

# Systemy Sztucznej Inteligencji

dokumentacja projektu Kulki XDXXD

Chład Paweł  
Grupa 2D

Meller Bartłomiej  
Grupa 2D

25 maja 2020

# Część I

Opis programu

Instrukcja obsługi

\*

Argumenty wejściowe

**Dodatkowe informacje**

Projekt został skompilowany za pomocą .NET Core 3.1

## Część II

### Opis działania sieci neuronowej

### Zbiór danych

### Implementacja

### Ogólna struktura

### Spis metod

```
public class MovementScoreRule : MonoBehaviour, IScoringRule
{
    • public float GetScore()

    public class Run public class AgentResult
    {
        • public string runName = "Run #N";

        • public event EventHandler<List<AgentResult>> RunComplete;

        • public List<GameObject> agents = new List<GameObject>();

        • public List<AgentResult> results = new List<AgentResult>();

        • public static GameObject agentPrefab;

        • public Run(int num_agents)

        • public Run(List<NetworkModel> models)

        • public void BeginRun()

        • public double score;

        • public NetworkModel model;

        • public AgentResult(double score, NetworkModel model)
    }

    public interface IScoringRule
    {
        •

        public class RunManager : MonoBehaviour
        {
            •

            public class CameraController : MonoBehaviour
            {
                • public Transform target;

                • public Vector3 offset;
            }

            public class ModelManager
```

- `public List<NetworkModel> Models = new List<NetworkModel>();`
- `public int NumModels { get; }`
- `public double LearningRate { get; }`
- `public ModelManager(List<NetworkModel> models, double learningRate = 0.1f)`
- `public void SaveTop(int n)`
- `public void Expand()`

```
public class Agent : MonoBehaviour
```

- `public NetworkModel network;`
- `public static Transform cookieJar;`
- `public Action<Agent> deathCallback;`
- `public List<double> lastInputs = new List<double>();`
- `public List<double> lastOutputs = new List<double>();`
- `public float ViewArc`
- `public void OnCollisionEnter(Collision c)`

```
public class InputMonitor : MonoBehaviour
```

- `public Agent Target;`
- `public Text text;`

```
public static class ScoreCalculator
```

- `public static float CalculateScore(GameObject obj)`

```
public class LifetimeScoreRule : MonoBehaviour, IScoringRule
```

- `public float GetScore()`

```
public class Pulse
```

- `public double Value { get; set; }`

```
public class NeuralLayer
```

- `public List<Neuron> Neurons { get; set; }`
- `public string Name { get; set; }`
- `public double Weight { get; set; }`

- `public NeuralLayer(int count, double initialWeight, Func<double, double> activation, string name = "")`
- `public void Randomize(double lr)`
- `public void Forward()`
- `public override string ToString()`

```
public class Dendrite
```

- `public Pulse InputPulse { get; set; }`
- `public double SynapticWeight { get; set; }`
- `public bool Learnable { get; set; }`
- `public Dendrite()`
- `public void Randomize(double lr)`
- `public override string ToString()`

```
public class Neuron
```

- `public List<Dendrite> Dendrites { get; set; }`
- `public Pulse OutputPulse { get; set; }`
- `public Func<double, double> Activation;`
- `public Neuron(Func<double, double> activation)`
- `public void Randomize(double lr)`
- `public void Fire()`
- `public override string ToString()`

```
public static class ActivationFunc
```

- `public static double Tanh(double x)`
- `public static double Linear(double x)`
- `public static double BinaryStep(double x)`

```
public class NetworkModel
```

- `public List<NeuralLayer> Layers { get; set; }`
- `public NetworkModel()`
- `public NetworkModel DeepCopy()`

- `public void AddLayer(NeuralLayer layer)`
  - `public void Build()`
  - `public void Randomize(double lr)`
  - `public List<double> Decide(List<double> X)`
  - `public void Print()`
  - `public override string ToString()`
- `public static class JsonService`
- `public static void SaveModelsList(List<NetworkModel> models)`
  - `public static List<NetworkModel> LoadModelsList(TextAsset jsonFile)`

Testy

Eksperymenty

# Pełen kod aplikacji

Kod znajduje się w repozytorium pod adresem: [reee jak i również pod spodem](#).

