Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологий»

Ф.М. Коротков, М.Н. Фофанова

Отчет по курсовой работе «Применение генетического программирования для генерации автомата в задаче о завоевании поля»

Санкт-Петербург
2010

Оглавление

Введение

В последнее время все шире начинает применяться автоматное программирование, в рамках которого поведение программ описывается с помощью конечных детерминированных автоматов [1].

Для многих задач автоматы удается строить эвристически, но существуют задачи, для которых такое построение автоматов затруднительно. К задачам этого класса относятся некоторые задачи генерации автоматов, моделирующих поведение объектов, в частности, и задача об «Умном муравье» [2–4].

1. Постановка задачи

Игра происходит на поверхности тора размером 32 на 32 клетки. На поле сражаются два соперника (два муравья разного цвета). В начальный момент первый соперник находится в клетке с координатами (4, 4) и смотрит влево, а второй – в клетке (29, 29) и смотрит вправо (рис. 1).

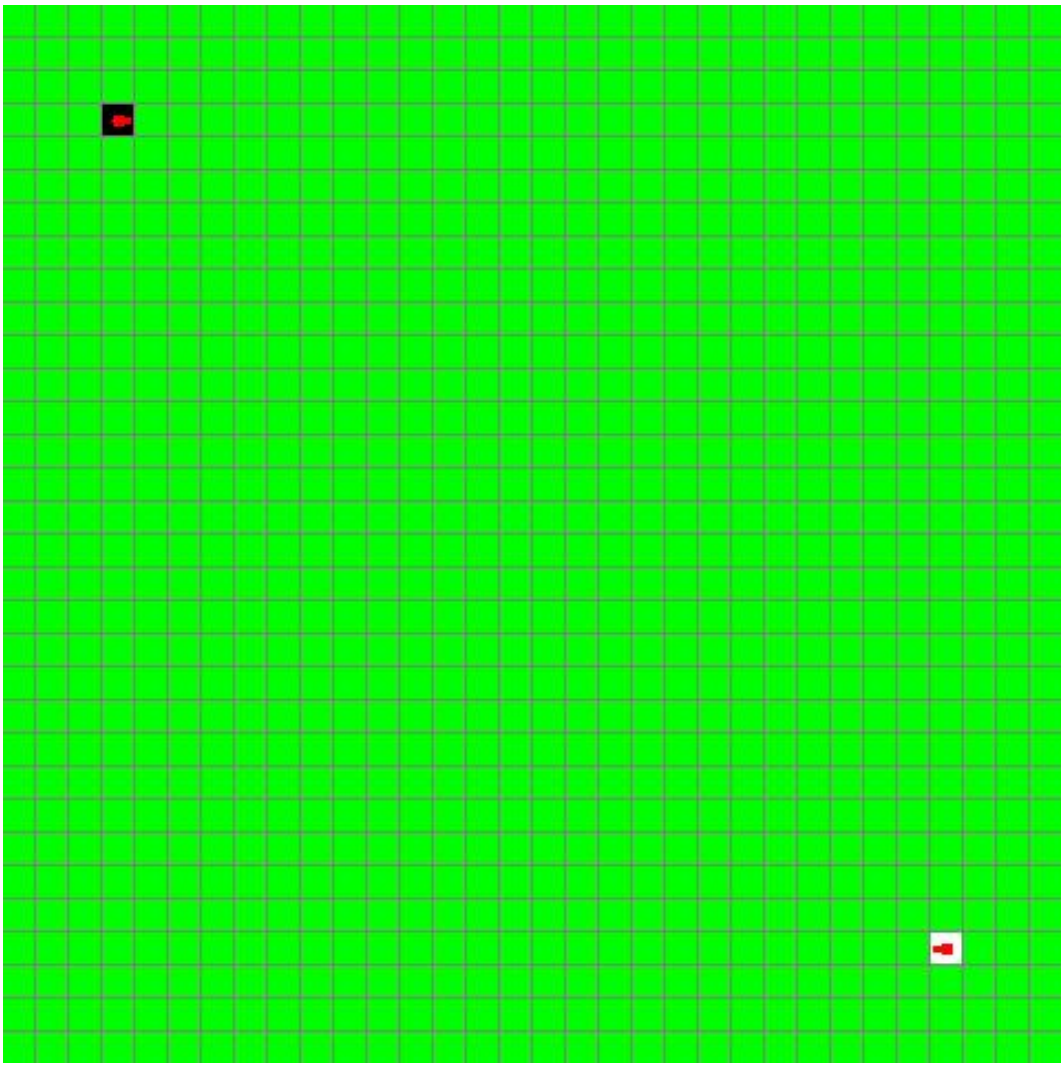

Рис. 1. Начальное состояние поля для проведения состязаний между двумя соперниками

Когда муравей ходит, клетка, в которую он походил, закрашивается в его цвет (черный или белый). Муравей может ходить только в клетки, которые заняты им (своего цвета), или в свободные (зеленые) клетки. Задача муравья состоит в том, чтобы занять как можно больше клеток.

За счет того, что первоначальное положение муравьев симметрично относительно диагонали, безразлично, какой из муравьев будет «белым», а какой «черным».

Когда муравей стоит в клетке, он смотрит в одном из четырех направлений (север, юг, запад, восток).

Муравей умеет определять цвет клетки, находящейся непосредственно перед ним, и за один игровой ход может совершить любое из трех действий:

- сделать шаг вперед, закрасив поле своим цветом, если оно было свободным;
- повернуть налево;
- повернуть направо.

Игра длится 5000 ходов, на каждом из которых муравей совершает одно из трех действий. По истечении 5000 ходов подсчитывается число занятых каждым муравьем клеток и объявляется победитель – тот, кто занял их больше.

Один из способов описания поведения муравья – конечный автомат с действиями на переходах (автомат Мили), имеющий одну входную переменную (цвет клетки перед ним), а множество выходных действий состоит из трех упомянутых выше.

Цель работы – создать автомат, управляющий муравьем, который за 5000 ходов займет как можно больше клеток.

2. Описание интерфейса

Программное средство Conquest Ant (рис. 1) позволяет выбрать размер поколения, количество состояний в генерируемом автомате, управляющем муравьем, вероятность мутации.

