ДОДАТОК А

Вихідний код програми для навчання мережі

NeuralNetwork.cs:

```
using System.Collections.Generic;
using System;
using System.IO;

public class NeuralNetwork : IComparable<NeuralNetwork>
{
    private int[] layers;
    private float[][] neurons;
    private float[][] biases;
    private float[][][] weights;
    private int[] activations;

    public float fitness = 0;

    public NeuralNetwork(int[] layers)
    {
        this.layers = new int[layers.Length];
        for (int i = 0; i < layers.Length; i++)
        {
            this.layers[i] = layers[i];
        }
        InitNeurons();
        InitBiases();
        InitWeights();
    }

    private void InitNeurons()
    {
        List<float[]> neuronsList = new List<float[]>();
        for (int i = 0; i < layers.Length; i++)
        {
            neuronsList.Add(new float[layers[i]]);
        }
        neurons = neuronsList.ToArray();
    }

    private void InitBiases()
    {
        List<float[]> biasList = new List<float[]>();
        for (int i = 0; i < layers.Length; i++)
        {
            float[] bias = new float[layers[i]];
            for (int j = 0; j < layers[i]; j++)
            {
```

```
                bias[j] = UnityEngine.Random.Range(-0.5f, 0.5f);
            }
            biasList.Add(bias);
        }
        biases = biasList.ToArray();
    }

    private void InitWeights()
    {
        List<float[][]> weightsList = new List<float[][]>();
        for (int i = 1; i < layers.Length; i++)
        {
            List<float[]> layerWeightsList = new List<float[]>();
            int neuronsInPreviousLayer = layers[i - 1];
            for (int j = 0; j < neurons[i].Length; j++)
            {
                float[] neuronWeights = new
float[neuronsInPreviousLayer];
                for (int k = 0; k < neuronsInPreviousLayer; k++)
                {
                    neuronWeights[k] = UnityEngine.Random.Range(-
0.5f, 0.5f);
                }
                layerWeightsList.Add(neuronWeights);
            }
            weightsList.Add(layerWeightsList.ToArray());
        }
        weights = weightsList.ToArray();
    }

    public float[] FeedForward(float[] inputs)
    {
        for (int i = 0; i < inputs.Length; i++)
        {
            neurons[0][i] = inputs[i];
        }
        for (int i = 1; i < layers.Length; i++)
        {
            int layer = i - 1;
            for (int j = 0; j < neurons[i].Length; j++)
            {
                float value = 0f;
                for (int k = 0; k < neurons[i - 1].Length; k++)
                {
                    value += weights[i - 1][j][k] * neurons[i -
1][k];
                }
                neurons[i][j] = activate(value + biases[i][j]);
            }
        }
        return neurons[neurons.Length - 1];
```

```
    }

    public float activate(float value)
    {
        return (float)Math.Tanh(value);
    }

    public void Mutate(int chance, float val)
    {
        for (int i = 0; i < biases.Length; i++)
        {
            for (int j = 0; j < biases[i].Length; j++)
            {
                biases[i][j] = (UnityEngine.Random.Range(0f, chance)
<= 5) ? biases[i][j] += UnityEngine.Random.Range(-val, val) :
biases[i][j];
            }
        }

        for (int i = 0; i < weights.Length; i++)
        {
            for (int j = 0; j < weights[i].Length; j++)
            {
                for (int k = 0; k < weights[i][j].Length; k++)
                {
                    weights[i][j][k] = (UnityEngine.Random.Range(0f,
chance) <= 5) ? weights[i][j][k] += UnityEngine.Random.Range(-val,
val) : weights[i][j][k];
                }
            }
        }
    }

    public int CompareTo(NeuralNetwork other)
    {
        if (other == null) return 1;

        if (fitness > other.fitness)
            return 1;
        else if (fitness < other.fitness)
            return -1;
        else
            return 0;
    }

    public NeuralNetwork copy(NeuralNetwork nn)
    {
        for (int i = 0; i < biases.Length; i++)
        {
            for (int j = 0; j < biases[i].Length; j++)
            {
```

```
                nn.biases[i][j] = biases[i][j];
            }
        }
        for (int i = 0; i < weights.Length; i++)
        {
            for (int j = 0; j < weights[i].Length; j++)
            {
                for (int k = 0; k < weights[i][j].Length; k++)
                {
                    nn.weights[i][j][k] = weights[i][j][k];
                }
            }
        }
        return nn;
    }

    public void Load(string path)
    {
        TextReader tr = new StreamReader(path);
        int NumberOfLines = (int)new FileInfo(path).Length;
        string[] ListLines = new string[NumberOfLines];
        int index = 1;
        for (int i = 1; i < NumberOfLines; i++)
        {
            ListLines[i] = tr.ReadLine();
        }
        tr.Close();
        if (new FileInfo(path).Length > 0)
        {
            for (int i = 0; i < biases.Length; i++)
            {
                for (int j = 0; j < biases[i].Length; j++)
                {
                    biases[i][j] = float.Parse(ListLines[index]);
                    index++;
                }
            }

            for (int i = 0; i < weights.Length; i++)
            {
                for (int j = 0; j < weights[i].Length; j++)
                {
                    for (int k = 0; k < weights[i][j].Length; k++)
                    {
                        weights[i][j][k] =
float.Parse(ListLines[index]); ;
                        index++;
                    }
                }
            }
        }
```

```
    }

    public void Save(string path)
    {
        File.Create(path).Close();
        StreamWriter writer = new StreamWriter(path, true);

        for (int i = 0; i < biases.Length; i++)
        {
            for (int j = 0; j < biases[i].Length; j++)
            {
                writer.WriteLine(biases[i][j]);
            }
        }

        for (int i = 0; i < weights.Length; i++)
        {
            for (int j = 0; j < weights[i].Length; j++)
            {
                for (int k = 0; k < weights[i][j].Length; k++)
                {
                    writer.WriteLine(weights[i][j][k]);
                }
            }
        }
        writer.Close();
    }
}
```