## MovementScoreRule.cs

```
1 using UnityEngine;
2
3
4 //Calculates score based on total moved distance
5 public class MovementScoreRule : MonoBehaviour, IScoringRule
6 {
7
8     float score = 0;
9     Vector3 lastPos;
10
11     void Start()
12     {
13         lastPos = this.transform.position;
14     }
15
16
17     void FixedUpdate()
18     {
19         score += Vector3.Distance(this.transform.position, lastPos);
20     }
21
22
23
24     public float GetScore()
25     {
26         return score;
27     }
28 }
```

---

## Run.cs

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using NeuralNetwork;
5 using UnityEngine;
6 using System.Linq;
7
8
9 //Class that represents data gathered from one simulation run
10 public class Run
11 {
12     public string runName = "Run #N";
13     public event EventHandler<List<AgentResult>> RunComplete;
14     public List<GameObject> agents = new List<GameObject>();
15     public List<AgentResult> results = new List<AgentResult>();
16
17     public static GameObject agentPrefab;
18
19     //Creates a new run with num_agents number of randomly initialized
20     //agents
21     public Run(int num_agents)
22     {
23         for (int i = 0; i < num_agents; i++)
24         {
25             agents.Add(CreateNewAgent());
26         }
27
28         //Creates a new run with agents initialized with given models
29         public Run(List<NetworkModel> models)
30         {
31             foreach (NetworkModel m in models)
32             {
33                 GameObject a = CreateNewAgent();
34                 a.GetComponent<Agent>().network = m;
35                 agents.Add(a);
36             }
37         }
38
39         //Begins run by activating all agents
40         public void BeginRun()
41         {
42             foreach (GameObject a in agents)
43             {
44                 a.gameObject.SetActive(true);
45             }
46         }
47
48         //Called when all agents died
49         private void EndRun()
50         {
51             Debug.Log(runName + " ended");
52             string resultString = "";
```



```

53         for (int i = 0; i < results.Count; i++)
54         {
55             resultString += "Agent #" + i + " | Score: " + results[i].
                    score;
56         }
57         Debug.Log(resultString);
58         agents.Clear();
59         RunComplete(this, results);
60     }
61
62     private void AgentDied(Agent a)
63     {
64         Debug.Log("Agent Died");
65         results.Insert(0, new AgentResult(ScoreCalculator.CalculateScore
                    (a.gameObject), a.network));
66         agents.Remove(a.gameObject);
67         if (agents.Count == 0)
68         {
69             EndRun();
70         }
71     }
72
73
74     //Creates new empty agent, that is inactive
75     private GameObject CreateNewAgent()
76     {
77         GameObject agent = GameObject.Instantiate(agentPrefab);
78         agent.SetActive(false);
79         agent.GetComponent<Agent>().deathCallback = AgentDied;
80         return agent;
81     }
82
83
84     public class AgentResult
85     {
86         public double score;
87         public NetworkModel model;
88
89         public AgentResult(double score, NetworkModel model)
90         {
91             this.score = score;
92             this.model = model.DeepCopy();
93         }
94     }
95 }

```

---

## IScoringRule.cs

```
1 public interface IScoringRule
2 {
3
4     //get current score from component
5     float GetScore();
6
7 }
```

---

## RunManager.cs

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.Linq;
5 using UnityEngine;
6 using NeuralNetwork;
7
8
9 ///
```

```

51     }
52     r.runName = "Run #" + run_num;
53     r.RunComplete += OnRunEnded;
54     r.BeginRun();
55 }
56
57 void OnRunEnded(object sender, List<Run.AgentResult> results)
58 {
59     //Accept only Run senders
60     if (!(sender is Run r))
61         throw new ArgumentException("Sender is not of the type Run")
62         ;
63
64     //Unsubscribe from sender to avoid memory leak
65     r.RunComplete -= OnRunEnded;
66
67     //store run
68     runs.Add(r);
69     List<NetworkModel> models = r.results.OrderBy(x => x.score).
70         Select(x => x.model).ToList();
71     Debug.Log(models[0].ToString());
72     StartNewRun();
73 }