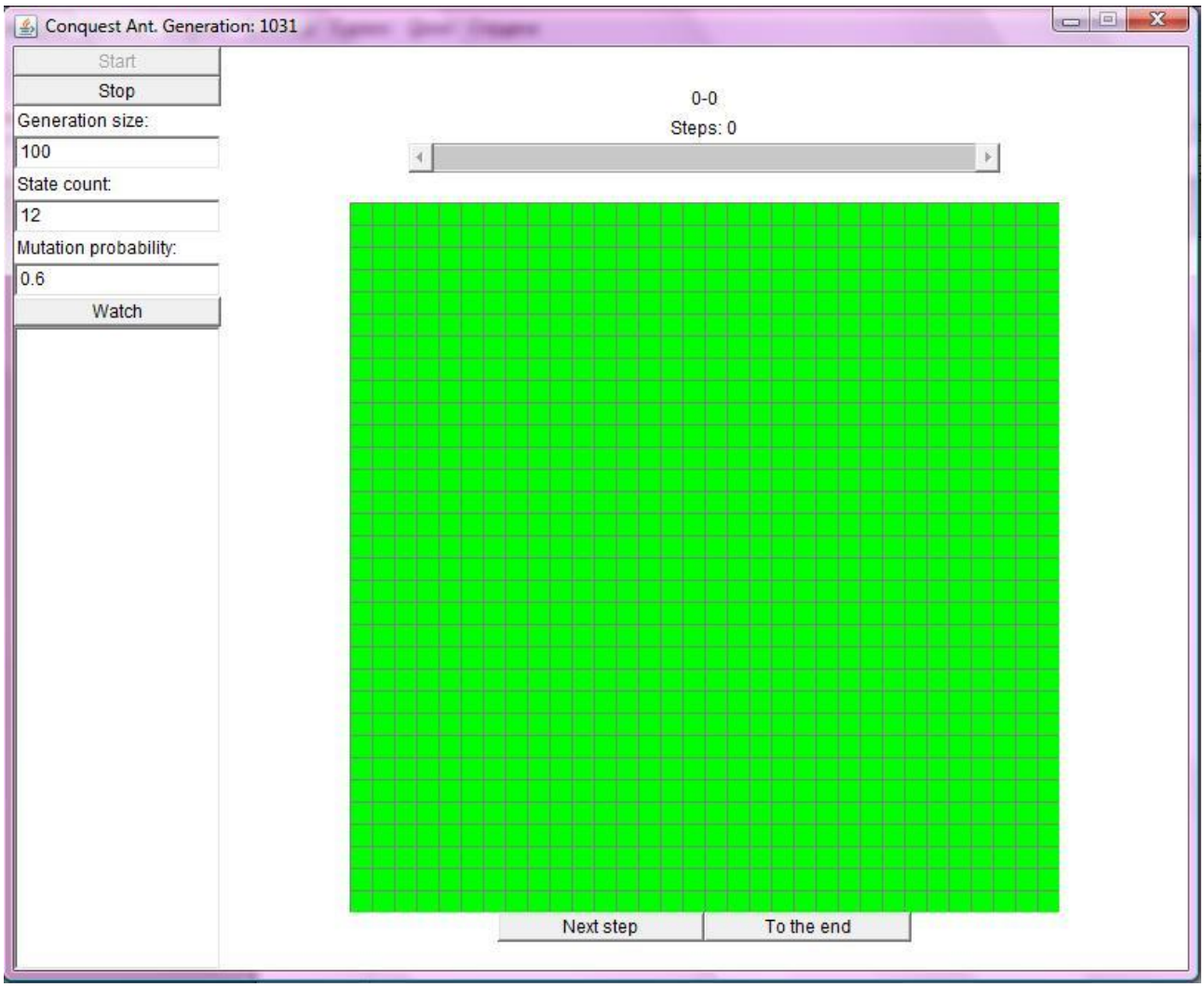

Рис. 2. Интерфейс

После нажатия кнопки `Start` начнется генерация автоматов, и в верхнем левом углу приложения будет указываться номер текущего поколения. После нажатия кнопки `Stop` процесс генерации завершится и в левой части приложения появится список полученных особей. Каждая особь описывается числом партий, выигранных ею в турнире с особями последнего поколения. Для просмотра игры между двумя особями необходимо выбрать их из спика и нажать кнопку `Watch`. Наблюдать за игрой можно пошагово, рассматривая каждый шаг муравьев. Для этого необходимо нажимать кнопку `Next step`. Можно посмотреть любое промежуточное состояние партии с помощью

прокрутки в верхней части приложения (рис. 3) или автоматически завершить игру, нажав кнопку `To the end`.

