Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Кубанский государственный технологический университет»

Кафедра информационных систем и программирования

**СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА**

Отчет по лабораторной работе №5-8

«Система обучающихся агентов»

Выполнил

студент 4 курса

группы 20-КБ-ПР1

Можаев М.Д.

Принял:

преподаватель кафедры

Тотухов К.Е.

Краснодар

2023

**Цель работы:**

В рамках четырёх работ (ЛР 5-8), реализовать обучение интеллектуальных нейросетевых агентов методом искусственной жизни. Цель ЛР №5 – собрать вместе все компоненты среды и запустить.

**Задание:**

Завершить программную реализацию искусственной жизни на основе ЛР 5-8.

Вариант 15

**Общее задание на ЛР 5-8**:

Если ваш вариант нечётный, то ваши агенты: хищники, травоядные, растения. Хищники поедают только травоядных, травоядные поедают только растения, растения генерируются случайно и существуют, пока их кто-то не съесть.

Если чётный, то вместо хищников у вас - фагоциты, вместо травоядных – бактерии, вместо растений – молекулы глюкозы. Бактерии могут есть только глюкозу. Фагоциты могут есть и бактерий, и глюкозу, но глюкоза даёт энергии в четыре раза меньше. Глюкоза генерируется случайно и существует, пока её не съедят.

Вариант определяется вашим номером в группе по списку.

**Ход выполнения:**

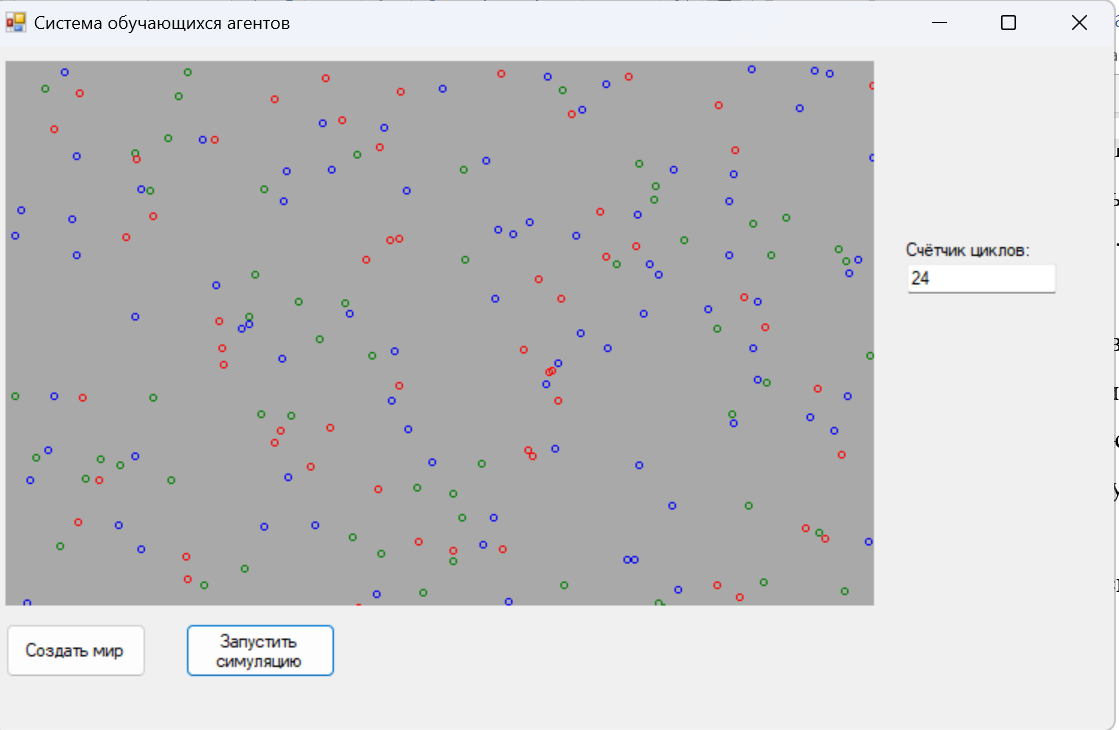
****

Рисунок 1 – Результат работы программы

**Листинг**

using lr5;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Serialization;

namespace lr5

{

public class Herbivore : Creature

{

public Herbivore(int X, int Y):base(X,Y) { }

public override void Eat(List<Creature> creatures)