```

---

## CameraController.cs

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5
6 public class CameraController : MonoBehaviour
7 {
8     public Transform target;
9     public Vector3 offset;
10
11     private Vector3 previousMousePos;
12     private Vector3 currRot = new Vector3(0, 0, 0);
13     private float rotSpeed = 100f;
14
15     // Start is called before the first frame update
16     void Start()
17     {
18         previousMousePos = Input.mousePosition;
19         Cursor.lockState = CursorLockMode.Locked;
20     }
21
22     // Update is called once per frame
23     void Update()
24     {
25
26         if (Input.GetButtonDown("ToggleCursor"))
27         {
28             if (Cursor.lockState != CursorLockMode.Locked)
29             {
30                 Cursor.lockState = CursorLockMode.Locked;
31             }
32             else
33             {
34                 Cursor.lockState = CursorLockMode.None;
35             }
36         }
37
38
39         if (Input.GetButtonDown("BreakTarget"))
40             target = null;
41
42         if (Input.GetButtonDown("BestTarget"))
43             target = ChooseBestTarget();
44
45
46         if (target == null)
47         {
48             transform.Translate(Vector3.forward * Input.GetAxis("Forward"));
49             transform.Translate(Vector3.up * Input.GetAxis("Up"));
50             transform.Translate(Vector3.right * Input.GetAxis("Right"));
51
52             Quaternion rot = Quaternion.identity;
```

```
53         currRot += new Vector3(-Input.GetAxis("Mouse Y"), Input.  
54             GetAxis("Mouse X"), 0f) * Time.deltaTime * rotSpeed;  
55         rot.eulerAngles = currRot;  
56         transform.rotation = rot;  
57     }  
58     previousMousePos = Input.mousePosition;  
59  
60     if (target != null)  
61         transform.position = target.position - offset;  
62     //Check for input  
63  
64 }  
65  
66  
67  
68  
69 Transform ChooseBestTarget()  
70 {  
71     throw new NotImplementedException();  
72 }  
73  
74 }
```

---

## ModelManager.cs

```
1 using System;
2 using System.Collections.Generic;
3 using System.Data;
4 using System.Linq;
5 using UnityEngine;
6 using NeuralNetwork;
7
8 public class ModelManager
9 {
10     public List<NetworkModel> Models = new List<NetworkModel>();
11     public int NumModels { get; }
12     public double LearningRate { get; }
13
14
15     public ModelManager(List<NetworkModel> models, double learningRate =
16         0.1f)
17     {
18         LearningRate = learningRate;
19         Models = models;
20         NumModels = models.Count;
21     }
22
23     public void SaveTop(int n)
24     {
25         if (n > Models.Count)
26         {
27             Debug.Log("Provided number is lower than count of models!");
28         }
29         else
30         {
31             Models.RemoveRange(0, n);
32             JsonService.SaveModelsList(Models);
33         }
34     }
35
36     public void Expand()
37     {
38         int n = Models.Count;
39
40         int clonesNeeded = NumModels - n;
41
42         List<NetworkModel> clones = new List<NetworkModel>();
43
44         if (n >= NumModels)
45         {
46             Debug.Log("Models collection is full!");
47         }
48         else
49         {
50             for (int i = 0; i < clonesNeeded; i++)
51             {
52                 clones.Add(Models[i % n].DeepCopy()); // add
```

```

        randomization
53         clones[clones.Count() - 1].Randomize(LearningRate);
54     }
55 }
56
57 Models = new List<NetworkModel>(Models.Concat(clones));
58 }
59
60
61
62 }
```

---



## Agent.cs

