }
69     }
70     else
71     {
72         throw new InvalidVector();
73     }
74 }
75
76 // Checks if two matrices are the same
77 public static bool Equal(Matrix a, Matrix b)
78 {
79     if (a.Rows == b.Rows && a.Columns == b.Columns)
80     {
81         for (int i = 0; i < a.Rows; i++)
82         {
83             for (int j = 0; j < a.Columns; j++)
84             {
85                 if (a[i, j] != b[i, j])
86                 {
87                     return false;
88                 }
89             }
90         }
91     }
92     else
93     {
94         return false;
95     }
96     return true;
97 }
98
99 // Changes a matrix randomly
100 public void Modify(int seed, int place, int generation)
101 {
102     double majorBarrier = 80;
103     double weightedChange = place / generation;
104     majorBarrier += 19 * Math.Pow(Math.E, -weightedChange);
105     Stats.Seed *= seed;
106     Stats.Seed += 1;
```

```
107     var random = new System.Random(Stats.Seed / 2);
108     for (int i = 0; i < Rows; i++)
109     {
110         for (int j = 0; j < Columns; j++)
111         {
112             if (random.Next(1, 101) > majorBarrier)
113             {
114                 if (random.Next(1, 101) > 50)
115                 {
116                     Values[i, j] *= 2;
117                 }
118                 else
119                 {
120                     Values[i, j] *= -1;
121                 }
122             }
123             else
124             {
125                 Values[i, j] += (random.NextDouble() - 0.5) * 0.2 *
126                 Values[i, j];
127             }
128         }
129     }
130
131     // Fills a matrix with 0's
132     private void EmptyMatrix()
133     {
134         for (int i = 0; i < Rows; i++)
135         {
136             for (int j = 0; j < Columns; j++)
137             {
138                 this[i, j] = 0;
139             }
140         }
141     }
142
143     // Fills a matrix with random values between -1 and 1
144     public void Randomise(int seed)
145     {
146         System.Random random = new System.Random(seed);
147
148         for (int i = 0; i < Rows; i++)
149         {
150             for (int j = 0; j < Columns; j++)
151             {
152                 this[i, j] = (random.NextDouble() - 0.5) * 2;
153             }
154         }
155     }
156
157     // Multiplies two matrices together
158     public static Matrix Multiply(Matrix left, Matrix right)
```

```
159     {
160         if (left.Columns != right.Rows)
161         {
162             throw new InvalidSize();
163         }
164         double[,] values = new double[left.Rows, right.Columns];
165         for (int i = 0; i < left.Rows; i++)
166         {
167             for (int j = 0; j < right.Columns; j++)
168             {
169                 for (int k = 0; k < left.Columns; k++)
170                 {
171                     values[i, j] += left[i, k] * right[k, j];
172                 }
173             }
174         }
175         return new Matrix(values);
176     }
177
178     // Adds two matrices
179     public static Matrix Add(Matrix a, Matrix b)
180     {
181         if (a.Rows != b.Rows || a.Columns != b.Columns)
182         {
183             throw new InvalidSize();
184         }
185         double[,] values = new double[a.Rows, b.Columns];
186         for (int i = 0; i < a.Rows; i++)
187         {
188             for (int j = 0; j < a.Columns; j++)
189             {
190                 values[i, j] = a[i, j] + b[i, j];
191             }
192         }
193         return new Matrix(values);
194     }
195
196     // Makes every value either 1 or 0
197     public void Standardise()
198     {
199         for (int i = 0; i < Rows; i++)
200         {
201             for (int j = 0; j < Columns; j++)
202             {
203                 if (this[i, j] >= 0)
204                 {
205                     this[i, j] = 1;
206                 }
207                 else
208                 {
209                     this[i, j] = 0;
210                 }
211             }
212         }
213     }
214 }
```

```
212     }
213 }
214
215 // Returns a single column from a matrix
216 public double[] GetColumn(int columnNumber)
217 {
218     if (columnNumber >= Columns)
219     {
220         throw new OutOfBounds();
221     }
222     double[] column = new double[Rows];
223     for (int i = 0; i < Rows; i++)
224     {
225         column[i] = this[i, columnNumber];
226     }
227     return column;
228 }
229
230 // Allows the neural network to be saved as a JSON file since JSON
231 // cannot store 2D arrays
232 public void PrepareJSON()
233 {
234     ValuesJSON = new double[Rows * Columns];
235     int count = 0;
236     for (int i = 0; i < Rows; i++)
237     {
238         for (int j = 0; j < Columns; j++)
239         {
240             ValuesJSON[count] = Values[i, j];
241             count++;
242         }
243     }
244
245 // Converts a matrix back to normal after being loaded from a JSON file
246 public void ConvertFromJSON()
247 {
248     Values = new double[Rows, Columns];
249     int count = 0;
250     for (int i = 0; i < Rows; i++)
251     {
252         for (int j = 0; j < Columns; j++)
253         {
254             Values[i, j] = ValuesJSON[count];
255             count++;
256         }
257     }
258 }
259 }
```