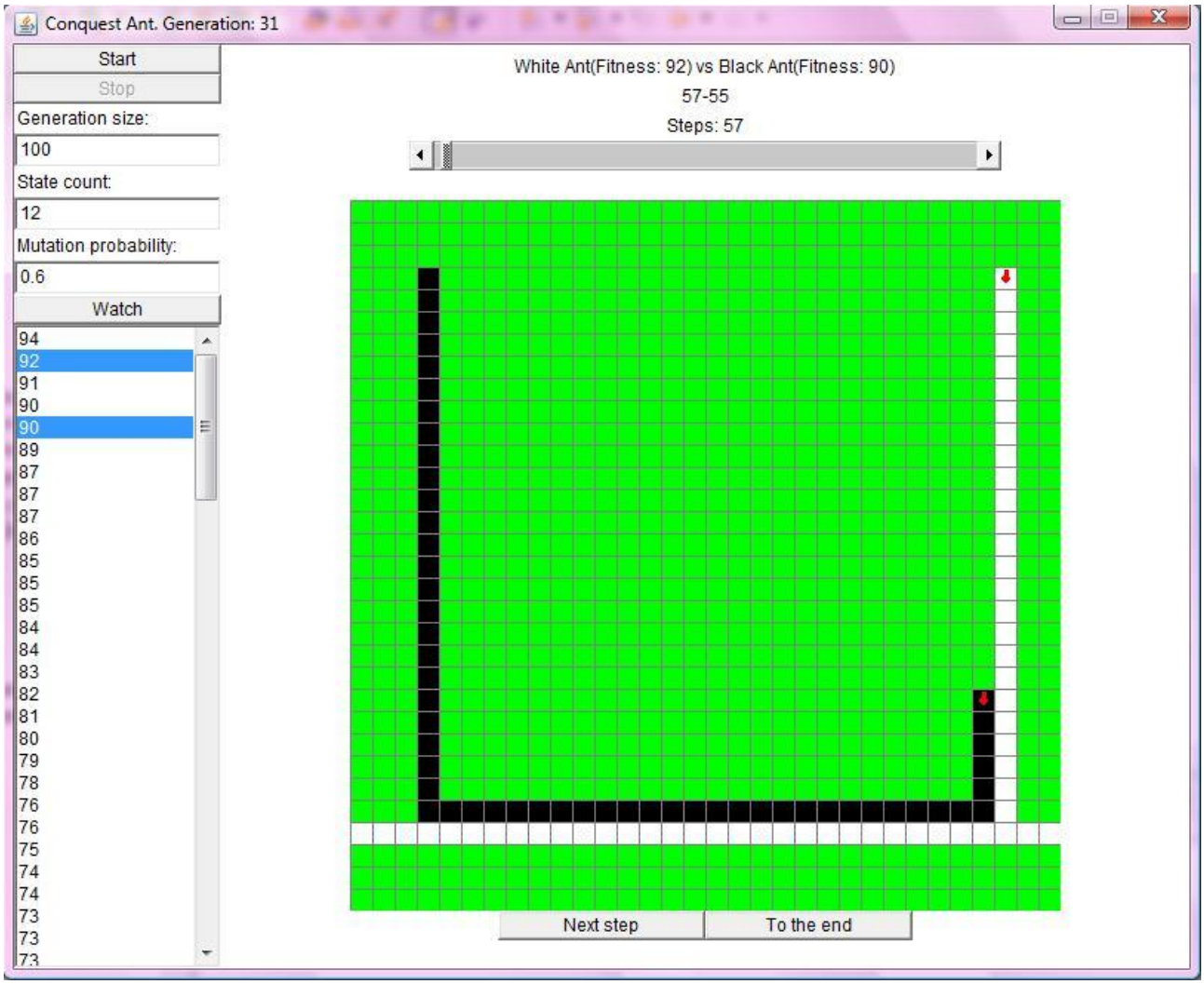

Рис. 3. Промежуточное состояние партии

3. Реализация

Разработанный алгоритм генетического программирования состоит из четырех частей:

- создание начального поколения;
- мутация;
- скрещивание (кроссовер);
- отбор особей для формирования следующего поколения.

Каждая особь представляет собой некоторый автомат, описывающий поведение муравья. Хромосома особи состоит из номера начального состояния и описаний состояний, каждое из которых содержит описания трех переходов, соответствующих тому, какого цвета клетка перед ней (своего, чужого или

зеленого). Описание перехода состоит из номера состояния, в которое он ведет, и действия, выполняемого при выборе этого перехода.

Хромосома представляется не в виде битовой строки, как в генетических алгоритмах, а в виде объекта в языке программирования *Java*. Этот объект имеет следующую структуру:

```
public class Automaton {
    public Transition[][] transitions;
    public int initialState;
    public int stateCount;
}
```

3.1. Создание начального поколения

Начальное поколение состоит из фиксированного числа случайно сгенерированных автоматов. Все автоматы в поколении имеют одинаковое (наперед заданное) число состояний.

