ДОДАТОК А

ФРАГМЕНТИ ВИХІДНОГО КОДУ СИСТЕМИ

```
public class GodManager : MonoBehaviour
{
    public GameObject CellPrefab;
    public GameObject Grid;

    private MapTile[,] Map;

    private List<Unit> Units;

    private Team _myTeam;
    private Team _enemyTeam;

    private int _index;

    // Start is called before the first frame update
    void Start()
    {
        Map = new MapTile[10, 6];
        for (int j = 0; j < 6; j++)
        {
            for (int i = 0; i < 10; i++)
            {
                var go = Instantiate(CellPrefab, Grid.transform);
                var cell = go.GetComponent<MapTile>();
                Map[i, j] = cell;
            }
        }

        SetUnits();
        StartCoroutine(nameof(Mooving));
        _index = 0;
    }


    private IEnumerator Mooving()
    {
        while (true)
        {
            yield return new WaitForSeconds(1);
            _index %= Units.Count;
            var unit = Units[_index];
            UnitMove(unit, unit.Network.Execute(MapInputs()));

            _index++;
        }
    }

    private void UnitMove(Unit unit, List<double> execute)
    {
        var keys = execute.Select((item, index) => new
            {
                Key = index,
                Value = item
            })
            .OrderByDescending(item => unit.IsAttack(item.Key))
```

```
            .ThenBy(item => item.Value)
            .Select(item=>item.Key);

        foreach (var key in keys)
        {
            var attack = unit.IsAttack(key);
            var position = unit.GetPosition(key);

            var x = unit.X + (int) position.x;
            var y = unit.Y + (int) position.y;

            if (x < 0 || x >= Map.GetLength(0))
                continue;

            if (y < 0 || y >= Map.GetLength(1))
                continue;

            if (!attack)
            {
                if (Map[x, y].EnemyType != EnemyType.None)
                    continue;

                unit.X = x;
                unit.Y = y;
                break;
            }

            if (Map[x, y].EnemyType == EnemyType.None)
                continue;

            var enemyUnit = Units.FirstOrDefault(item => item.X == x && item.Y == y);

            if (enemyUnit == null || enemyUnit.Team == unit.Team)
                continue;

            enemyUnit.Health -= unit.Attack(enemyUnit.Type);

            if (unit.Type != UnitType.Archer)
                unit.Health -= enemyUnit.Attack(unit.Type)/2;

            break;
        }
    }

    private List<double> MapInputs()
    {
        var result = new List<double>();
        for (var i = 0; i < Map.GetLength(0); i++)
        {
            for (var j = 0; j < Map.GetLength(1); j++)
            {
                var mapTile = Map[i, j];
                result.Add(mapTile.MapType == MapType.None ? 1 : 0);
                result.Add(mapTile.MapType == MapType.Archer ? 1 : 0);
                result.Add(mapTile.MapType == MapType.Swordsman ? 1 : 0);
                result.Add(mapTile.MapType == MapType.Spearman ? 1 : 0);
                result.Add(mapTile.MapType == MapType.Rider ? 1 : 0);
                result.Add(mapTile.EnemyType == EnemyType.None ? 1 : 0);
                result.Add(mapTile.EnemyType == EnemyType.Mine ? 1 : 0);
                result.Add(mapTile.EnemyType == EnemyType.Enemy ? 1 : 0);
            }
```

```
        }

        return result;
    }

    public void SetUnits()
    {
        Units = new List<Unit>();
        _myTeam = new Team();
        _enemyTeam = new Team();

        Units.Add(new Unit(0, 0, UnitType.Archer, _myTeam));
        Units.Add(new Unit(0, 2, UnitType.Swordsman, _myTeam));
        Units.Add(new Unit(0, 3, UnitType.Spearman, _myTeam));
        Units.Add(new Unit(0, 5, UnitType.Rider, _myTeam));

        Units.Add(new Unit(9, 0, UnitType.Archer, _enemyTeam));
        Units.Add(new Unit(9, 2, UnitType.Swordsman, _enemyTeam));
        Units.Add(new Unit(9, 3, UnitType.Spearman, _enemyTeam));
        Units.Add(new Unit(9, 5, UnitType.Rider, _enemyTeam));
    }

    // Update is called once per frame
    void Update()
    {
        for (var j = 0; j < 6; j++)
        {
            for (var i = 0; i < 10; i++)
            {
                var mapTile = Map[i, j];
                mapTile.EnemyType = EnemyType.None;
                mapTile.MapType = MapType.None;
            }
        }

        var unitsForDelete = new List<Unit>();

        foreach (var unit in Units)
        {
            var mapTile = Map[unit.X, unit.Y];
            mapTile.EnemyType = unit.Team == _myTeam ? EnemyType.Mine :
EnemyType.Enemy;

            mapTile.UnitHealth = unit.Health;

