```
using System;
using UnityEngine;

[Serializable]
public class Matrix
{
    private double[,] Values;
    [SerializeField]
    private int Rows;
    [SerializeField]
    private int Columns;
    [SerializeField]
    private double[] ValuesJSON;

    // Returns the appropriate value
    public double this[int row, int col]
    {
        get
        {
            return Values[row, col];
        }
        set
        {
            Values[row, col] = value;
        }
    }
    public enum VectorType
    {
        Row,
        Column
    }

    public Matrix(int rows, int columns)
    {
        Values = new double[rows, columns];
        Rows = rows;
        Columns = columns;
        EmptyMatrix();
    }

    public Matrix(double[,] values)
    {
        Rows = values.GetLength(0);
        Columns = values.GetLength(1);
        Values = values;
    }

    public Matrix(double[] values, VectorType type)
    {
        if (type == VectorType.Row)
        {
            Rows = 1;
            Columns = values.Length;
            Values = new double[1, Columns];
            for (int i = 0; i < Columns; i++)
            {
                this[0, i] = values[i];
            }
        }
        else if (type == VectorType.Column)
        {
            Columns = 1;
            Rows = values.Length;
            Values = new double[Rows, 1];
            for (int i = 0; i < Rows; i++)
            {
                this[i, 0] = values[i];
            }
        }
        else
        {
            throw new InvalidVector();
        }
    }

    // Checks if two matrices are the same
    public static bool Equal(Matrix a, Matrix b)
    {
        if (a.Rows == b.Rows && a.Columns == b.Columns)
        {
            for (int i = 0; i < a.Rows; i++)
            {
                for (int j = 0; j < a.Columns; j++)
                {
                    if (a[i, j] != b[i, j])
                    {
                        return false;
                    }
                }
            }
        }
        else
        {
            return false;
        }
        return true;
    }

    // Changes a matrix randomly
    public void Modify(int seed, int place, int generation)
    {
        double majorBarrier = 80;
        double weightedChange = place / generation;
        majorBarrier += 19 * Math.Pow(Math.E, -weightedChange);
        Stats.Seed *= seed;
        Stats.Seed += 1;
        var random = new System.Random(Stats.Seed / 2);
        for (int i = 0; i < Rows; i++)
        {
            for (int j = 0; j < Columns; j++)
            {
                if (random.Next(1, 101) > majorBarrier)
                {
                    if (random.Next(1, 101) > 50)
                    {
                        Values[i, j] *= 2;
                    }
                    else
                    {
                        Values[i, j] *= -1;
                    }
                }
                else
                {
                    Values[i, j] += (random.NextDouble() - 0.5) * 0.2 *
                   Values[i, j];
                }
            }
        }
    }

    // Fills a matrix with 0's
    private void EmptyMatrix()
    {
        for (int i = 0; i < Rows; i++)
        {
            for (int j = 0; j < Columns; j++)
            {
                this[i, j] = 0;
            }
        }
    }

    // Fills a matrix with random values between -1 and 1
    public void Randomise(int seed)
    {
        System.Random random = new System.Random(seed);

        for (int i = 0; i < Rows; i++)
        {
            for (int j = 0; j < Columns; j++)
            {
                this[i, j] = (random.NextDouble() - 0.5) * 2;
            }
        }
    }

    // Multiplies two matrices together
    public static Matrix Multiply(Matrix left, Matrix right)
    {
        if (left.Columns != right.Rows)
        {
            throw new InvalidSize();
        }
        double[,] values = new double[left.Rows, right.Columns];
        for (int i = 0; i < left.Rows; i++)
        {
            for (int j = 0; j < right.Columns; j++)
            {
                for (int k = 0; k < left.Columns; k++)
                {
                    values[i, j] += left[i, k] * right[k, j];
                }
            }
        }
        return new Matrix(values);
    }

    // Adds two matrices
    public static Matrix Add(Matrix a, Matrix b)
    {
        if (a.Rows != b.Rows || a.Columns != b.Columns)
        {
            throw new InvalidSize();
        }
        double[,] values = new double[a.Rows, b.Columns];
        for (int i = 0; i < a.Rows; i++)
        {
            for (int j = 0; j < a.Columns; j++)
            {
                values[i, j] = a[i, j] + b[i, j];
            }
        }
        return new Matrix(values);
    }

    // Makes every value either 1 or 0
    public void Standardise()
    {
        for (int i = 0; i < Rows; i++)
        {
            for (int j = 0; j < Columns; j++)
            {
                if (this[i, j] >= 0)
                {
                    this[i, j] = 1;
                }
                else
                {
                    this[i, j] = 0;
                }
            }
        }
    }

    // Returns a single column from a matrix
    public double[] GetColumn(int columnNumber)
    {
        if (columnNumber >= Columns)
        {
            throw new OutOfBounds();
        }
        double[] column = new double[Rows];
        for (int i = 0; i < Rows; i++)
        {
            column[i] = this[i, columnNumber];
        }
        return column;
    }

    // Allows the neural network to be saved as a JSON file since JSON
      cannot store 2D arrays
    public void PrepareJSON()
    {
        ValuesJSON = new double[Rows * Columns];
        int count = 0;
        for (int i = 0; i < Rows; i++)
        {
            for (int j = 0; j < Columns; j++)
            {
                ValuesJSON[count] = Values[i, j];
                count++;
            }
        }
    }

    // Converts a matrix back to normal after being loaded from a JSON file
    public void ConvertFromJSON()
    {
        Values = new double[Rows, Columns];
        int count = 0;
        for (int i = 0; i < Rows; i++)
        {
            for (int j = 0; j < Columns; j++)
            {
                Values[i, j] = ValuesJSON[count];
                count++;
            }
        }
    }
}
```