3.2. Описание метода мутации

Метод мутации особи состоит в случайном изменении одного из переходов случайного состояния. Кроме того, с вероятностью 50% изменяется начальное состояние на случайно выбранное.

3.3. Описание метода скрещивания

Создаются два автомата (`a1` и `a2`) с таким же числом состояний, как у исходных (`p1` и `p2`). Начальные состояния `a1` и `a2` либо совпадают с `p1` и `p2`, либо с `p2` и `p1` соответственно. Каждое состояние автомата имеет два перехода. Для двух родительских состояний возможны четыре перехода, и четыре варианта распределить их среди детей. Для каждого состояния детей, случайным образом распределяются эти четыре родительских перехода. Таким образом, получим новые особи.

3.4. Генерация нового поколения

Генерация нового поколения состоит из трех этапов. Сначала турнирным методом отбираются особи для дальнейшего скрещивания, а потом происходят мутации. Турнирный метод состоит в том, что выбираются две случайные

особи, проводится турнир между ними, и та, которая займет больше клеток, попадает в новое поколение.

3.5. Способ вычисления приспособленности

В данном исследовании введения функции приспособленности можно избежать. Для выбора лучшей особи каждые 100 поколений проводится турнир «каждый с каждым», и его победитель сравнивается с текущей лучшей особью, и если он побеждает, то становится лучшим.

3.6. Примеры построенных автоматов

На каждом переходе автомата, управляющего муравьем (рис. 4), указано условие, при котором совершается данный переход, и выполняемое действие. Возможны три условия перехода:

- E – клетка впереди свободна;
- OS – клетка впереди занята данным муравьем;
- OO – клетка впереди занята противником.

Муравей на каждом шаге может выполнить следующие действия:

- L – повернуть налево;
- R – повернуть направо;
- M – сделать шаг вперед.

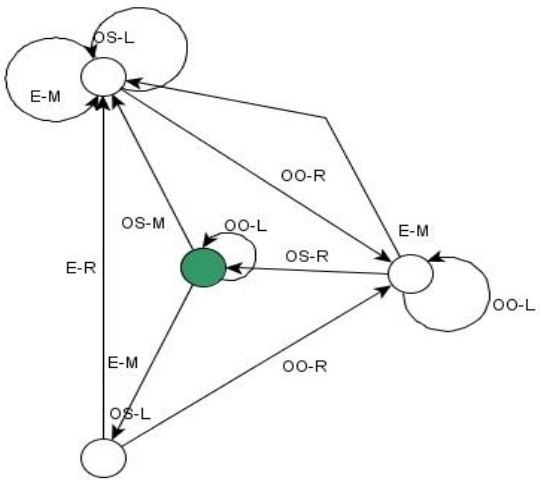

Рис. 4. Автомат, управляющим муравьем, дробящим поле сеткой

4. Результаты

Интересной представляется ситуация, когда противник бездействует (стоит на месте и не двигается). В результате вычислительных экспериментов установлено две интересные закономерности.

При числе состояний больше восьми особь пытается разделить поле сеткой, а потом дробить каждую из ячеек сетки (рис. 5). Это было названо стратегией с дроблением поля сеткой.