```
1 using System;
2 using NeuralNetwork;
3 using UnityEngine;
4 using System.Collections.Generic;
5 using System.Collections;
6
7 public class Agent : MonoBehaviour
8 {
9     public NetworkModel network;
10
11     //Target cookie jar for every agent that exists (?TODO?: handle null
12     //case?)
13     public static Transform cookieJar;
14
15     //function to call when this agent dies
16     public Action<Agent> deathCallback;
17     public List<double> lastInputs = new List<double>(); //inputs that
18     //were fed in previous frame (for UI and debugging)
19     public List<double> lastOutputs = new List<double>(); //outputs that
20     //were outputted in previous frame (for UI and debugging)
21     private float ForceMultiplier = 10.0f;
22
23     //View arc in radians
24     [SerializeField]
25     private float viewArc = 2.0f;
26
27     public float ViewArc
28     {
29         get { return viewArc; }
30         set
31         {
32             viewArc = value;
33             arcStep = viewArc / (float)rayCount; // recalculate arcStep
34         }
35     }
36
37     [SerializeField]
38     //Number of rays that will be cast
39     private int rayCount = 8;
40
41     private float arcStep = 0f;
42
43     void Awake()
44     {
45         arcStep = viewArc / (float)rayCount;
46
47         cookieJar = GameObject.Find("cookieJar").transform;
48         network = new NetworkModel();
49         network.Layers.Add(new NeuralLayer(1 + rayCount, 0.0,
50             ActivationFunc.Linear, "INPUT")); //rayCount + one for
```

```

50         CookieJar position
51         network.Layers.Add(new NeuralLayer(11, 0.0, ActivationFunc.
52             Linear, "HIDDEN"));
53         network.Layers.Add(new NeuralLayer(2, 0.0, ActivationFunc.Tanh,
54             "OUTPUT"));
55         network.Build();
56         network.Randomize(0.5);
57     }
58
59     void Start()
60     {
61
62     }
63
64     void FixedUpdate()
65     {
66         lastOutputs = network.Decide(GatherInputs());
67         ParseOutput(lastOutputs);
68     }
69
70     ///

```

```

96         {
97             results.Add(1.0f); // if nothing was hit, add max
98         }
99     }
100
101     //2. get distance from the cookie jar
102     results.Add(Vector3.Distance(this.transform.position, cookieJar.
103         position) / 100.0f);
104     lastInputs = results;
105     return results;
106 }
107
108 public void OnCollisionEnter(Collision c)
109 {
110     //if layer is wall layer
111     if (c.collider.gameObject.layer == 10)
112     {
113         //die...
114         this?.deathCallback(this);
115         Destroy(this.gameObject);
116     }
117 }
118
119 //EDITOR
120 void OnValidate()
121 {
122     //Because Unity does not support property exposing to the
123     //Inspector, we use OnValidate (called whenever, whatever
124     //changed by the Inspeotr)
125     //And force property to fire.
126     ViewArc = viewArc;
127 }

```

---

## InputMonitor.cs

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.UI;
5
6
7 public class InputMonitor : MonoBehaviour
8 {
9
10
11     public Agent Target;
12     public Text text;
13     // Start is called before the first frame update
14     void Start()
15     {
16         if (Target == null)
17             Debug.LogWarning("Target is null, no data will be shown.");
18     }
19
20     // Update is called once per frame
21     void FixedUpdate()
22     {
23         if (Target != null)
24         {
25             string input_data = "";
26             string output_data = "";
27             for (int i = 0; i < Target.lastInputs.Count; i++)
28             {
29                 input_data += "Input " + i + " --- " + Target.lastInputs
30                     [i] + "\n";
31
32                 for (int i = 0; i < Target.lastOutputs.Count; i++)
33                 {
34                     output_data += "Output " + i + " --- " + Target.
35                         lastOutputs[i] + "\n";
36                 }
37
38                 text.text = input_data + output_data;
39             }
40         }
41     }
```