{

if (eatPlantIndex > -1)

{

this.health += 4;

MakeNewCreature(creatures);

}

}

protected override void MakeNewCreature(List<Creature> creatures)

{

Random rnd = new Random();

int childX = Location.X + rnd.Next(-2, 2);

int chlidY = Location.Y + rnd.Next(-2, 2);

if (health >= 20)

{

Herbivore child = new Herbivore(childX, chlidY);

for (int i = 0; i < neurons.Count; i++)

{

double[] mutatedWeights = neurons[i].Weights;

for (int j = 0; j < mutatedWeights.Length; j++)

{

mutatedWeights[j] += (rnd.NextDouble() \* 2 - 1) / 2.0;

}

child.neurons[i].Weights = mutatedWeights;

}

creatures.Add(child);

health = 10;

}

}

public override Pen GetCreaturePen()

{

return new Pen(Color.Blue);

}

}

}

using lr5.Creatures;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms.VisualStyles;

namespace lr5

{

public class Creature

{

protected List<Neuron> neurons = new List<Neuron>();

protected Perception perception;

protected int eatHerbIndex;

protected int eatPlantIndex;

protected int health { get; set; }

protected int notEat = 0;

protected Direction direction;

public Point Location { get; set; }

public Creature(int X, int Y)

{

health = 10;

perception = new Perception();

direction = Direction.North;

Location = new Point(X, Y);

for (int i = 0; i < 4; i++)

{

neurons.Add(new Neuron());

}

perception.UpdatePerceptionFacingNorth(Location);

}

public void Damage(List<Creature> creatures,Creature creature)

{

creature.health--;

if (creature.health <= 0) creatures.Remove(creature);

}

public virtual Pen GetCreaturePen()

{

return new Pen(Color.Transparent);

}

private int[] GetNeuronOutput()

{

int[] output = new int[4];

int neuronIndex = -1;

double tmpMax = Double.MinValue;

for (int i = 0; i < 4; i++)

{

Neuron neuron = neurons[i];

double tmpOutput = neuron.CalculatePotential();

if (tmpOutput >= tmpMax)

{

tmpMax = tmpOutput;

neuronIndex = i;

}

}

output[neuronIndex] = 1;

return output;

}

public void ScanNearbyWorld(ref List<Creature> creatures)

{

int[] neuronInputArray = new int[12];

this.eatHerbIndex = -1;

eatPlantIndex = -1;

for (int i = 0; i < creatures.Count; i++)

{

var creature = creatures[i];

if ((Math.Pow(creature.Location.X - this.Location.X, 2) + Math.Pow(creature.Location.Y - this.Location.Y, 2)) < 36 && ((creature.GetType() == typeof(Herbivore) && this.GetType() == typeof(Predator)) || (creature.GetType() == typeof(Plant) && this.GetType() == typeof(Herbivore))))

{

creatures.Remove(creature);

this.health += 4;

}

foreach (Point frontPoint in perception.frontPerception)

{

if (creature.Location == frontPoint)

{

if (creature.GetType() == typeof(Herbivore))

{

neuronInputArray[0] = 1;

}

else if (creature.GetType() == typeof(Predator))

{

neuronInputArray[1] = 1;

}

else

{

neuronInputArray[2] = 1;

}

}

}

foreach (Point leftPoint in perception.leftPerception)

{

if (creature.Location == leftPoint)

{

if (creature.GetType() == typeof(Herbivore))

{

neuronInputArray[3] = 1;

}

else if (creature.GetType() == typeof(Predator))

{

neuronInputArray[4] = 1;

}

else

{

neuronInputArray[5] = 1;