            if (unit.Health <= 0)
                unitsForDelete.Add(unit);

            switch (unit.Type)
            {
                case UnitType.Archer:
                    mapTile.MapType = MapType.Archer;
                    break;
                case UnitType.Swordsman:
                    mapTile.MapType = MapType.Swordsman;
                    break;
                case UnitType.Spearman:
                    mapTile.MapType = MapType.Spearman;
                    break;
                case UnitType.Rider:
                    mapTile.MapType = MapType.Rider;
```

```
                    break;
                }
            }

            foreach (var unit in unitsForDelete)
            {
                Units.Remove(unit);
            }
        }
    }

    public enum MapType
    {
        None,
        Archer,
        Swordsman,
        Spearman,
        Rider,
    }

    public enum EnemyType
    {
        None,
        Mine,
        Enemy
    }

    public class MapTile : MonoBehaviour
    {
        public EnemyType EnemyType;

        public MapType MapType;

        public Image Image;

        public Sprite Archer;
        public Sprite Swordsman;
        public Sprite Spearman;
        public Sprite Rider;

        public Image[] Health;

        public int UnitHealth;

        // Start is called before the first frame update
        void Start()
        {

        }

        // Update is called once per frame
        void Update()
        {
            foreach (var image in Health)
                image.gameObject.SetActive(true);

            switch (MapType)
            {
                case MapType.None:
                    Image.sprite = null;
                    foreach (var image in Health)
```

```
                    image.gameObject.SetActive(false);
                break;
            case MapType.Archer:
                Image.sprite = Archer;
                break;
            case MapType.Swordsman:
                Image.sprite = Swordsman;
                break;
            case MapType.Spearman:
                Image.sprite = Spearman;
                break;
            case MapType.Rider:
                Image.sprite = Rider;
                break;
        }

        switch (EnemyType)
        {
            case EnemyType.None:
                Image.color = new Color(0.074f, 0.547f ,0.023f);
                break;
            case EnemyType.Mine:
                Image.color = Color.green;
                break;
            case EnemyType.Enemy:
                Image.color = Color.red;
                break;
            default:
                throw new ArgumentOutOfRangeException();
        }

        for (var index = 0; index < Health.Length; index++)
            Health[index].color = UnitHealth > index ? new Color(0.5f, 0.1f, 0.1f) :
new Color(0, 0, 0, 0);
    }
}

public enum UnitType
{
    Archer,
    Swordsman,
    Spearman,
    Rider,
}

public class Unit
{
    public UnitType Type;

    public int X;

    public int Y;

    public Team Team;

    public Network Network;

    public int Health;

    public Unit(int x, int y, UnitType type, Team team)
    {
```

```
        X = x;
        Y = y;
        Type = type;
        Team = team;
        Network = SetNetworkByType(type);
        Health = 10;
    }

    private Network SetNetworkByType(UnitType type)
    {
        Network result;
        switch (type)
        {
            case UnitType.Archer:
                result = new Network(new[] {480, 120, 24},
ActivationFunctions.Sigmoid);
                break;
            case UnitType.Swordsman:
                result = new Network(new[] {480, 96, 16},
ActivationFunctions.Sigmoid);
                break;
            case UnitType.Spearman:
                result = new Network(new[] {480, 96, 16},
ActivationFunctions.Sigmoid);
                break;
            case UnitType.Rider:
                result = new Network(new[] {480, 132, 44},
ActivationFunctions.Sigmoid);
                break;
            default:
                throw new ArgumentOutOfRangeException(nameof(type), type, null);
        }

        return result;
    }

    public int Attack(UnitType type)
    {
        switch (Type)
        {
            case UnitType.Archer:
                switch (type)
                {
                    case UnitType.Archer:
                        return 2;
                    case UnitType.Swordsman:
                        return 2;
                    case UnitType.Spearman:
                        return 2;
                    case UnitType.Rider:
                        return 2;
                }

                break;
            case UnitType.Swordsman:
                switch (type)
                {
                    case UnitType.Archer:
                        return 2;
                    case UnitType.Swordsman:
                        return 2;
```

```
                case UnitType.Spearman:
                    return 3;
                case UnitType.Rider:
                    return 1;
            }

            break;
        case UnitType.Spearman:
            switch (type)
            {
                case UnitType.Archer:
                    return 2;
                case UnitType.Swordsman:
                    return 1;
                case UnitType.Spearman:
                    return 2;
                case UnitType.Rider:
                    return 3;
            }

            break;
        case UnitType.Rider:
            switch (type)
            {
                case UnitType.Archer:
                    return 2;
                case UnitType.Swordsman:
                    return 3;
                case UnitType.Spearman:
                    return 1;
                case UnitType.Rider:
                    return 2;
            }

            break;
    }

    return 0;
}

public bool IsAttack(int key)
{
    switch (Type)
    {
        case UnitType.Archer:
            return key >= 12;
        case UnitType.Swordsman:
            return key >= 12;
        case UnitType.Spearman:
            return key >= 12;
        case UnitType.Rider:
            return key >= 40;
    }

    return false;
}

public Vector2 GetPosition(int key)
{
    switch (Type)
    {
```

```
            case UnitType.Archer:
                return GetArcherPosition(key);
            case UnitType.Swordsman:
                return GetSwordsmanPosition(key);
            case UnitType.Spearman:
                return GetSpearmanPosition(key);
            case UnitType.Rider:
                return GetRiderPosition(key);
            default:
                throw new ArgumentOutOfRangeException();
        }

        return new Vector2();
    }


    private Vector2 GetRiderPosition(int key)
    {
        var result = new Vector2();
        switch (key)
        {
            case 0:
                result = new Vector2(-4, 0);
                break;
            case 1:
                result = new Vector2(-3, -1);
                break;
            case 2:
                result = new Vector2(-3, 0);
                break;
            case 3:
                result = new Vector2(-3, 1);
                break;
            case 4:
                result = new Vector2(-2, -2);
                break;
            case 5:
                result = new Vector2(-2, -1);
                break;
            case 6:
                result = new Vector2(-2, 0);
                break;
            case 7:
                result = new Vector2(-2, 1);
                break;
            case 8:
                result = new Vector2(-2, 2);
                break;
            case 9:
                result = new Vector2(-1, -3);
                break;
            case 10:
                result = new Vector2(-1, -2);
                break;
            case 11:
                result = new Vector2(-1, -1);
                break;
            case 12:
                result = new Vector2(-1, 0);
                break;
            case 13:
```

```
            result = new Vector2(-1, 1);
            break;
        case 14:
            result = new Vector2(-1, 2);
            break;
        case 15:
            result = new Vector2(-1, 3);
            break;
        case 16:
            result = new Vector2(0, -4);
            break;
        case 17:
            result = new Vector2(0, -3);
            break;
        case 18:
            result = new Vector2(0, -2);
            break;
        case 19:
            result = new Vector2(0, -1);
            break;
        case 20:
            result = new Vector2(0, 1);
            break;
        case 21:
            result = new Vector2(0, 2);
            break;
        case 22:
            result = new Vector2(0, 3);
            break;
        case 23:
            result = new Vector2(0, 4);
            break;
        case 24:
            result = new Vector2(1, -3);
            break;
        case 25:
            result = new Vector2(1, -2);
            break;
        case 26:
            result = new Vector2(1, -1);
            break;
        case 27:
            result = new Vector2(1, 0);
            break;
        case 28:
            result = new Vector2(1, 1);
            break;
        case 29:
            result = new Vector2(1, 2);
            break;
        case 30:
            result = new Vector2(1, 3);