---

## ScoreCalculator.cs

```
1 using UnityEngine;
2 using System.Collections.Generic;
3
4 public static class ScoreCalculator
5 {
6
7
8     public static float CalculateScore(GameObject obj)
9     {
10         //Get all components with given interfaces and map them to
11         //theirs type names
12         IScoringRule[] rules = obj.GetComponents<IScoringRule>();
13         Dictionary<string, IScoringRule> ruleDictionary = new Dictionary
14             <string, IScoringRule>();
15
16         foreach(IScoringRule r in rules)
17         {
18             ruleDictionary.Add(r.GetType().ToString(), r);
19         }
20
21         return ruleDictionary["MovementScoreRule"].GetScore() + 0.1f*
22             ruleDictionary["LifetimeScoreRule"].GetScore();
23     }
24
25
26 }
```

---

## LifetimeScoreRule.cs

```
1 using UnityEngine;
2
3 //Scoring rule: Add point depending on lifetime
4 public class LifetimeScoreRule : MonoBehaviour, IScoringRule
5 {
6     float score = 0;
7
8     void FixedUpdate()
9     {
10         score++;
11     }
12
13     public float GetScore()
14     {
15         return score;
16     }
17 }
```

---

## Pulse.cs

```
1 using System;
2 using System.Collections.Generic;
3 using System.Data;
4 using System.Linq;
5 using System.Runtime.Serialization.Formatters.Binary;
6 using System.Runtime.Serialization;
7 using System.IO;
8 using UnityEngine;
9 using Newtonsoft.Json;
10
11 namespace NeuralNetwork
12 {
13     [Serializable]
14     public class Pulse
15     {
16         [JsonProperty]
17         public double Value { get; set; }
18     }
19 }
```

---

## NeuralLayer.cs

```
1 using System;
2 using System.Collections.Generic;
3 using System.Data;
4 using System.Linq;
5 using System.Runtime.Serialization.Formatters.Binary;
6 using System.Runtime.Serialization;
7 using System.IO;
8 using UnityEngine;
9 using Newtonsoft.Json;
10
11 namespace NeuralNetwork
12 {
13     [Serializable]
14     public class NeuralLayer{
15         public List<Neuron> Neurons { get; set; }
16
17         public string Name { get; set; }
18
19         public double Weight { get; set; }
20
21         Func<double, double> Activation;
22         public NeuralLayer(int count, double initialWeight, Func<double,
23             double> activation, string name = "")
24         {
25             Activation = activation;
26             Neurons = new List<Neuron>();
27             for (int i = 0; i < count; i++)
28             {
29                 Neurons.Add(new Neuron(Activation));
30             }
31
32             Name = name;
33         }
34
35         public void Randomize(double lr)
36         {
37             foreach (var neuron in Neurons)
38             {
39                 neuron.Randomize(lr);
40             }
41         }
42
43         public void Forward()
44         {
45             foreach (var neuron in Neurons)
46             {
47                 neuron.Fire();
48             }
49         }
50
51         public override string ToString()
```



```
53     {
54         string tmp = "{\n";
55         for (int i = 0; i < Neurons.Count; i++)
56         {
57             tmp += Neurons[i].ToString();
58             if (i != Neurons.Count - 1)
59                 tmp += "\n";
60         }
61         tmp+="\n}";
62         return tmp;
63     }
64 }
65 }
66 }
```

---

## Dendrite.cs

```
1 using System;
2 using System.Collections.Generic;
3 using System.Data;
4 using System.Linq;
5 using System.Runtime.Serialization.Formatters.Binary;
6 using System.Runtime.Serialization;
7 using System.IO;
8 using UnityEngine;
9 using Newtonsoft.Json;
10
11 namespace NeuralNetwork
12 {
13
14     [Serializable]
15     public class Dendrite
16     {
17         public Pulse InputPulse { get; set; }
18
19         public double SynapticWeight { get; set; }
20
21         public bool Learnable { get; set; }
22
23         public Dendrite()
24         {
25             SynapticWeight = 0;
26         }
27
28         public void Randomize(double lr)
29         {
30             float t = (float)lr;
31             SynapticWeight += (double)UnityEngine.Random.Range(-t, t);
32         }
33
34
35         public override string ToString()
36         {
37             return SynapticWeight.ToString();
38         }
39
40     }
41 }
42 }
```