}

}

}

foreach (Point rightPoint in perception.rightPerception)

{

if (creature.Location == rightPoint)

{

if (creature.GetType() == typeof(Herbivore))

{

neuronInputArray[6] = 1;

}

else if (creature.GetType() == typeof(Predator))

{

neuronInputArray[7] = 1;

}

else

{

neuronInputArray[8] = 1;

}

}

}

foreach (Point nearPoint in perception.nearPerception)

{

if (creature.Location == nearPoint)

{

if (creature.GetType() == typeof(Herbivore))

{

neuronInputArray[9] = 1;

creature.health = 0;

Damage(creatures, creature);

this.eatHerbIndex = i;

}

else if (creature.GetType() == typeof(Predator))

{

neuronInputArray[10] = 1;

}

else

{

neuronInputArray[11] = 1;

creature.health = 0;

Damage(creatures, creature);

this.eatPlantIndex = i;

}

}

}

}

foreach(Neuron n in neurons)

{

n.Input = neuronInputArray;

}

}

public void Act(ref List<Creature> creatures)

{

int[] neuronOutput = GetNeuronOutput();

for (int i = 0; i < neuronOutput.Length; i++)

{

if (notEat == 10) Damage(creatures,this);

else if (neuronOutput[i] != 0 && i == 0)

{

TurnLeft();

notEat++;

return;

}

else if (neuronOutput[i] != 0 && i == 1)

{

TurnRight();

notEat++;

return;

}

else if (neuronOutput[i] != 0 && i == 2)

{

notEat++;

return;

}

else if (this.GetType() != typeof(Plant) && neuronOutput[i] != 0 && i == 3)

{

Eat(creatures);

notEat = 0;

return;

}

Move();

}

}

public void TurnLeft()

{

switch (direction)

{

case Direction.North:

direction = Direction.West;

perception.UpdatePerceptionFacingWest(Location);

break;

case Direction.West:

direction = Direction.South;

perception.UpdatePerceptionFacingSouth(Location);

break;

case Direction.South:

direction = Direction.East;

perception.UpdatePerceptionFacingEast(Location);

break;

case Direction.East:

direction = Direction.North;

perception.UpdatePerceptionFacingNorth(Location);

break;

}

}

public void TurnRight()

{

switch (direction)

{

case Direction.North:

direction = Direction.East;

perception.UpdatePerceptionFacingEast(Location);

break;

case Direction.West:

direction = Direction.North;

perception.UpdatePerceptionFacingNorth(Location);

break;

case Direction.South:

direction = Direction.West;

perception.UpdatePerceptionFacingWest(Location);

break;

case Direction.East:

direction = Direction.South;

perception.UpdatePerceptionFacingSouth(Location);

break;

}

}

public virtual void Move()

{

switch (direction)

{

case Direction.North:

int newCoordYNorth = this.Location.Y - 1;

if (newCoordYNorth < 0)

this.Location = new Point(this.Location.X, Utilities.WorldSizeY);

else this.Location = new Point(this.Location.X, newCoordYNorth);

break;

case Direction.West:

int newCoordXWest = this.Location.X - 1;

if (newCoordXWest < 0)

this.Location = new Point(Utilities.WorldSizeX, this.Location.Y);

else this.Location = new Point(newCoordXWest, this.Location.Y);

break;

case Direction.South:

int newCoordYSouth = this.Location.Y + 1;

if (newCoordYSouth > Utilities.WorldSizeY)

this.Location = new Point(this.Location.X, Utilities.WorldSizeY);

else this.Location = new Point(this.Location.X, newCoordYSouth);

break;

case Direction.East:

int newCoordXEast = this.Location.X + 1;

if (newCoordXEast > Utilities.WorldSizeX)

this.Location = new Point(0, this.Location.Y);

else this.Location = new Point(newCoordXEast, this.Location.Y);

break;

}

}

public virtual void Eat(List<Creature> creatures)

{

return;

}

protected virtual void MakeNewCreature(List<Creature> creatures)