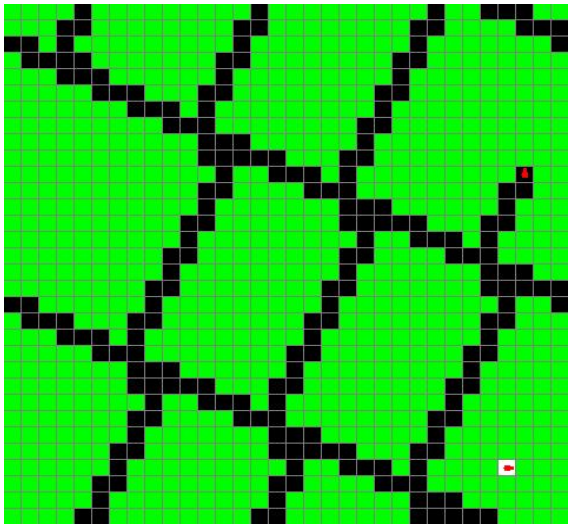

Рис. 5. Поведение особи с девятью состояниями при бездействующем противнике. Дробление поля сеткой

При числе состояний меньше девяти особь пытается захватить поле, двигаясь параллельными линиями (рис.6).

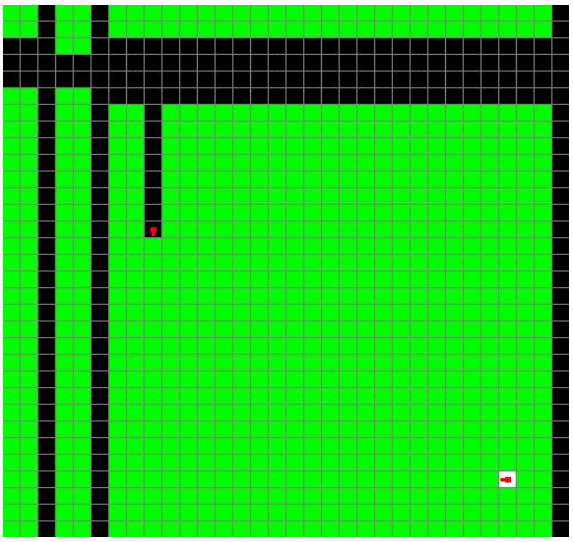

Рис. 6. Поведение особи с восьмью состояниями при бездействующем противнике. Движение параллельными линиями

Такая закономерность может быть связана с тем, что память особи ограничена числом состояний.

Если две особи, двигающиеся параллельными линиями, сражаются друг с другом, каждая старается окружить соперника. Таким образом, из-за симметрии начальных положений и одинаковых стратегий особей траектории муравьев образуют спираль (рис. 7).

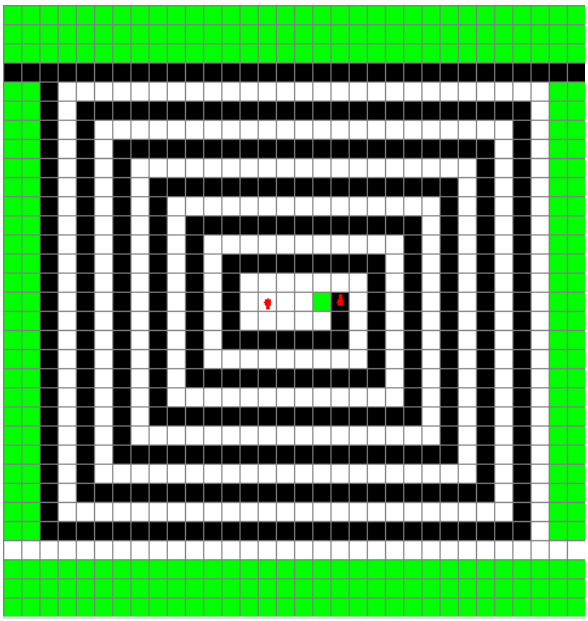

Рис. 7. Спираль, возникающая при сражении двух особей

Следует отметить, что стратегия с дроблением поля сеткой выглядит более перспективной, так как если противник попадет в такую ячейку, он **не сможет выбраться из нее**, потому что по правилам игры нельзя ходить в клетку, которая занята противником.

Во втором случае, если второй муравей начнет двигаться, он будет иметь большую область поля для движения.

Заключение

В данном исследовании была рассмотрена простейшая модель, когда особь располагает информацией только о цвете клетки непосредственно перед ней. Даже при таком упрощении была получена стратегия, свидетельствующая, что данную задачу можно решать, применяя генетические алгоритмы.