---

## Neuron.cs

```
1 using System;
2 using System.Collections.Generic;
3 using System.Data;
4 using System.Linq;
5 using System.Runtime.Serialization.Formatters.Binary;
6 using System.Runtime.Serialization;
7 using System.IO;
8 using UnityEngine;
9 using Newtonsoft.Json;
10
11 namespace NeuralNetwork
12 {
13     [Serializable]
14     public class Neuron
15     {
16         public List<Dendrite> Dendrites { get; set; }
17
18         public Pulse OutputPulse { get; set; }
19
20         public Func<double, double> Activation;
21
22         public Neuron(Func<double, double> activation)
23         {
24             Dendrites = new List<Dendrite>();
25             OutputPulse = new Pulse();
26             Activation = activation;
27         }
28
29         public void Randomize(double lr)
30         {
31             foreach (var dendrite in Dendrites)
32             {
33                 dendrite.Randomize(lr);
34             }
35         }
36
37         public void Fire()
38         {
39             OutputPulse.Value = Sum();
40
41             OutputPulse.Value = Activation(OutputPulse.Value);
42         }
43
44         private double Sum()
45         {
46             double computeValue = 0.0f;
47             foreach (var d in Dendrites)
48             {
49                 computeValue += d.InputPulse.Value * d.SynapticWeight;
50             }
51
52             return computeValue;
53         }
54     }
55 }
```

```
54
55     public override string ToString()
56     {
57         string tmp = "[";
58         for (int i = 0; i < Dendrites.Count; i++)
59         {
60             tmp += Dendrites[i].ToString();
61             if (i != Dendrites.Count - 1)
62                 tmp += ", ";
63         }
64         tmp += "]";
65         return tmp;
66     }
67
68
69 }
70 }
```

---

## Activation.cs

```
1 using UnityEngine;
2 using System;
3
4 namespace NeuralNetwork
5 {
6     public static class ActivationFunc
7     {
8         public static double Tanh(double x)
9         {
10             return 1 - (2.0) / (Math.Exp(2 * x) + 1);
11         }
12
13         public static double Linear(double x)
14         {
15             return x;
16         }
17
18         public static double BinaryStep(double x)
19         {
20             if (x > 0)
21             {
22                 return 1;
23             }
24             if (x == 0)
25             {
26                 return 0;
27             }
28             if (x < 0)
29             {
30                 return -1;
31             }
32             return 0;
33         }
34     }
35 }
36
37 }
```