{

return;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lr5

{

public class Neuron

{

public int[] Input { get; set; }

public double[] Weights { get; set; }

public Neuron()

{

Weights = new double[12];

Input = new int[12];

Random rnd = new Random();

for (int i = 0; i < 12; i++)

{

Weights[i] = rnd.NextDouble()\*2-1;

}

}

public double CalculatePotential()

{

double result = 0;

for (int i = 0; i < Weights.Length; i++)

{

result += Weights[i] \* Input[i];

}

return result;

}

}

}

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lr5.Creatures

{

public class Perception

{

public Point[] frontPerception = new Point[90];

public Point[] leftPerception = new Point[50];

public Point[] rightPerception = new Point[50];

public Point[] nearPerception = new Point[5];

public void UpdatePerceptionFacingNorth(Point location)

{

int X = location.X;

int Y = location.Y;

int frontBias = 2;

int leftBias = 2;

int rightBias = 2;

for (int i = 0; i < frontPerception.Length; i += 5)

{

frontPerception[i] = new Point(X - 2, Y - frontBias);

frontPerception[i + 1] = new Point(X - 1, Y - frontBias);

frontPerception[i + 2] = new Point(X, Y - frontBias);

frontPerception[i + 3] = new Point(X + 1, Y - frontBias);

frontPerception[i + 4] = new Point(X + 2, Y - frontBias);

frontBias += 1;

}

for (int i = 0; i < leftPerception.Length; i += 2)

{

leftPerception[i] = new Point(X - leftBias, Y - 1);

leftPerception[i + 1] = new Point(X - leftBias, Y);

leftBias += 1;

}

for (int i = 0; i < rightPerception.Length; i += 2)

{

rightPerception[i] = new Point(X + rightBias, Y);

rightPerception[i + 1] = new Point(X + rightBias, Y - 1);

rightBias += 1;

}

nearPerception[0] = new Point(X - 1, Y);

nearPerception[1] = new Point(X - 1, Y - 1);

nearPerception[2] = new Point(X, Y - 1);

nearPerception[3] = new Point(X + 1, Y - 1);

nearPerception[4] = new Point(X + 1, Y);

}

public void UpdatePerceptionFacingSouth(Point location)

{

int X = location.X;

int Y = location.Y;

int frontBias = 2;

int leftBias = 2;

int rightBias = 2;

for (int i = 0; i < frontPerception.Length; i += 5)

{

frontPerception[i] = new Point(X - 2, Y + frontBias);

frontPerception[i + 1] = new Point(X - 1, Y + frontBias);

frontPerception[i + 2] = new Point(X, Y + frontBias);

frontPerception[i + 3] = new Point(X + 1, Y + frontBias);

frontPerception[i + 4] = new Point(X + 2, Y + frontBias);

frontBias += 1;

}

for (int i = 0; i < leftPerception.Length; i += 2)

{

leftPerception[i] = new Point(X + leftBias, Y + 1);

leftPerception[i + 1] = new Point(X + leftBias, Y);

leftBias += 1;

}

for (int i = 0; i < rightPerception.Length; i += 2)

{

rightPerception[i] = new Point(X - rightBias, Y);

rightPerception[i + 1] = new Point(X - rightBias, Y + 1);

rightBias += 1;

}

nearPerception[0] = new Point(X - 1, Y);

nearPerception[1] = new Point(X - 1, Y + 1);

nearPerception[2] = new Point(X, Y + 1);

nearPerception[3] = new Point(X + 1, Y + 1);

nearPerception[4] = new Point(X + 1, Y);

}

public void UpdatePerceptionFacingWest(Point location)

{

int X = location.X;

int Y = location.Y;

int frontBias = 2;

int leftBias = 2;

int rightBias = 2;

for (int i = 0; i < frontPerception.Length; i += 5)