            break;
        case 31:
            result = new Vector2(2, -2);
            break;
        case 32:
            result = new Vector2(2, -1);
            break;
        case 33:
            result = new Vector2(2, 0);
```

```
            break;
        case 34:
            result = new Vector2(2, 1);
            break;
        case 35:
            result = new Vector2(2, 2);
            break;
        case 36:
            result = new Vector2(3, -1);
            break;
        case 37:
            result = new Vector2(3, 0);
            break;
        case 38:
            result = new Vector2(3, 1);
            break;
        case 39:
            result = new Vector2(4, 0);
            break;
        case 40:
            result = new Vector2(-1, 0);
            break;
        case 41:
            result = new Vector2(0, -1);
            break;
        case 42:
            result = new Vector2(0, 1);
            break;
        case 43:
            result = new Vector2(1, 0);
            break;
    }

    return result;
}

private Vector2 GetArcherPosition(int key)
{
    var result = new Vector2();
    switch (key)
    {
        case 0:
            result = new Vector2(-2, 0);
            break;
        case 1:
            result = new Vector2(-1, -1);
            break;
        case 2:
            result = new Vector2(-1, 0);
            break;
        case 3:
            result = new Vector2(-1, 1);
            break;
        case 4:
            result = new Vector2(0, -2);
            break;
        case 5:
            result = new Vector2(0, -1);
            break;
        case 6:
            result = new Vector2(0, 1);
```

```
            break;
        case 7:
            result = new Vector2(0, 2);
            break;
        case 8:
            result = new Vector2(1, -1);
            break;
        case 9:
            result = new Vector2(1, 0);
            break;
        case 10:
            result = new Vector2(1, 1);
            break;
        case 11:
            result = new Vector2(2, 0);
            break;
        case 12:
            result = new Vector2(-2, 0);
            break;
        case 13:
            result = new Vector2(-1, -1);
            break;
        case 14:
            result = new Vector2(-1, 0);
            break;
        case 15:
            result = new Vector2(-1, 1);
            break;
        case 16:
            result = new Vector2(0, -2);
            break;
        case 17:
            result = new Vector2(0, -1);
            break;
        case 18:
            result = new Vector2(0, 1);
            break;
        case 19:
            result = new Vector2(0, 2);
            break;
        case 20:
            result = new Vector2(1, -1);
            break;
        case 21:
            result = new Vector2(1, 0);
            break;
        case 22:
            result = new Vector2(1, 1);
            break;
        case 23:
            result = new Vector2(2, 0);
            break;
    }

    return result;
}

private Vector2 GetSwordsmanPosition(int key)
{
    var result = new Vector2();
    switch (key)
```

```
    {
        case 0:
            result = new Vector2(-2, 0);
            break;
        case 1:
            result = new Vector2(-1, -1);
            break;
        case 2:
            result = new Vector2(-1, 0);
            break;
        case 3:
            result = new Vector2(-1, 1);
            break;
        case 4:
            result = new Vector2(0, -2);
            break;
        case 5:
            result = new Vector2(0, -1);
            break;
        case 6:
            result = new Vector2(0, 1);
            break;
        case 7:
            result = new Vector2(0, 2);
            break;
        case 8:
            result = new Vector2(1, -1);
            break;
        case 9:
            result = new Vector2(1, 0);
            break;
        case 10:
            result = new Vector2(1, 1);
            break;
        case 11:
            result = new Vector2(2, 0);
            break;
        case 12:
            result = new Vector2(-1, 0);
            break;
        case 13:
            result = new Vector2(0, -1);
            break;
        case 14:
            result = new Vector2(0, 1);
            break;
        case 15:
            result = new Vector2(-1, 0);
            break;
    }

    return result;
}

private Vector2 GetSpearmanPosition(int key)
{
    var result = new Vector2();
    switch (key)
    {
        case 0:
            result = new Vector2(-2, 0);
```

```
                break;
            case 1:
                result = new Vector2(-1, -1);
                break;
            case 2:
                result = new Vector2(-1, 0);
                break;
            case 3:
                result = new Vector2(-1, 1);
                break;
            case 4:
                result = new Vector2(0, -2);
                break;
            case 5:
                result = new Vector2(0, -1);
                break;
            case 6:
                result = new Vector2(0, 1);
                break;
            case 7:
                result = new Vector2(0, 2);
                break;
            case 8:
                result = new Vector2(1, -1);
                break;
            case 9:
                result = new Vector2(1, 0);
                break;
            case 10:
                result = new Vector2(1, 1);
                break;
            case 11:
                result = new Vector2(2, 0);
                break;
            case 12:
                result = new Vector2(-1, 0);
                break;
            case 13:
                result = new Vector2(0, -1);
                break;
            case 14:
                result = new Vector2(0, 1);
                break;
            case 15:
                result = new Vector2(-1, 0);
                break;
        }

        return result;
    }
}

public class Neuron
{
    public List<double> Weights;
    public Func<double, double> Activation;

    public Neuron(Func<double, double> activation, int countWeights)
    {
        Activation = activation;
        Weights = new List<double>();
```

```
            for (var i = 0; i < countWeights; i++)
                Weights.Add(Random.Range(-100f, 100f));
        }

        public double Execute(List<double> inputs)
        {
            if (inputs.Count != Weights.Count)
                throw new ArgumentException();

            return Activation(inputs.Select((t, i) => Weights[i] * inputs.Count).Sum());
        }

        public void Mutate()
        {
            for (var index = 0; index < Weights.Count; index++)
            {
                Weights[index] = Weights[index] + Random.Range(-20f, 20f);
                if (Weights[index] < -100)
                    Weights[index] = -100;
                if (Weights[index] > 100)
                    Weights[index] = 100;
            }
        }
    }

    public class Layer
    {
        public List<Neuron> Neurons;

        public Layer(int currentLayer, int previousLayer, Func<double, double>
    activationFunction)
        {
            Neurons = new List<Neuron>();

            for (var i = 0; i < currentLayer; i++)
                Neurons.Add(new Neuron(activationFunction, previousLayer));
        }

        public List<double> Excute(List<double> input)
        {
            return Neurons.Select(item => item.Execute(input)).ToList();
        }

        public void Mutate()
        {
            foreach (var neuron in Neurons)
                neuron.Mutate();
        }
    }

    public class Network
    {
        public List<Layer> Layers;

        private int[] _settings;
        private Func<double, double> _activationFunction;

        public Network(int[] settings, Func<double, double> activationFunction)
        {
            _settings = settings;
```

```
        _activationFunction = activationFunction;

        Layers = new List<Layer>();

        for (var index = 1; index < settings.Length; index++)
            Layers.Add(new Layer(settings[index], settings[index-1],
activationFunction));
    }

    public List<double> Execute(List<double> input)
    {
        return Layers.Aggregate(input, (current, layer) => layer.Excute(current));
    }

    public void Mutate()
    {
        foreach (var layer in Layers)
            layer.Mutate();
    }

    public Network Cross(Network network)
    {
        var result = new Network(_settings, _activationFunction);
        for (var i = 0; i < result.Layers.Count; i++)
        {
            var layer = result.Layers[i];
            for (var j = 0; j < layer.Neurons.Count; j++)
            {
                var layerNeuron = layer.Neurons[j];
                for (var k = 0; k < layerNeuron.Weights.Count; k++)
                {
                    layerNeuron.Weights[k] = (Layers[i].Neurons[j].Weights[k] +
network.Layers[i].Neurons[j].Weights[k])/2;
                }
            }
        }

        return result;
    }
}

public static class ActivationFunctions
{
    public static double Line(double x)
    {
        return x;
    }

    public static double Sigmoid(double x)
    {
        return 1 / (1 + Math.Exp(-x));
    }

    public static double Th(double x)
    {
        return (Math.Exp(2 * x) - 1) / (Math.Exp(2 * x) + 1);
    }
}
```