---

## NetworkModel.cs

```
1 using System;
2 using System.Collections.Generic;
3 using System.Data;
4 using System.Linq;
5 using System.Runtime.Serialization.Formatters.Binary;
6 using System.Runtime.Serialization;
7 using System.IO;
8 using UnityEngine;
9 using Newtonsoft.Json;
10
11 namespace NeuralNetwork
12 {
13
14     [Serializable]
15     public class NetworkModel
16     {
17         public List<NeuralLayer> Layers { get; set; }
18
19         public NetworkModel()
20         {
21             Layers = new List<NeuralLayer>();
22         }
23
24         public NetworkModel DeepCopy()
25         {
26             using (MemoryStream ms = new MemoryStream())
27             {
28                 BinaryFormatter formatter = new BinaryFormatter();
29                 formatter.Context = new StreamingContext(
30                     StreamingContextStates.Clone);
31                 formatter.Serialize(ms, this);
32                 ms.Position = 0;
33                 return (NetworkModel)formatter.Deserialize(ms);
34             }
35
36             public void AddLayer(NeuralLayer layer)
37             {
38                 int dendriteCount = 1;
39                 if (Layers.Count > 0)
40                 {
41                     dendriteCount = Layers[Layers.Count - 1].Neurons.Count;
42                 }
43
44                 foreach (var element in layer.Neurons)
45                 {
46                     for (int i = 0; i < dendriteCount; i++)
47                     {
48                         element.Dendrites.Add(new Dendrite());
49                     }
50                 }
51             }
52
53         }
```

```

53     public void Build()
54     {
55         int i = 0;
56         foreach (var layer in Layers)
57         {
58             if (i >= Layers.Count - 1)
59             {
60                 break;
61             }
62             var nextLayer = Layers[i + 1];
63             CreateNetwork(layer, nextLayer);
64             i++;
65         }
66     }
67
68
69     public void Randomize(double lr)
70     {
71         foreach (var layer in Layers)
72         {
73             layer.Randomize(lr);
74         }
75     }
76
77     public List<double> Decide(List<double> X)
78     {
79         var inputLayer = Layers[0];
80         List<double> outputs = new List<double>();
81
82         for (int i = 0; i < X.Count; i++)
83         {
84             inputLayer.Neurons[i].OutputPulse.Value = X[i];
85         }
86         ComputeOutput();
87         foreach (var neuron in Layers.Last().Neurons)
88         {
89             outputs.Add(neuron.OutputPulse.Value);
90         }
91         return outputs;
92     }
93
94     public void Print()
95     {
96
97         Debug.Log("Name | Neurons");
98
99         foreach (var layer in Layers)
100         {
101             Debug.Log(layer.Name + " | " + layer.Neurons.Count);
102         }
103     }
104
105     private void CreateNetwork(NeuralLayer connectingFrom,
106                             NeuralLayer connectingTo)
107     {

```

```

107         foreach (var to in connectingTo.Neurons)
108         {
109             foreach (var from in connectingFrom.Neurons)
110             {
111                 to.Dendrites.Add(new Dendrite() { InputPulse = from.
                    OutputPulse });
112             }
113         }
114     }
115
116     private void ComputeOutput()
117     {
118         bool first = true;
119         foreach (var layer in Layers)
120         {
121             if (first)
122             {
123                 first = false;
124             }
125             else
126             {
127                 layer.Forward();
128             }
129         }
130     }
131
132     public override string ToString()
133     {
134         string tmp = "";
135         for (int i = 1; i < Layers.Count; i++)
136         {
137             tmp += Layers[i].ToString();
138             if (i != Layers.Count - 1)
139                 tmp += "\n";
140         }
141         tmp += "\n";
142         return tmp;
143     }
144
145 }
146
147 }

```

---



## JsonService.cs

```
1 using System;
2 using System.IO;
3 using System.Text;
4 using System.Collections.Generic;
5 using System.Data;
6 using System.Linq;
7 using UnityEngine;
8 using NeuralNetwork;
9 using Newtonsoft.Json;
10 public static class JsonService
11 {
12     public static void SaveModelsList(List<NetworkModel> models)
13     {
14         string timeString = DateTime.Now.ToString("yy-MM-dd_HH-mm-ss");
15         timeString = timeString.Replace(' ', '_');
16
17         JsonSerializerSettings settings = new JsonSerializerSettings();
18         settings.NullValueHandling = NullValueHandling.Include;
19         settings.ReferenceLoopHandling = ReferenceLoopHandling.Serialize;
20
21         string jsonString = JsonConvert.SerializeObject(models);
22         string path = Application.dataPath + "/jsonModels/" + timeString
23             + ".json";
24         File.WriteAllText(path, jsonString);
25     }
26     public static List<NetworkModel> LoadModelsList(TextAsset jsonFile)
27     {
28         List<NetworkModel> models = JsonConvert.DeserializeObject<List<
29             NetworkModel>>(jsonFile.ToString());
30         return models;
31     }
32 }
```

---