{

frontPerception[i] = new Point(X - frontBias, Y - 2);

frontPerception[i + 1] = new Point(X - frontBias, Y - 1);

frontPerception[i + 2] = new Point(X - frontBias, Y);

frontPerception[i + 3] = new Point(X - frontBias, Y + 1);

frontPerception[i + 4] = new Point(X - frontBias, Y + 2);

frontBias += 1;

}

for (int i = 0; i < leftPerception.Length; i += 2)

{

leftPerception[i] = new Point(X - 1, Y + leftBias);

leftPerception[i + 1] = new Point(X, Y + leftBias);

leftBias += 1;

}

for (int i = 0; i < rightPerception.Length; i += 2)

{

rightPerception[i] = new Point(X - 1, Y - rightBias);

rightPerception[i + 1] = new Point(X, Y - rightBias);

rightBias += 1;

}

nearPerception[0] = new Point(X, Y - 1);

nearPerception[1] = new Point(X - 1, Y - 1);

nearPerception[2] = new Point(X - 1, Y);

nearPerception[3] = new Point(X - 1, Y + 1);

nearPerception[4] = new Point(X, Y + 1);

}

public void UpdatePerceptionFacingEast(Point location)

{

int X = location.X;

int Y = location.Y;

int frontBias = 2;

int leftBias = 2;

int rightBias = 2;

for (int i = 0; i < frontPerception.Length; i += 5)

{

frontPerception[i] = new Point(X + frontBias, Y - 2);

frontPerception[i + 1] = new Point(X + frontBias, Y - 1);

frontPerception[i + 2] = new Point(X + frontBias, Y);

frontPerception[i + 3] = new Point(X + frontBias, Y + 1);

frontPerception[i + 4] = new Point(X + frontBias, Y + 2);

frontBias += 1;

}

for (int i = 0; i < leftPerception.Length; i += 2)

{

leftPerception[i] = new Point(X + 1, Y - leftBias);

leftPerception[i + 1] = new Point(X, Y - leftBias);

leftBias += 1;

}

for (int i = 0; i < rightPerception.Length; i += 2)

{

rightPerception[i] = new Point(X + 1, Y + rightBias);

rightPerception[i + 1] = new Point(X, Y + rightBias);

rightBias += 1;

}

nearPerception[0] = new Point(X, Y - 1);

nearPerception[1] = new Point(X + 1, Y - 1);

nearPerception[2] = new Point(X + 1, Y);

nearPerception[3] = new Point(X + 1, Y + 1);

nearPerception[4] = new Point(X, Y + 1);

}

}

}

using lr5.Creatures;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lr5

{

public class Plant : Creature

{

public Plant(int X, int Y)

: base(X,Y)

{

}

public override Pen GetCreaturePen()

{

return new Pen(Color.Green);

}

public override void Move()

{

return;

}

public override void Eat(List<Creature> creatures)

{

return;

}

}

}

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lr5

{

public class Predator : Creature

{

public Predator(int X, int Y) : base(X, Y) { }

public override void Eat(List<Creature> creatures)

{

if (eatHerbIndex > -1)

{

health += 4;

MakeNewCreature(creatures);

}

}

protected override void MakeNewCreature(List<Creature> creatures)

{

Random rnd = new Random();

int childX = Location.X + rnd.Next(-2, 2);

int chlidY = Location.Y + rnd.Next(-2, 2);

if (health >= 20)

{

health = 10;

Predator child = new Predator(childX, chlidY);

for (int i = 0; i < neurons.Count; i++)

{

double[] mutatedWeights = neurons[i].Weights;

for (int j = 0; j < mutatedWeights.Length; j++)

{

mutatedWeights[j] += (rnd.NextDouble() \* 2 - 1) / 2.0;

}

child.neurons[i].Weights = mutatedWeights;

}

creatures.Add(child);

}

}

public override Pen GetCreaturePen()

{

return new Pen(Color.Red);

}

}

}

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lr5.Creatures

{

public class Utilities

{

public static int WorldSizeY = 363;

public static int WorldSizeX = 579;

}

}

**Вывод:** в ходе выполнения лабораторной работы была создана система обучающихся агентов.