Manager.cs:

```
using System.Collections.Generic;
using UnityEngine;

public class Manager : MonoBehaviour
{

    public float timeframe;
    public int populationSize;
    public GameObject prefab;

    public int[] layers = new int[3] { 5, 3, 2 };

    [Range(0.0001f, 1f)] public float MutationChance = 0.01f;

    [Range(0f, 1f)] public float MutationStrength = 0.5f;

    [Range(0.1f, 10f)] public float Gamespeed = 1f;

    public List<NeuralNetwork> networks;
```

```
    private List<Bot> cars;
    private int counter = 0;

    void Start()
    {
        if (populationSize % 2 != 0)
            populationSize = 50;

        InitNetworks();
        CreateBots();
        InvokeRepeating("CreateBots", 0.1f, timeframe);
    }

    public void InitNetworks()
    {
        networks = new List<NeuralNetwork>();
        for (int i = 0; i < populationSize; i++)
        {
            NeuralNetwork net = new NeuralNetwork(layers);
            net.Load("Assets/Save.txt");
            networks.Add(net);
        }
    }

    public void CreateBots()
    {
        Time.timeScale = Gamespeed;
        if (cars != null)
        {
            for (int i = 0; i < cars.Count; i++)
            {
                GameObject.Destroy(cars[i].gameObject);
            }

            SortNetworks();
        }

        cars = new List<Bot>();
        for (int i = 0; i < populationSize; i++)
        {
            Bot car = (Instantiate(prefab, new Vector3(125, 3.9f,
42.15f), new Quaternion(0, 0, 1, 0))).GetComponent<Bot>();
            car.network = networks[i];
            cars.Add(car);
        }
        counter++;
        Debug.Log("Count of iterations: " + counter);
    }

    public void SortNetworks()
    {
```

```
        for (int i = 0; i < populationSize; i++)
        {
            cars[i].UpdateFitness();
        }
        networks.Sort();
        networks[populationSize - 1].Save("Assets/Save.txt");
        for (int i = 0; i < populationSize / 2; i++)
        {
            networks[i] = networks[i + populationSize / 2].copy(new
NeuralNetwork(layers));
            networks[i].Mutate((int)(1/MutationChance),
MutationStrength);
        }
    }
}
```

Bot.cs:

```
using System.Collections.Generic;
using UnityEngine;


public class Bot : MonoBehaviour
{
    public float speed;
    public float rotation;
    public LayerMask raycastMask;
    private HashSet<GameObject> _listGameObject = new
HashSet<GameObject>();

    private RaycastHit hit;
    private Ray Ray;
    private Vector3 newVector;

    private float[] input = new float[5];
    public NeuralNetwork network;

    public int position;
    public bool collided;

    void FixedUpdate()
    {
        if (!collided)
        {
            for (int i = 0; i < 5; i++)
            {
                newVector = Quaternion.AngleAxis(i * 45 - 90, new
Vector3(0, 1, 0)) * transform.right;
                Ray = new Ray(transform.position, newVector);

                if (Physics.Raycast(Ray, out hit, 10, raycastMask))
```

```
                {
                    input[i] = (10 - hit.distance) / 10;
                }
                else
                {
                    input[i] = 0;
                }
            }

            float[] output = network.FeedForward(input);

            transform.Rotate(0, output[0] * rotation, 0,
Space.World);
            transform.position += this.transform.right * output[1] *
speed;
        }
    }


    void OnCollisionEnter(Collision collision)
    {
        if(collision.collider.gameObject.layer ==
LayerMask.NameToLayer("CheckPoint"))
        {
            GameObject[] checkPoints =
GameObject.FindGameObjectsWithTag("CheckPoint");
            for (int i=0; i < checkPoints.Length; i++)
            {
                if (collision.collider.gameObject == checkPoints[i])
                {
                    position++;
                    _listGameObject.Add(checkPoints[i]);
                    break;
                }
            }
        }
        else if(collision.collider.gameObject.layer !=
LayerMask.NameToLayer("Learner"))
        {
            collided = true;
        }
    }

    public void UpdateFitness()
    {
        network.fitness = _listGameObject.Count;
    }
}
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	78 с.
	Інші документи	
2.	Презентаційні матеріали	15 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Скворцов Д.О.			Використання еволюційного навчання глибокої нейронної мережі в завданні обходу перешкод
Перевірив		Дейнеко А.Д.			
Н.контр.		Малєєва І.А.			
Затв.		Філатов В.О.			

Шифр групи	Код напр./спец.
СШІзм-20-1	122
ХНУРЕ кафедра ШІ	