Источники

1. *Шалыто А. А.* Технология автоматного программирования / Труды первой Всероссийской научной конференции «Методы и средства обработки информации». М.: МГУ. 2003. http://is.ifmo.ru/works/tech_aut_prog/

2. *Angeline P. J., Pollack J.* Evolutionary Module Acquisition / Proceedings of the Second Annual Conference on Evolutionary Programming. 1993. http://www.demo.cs.brandeis.edu/papers/ep93.pdf

3. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System. 1992. www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html

4. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volume III. CRC Press, 1999.

Приложение

Файл *Main.java*

```
package ru.ifmo.rain;

import java.awt.Canvas;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

@SuppressWarnings("serial")
public class Main extends Canvas {

	public static void main(String arg[]) {
		final ConquestAntFrame f = new ConquestAntFrame("Conquest Ant.");
		f.addWindowListener(new WindowAdapter() {
			public void windowClosing(WindowEvent e) {
				f.dispose();
			}
		});

	    f.setSize(800, 650);
	    f.setResizable(false);
	    f.initApplication();
	    f.setVisible(true);
	  }
}
```

Файл *ConquestAntFrame.java*

```
package ru.ifmo.rain;

import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.Label;
import java.awt.List;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.Box;
import javax.swing.Timer;

import ru.ifmo.rain.components.AntViewer;
import ru.ifmo.rain.problem.Automation;
import ru.ifmo.rain.problem.Mover;
import ru.ifmo.rain.problem.SimpleGeneticAlgorithm;

@SuppressWarnings("serial")
public class ConquestAntFrame extends Frame{
	/**
	 *
	 */
	private static final int defaultComponentWidth = 140;
	private static final int defaultComponentHeight = 20;
```

```
	/**
	 * Кнопка старта генерации поколений.
	 */
	private Button startBtn;
	/**
	 * Кнопка остановки генерации поколений.
	 */
	private Button stopBtn;
	/**
	 * Список особей.
	 */
	private List antsList;
	/*
	 *
	 */
	private TextField mutationTxt;
	/*
	 *
	 */
	private TextField crossOverTxt;
	/*
	 *
	 */
	private TextField stateCountTxt;
	/*
	 *
	 */
	private TextField generationSizeTxt;
	/*
	 *
	 */
	private SimpleGeneticAlgorithm algorithm;
	/*
	 *
	 */
	private AntViewer viewer;

	/*
	 *
	 */
	private Timer timer;

	private Automation[] generation;
	private int[] wc;

	public ConquestAntFrame(String string) {
		// TODO Auto-generated constructor stub
		super(string);
	}
	public void initApplication(){
		setLayout(new BorderLayout());
		initLeftPanel();
	}
	private void initLeftPanel(){
		Box b = Box.createVerticalBox();

		startBtn = new Button("Start");
		startBtn.setMinimumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
		startBtn.setMaximumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
		startBtn.setPreferredSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
```

```
            startBtn.setSize(defaultComponentWidth, defaultComponentHeight);
            startBtn.addActionListener(new ActionListener(){

                  @Override
                  public void actionPerformed(ActionEvent e) {
                        // TODO Auto-generated method stub
                        antsList.removeAll();
                        int gc = 0, sc = 0;
                        double mp = 0.0;
                        try{
                              gc =
Integer.parseInt(generationSizeTxt.getText());
                        }catch(Throwable ex){
                              generationSizeTxt.setText("Bad:
"+generationSizeTxt.getText());
                              return;
                        }
                        if(gc <= 0){
                              generationSizeTxt.setText("Bad:
"+generationSizeTxt.getText());
                              return;
                        }
                        try{
                              sc = Integer.parseInt(stateCountTxt.getText());
                        }catch(Throwable ex){
                              stateCountTxt.setText("Bad:
"+stateCountTxt.getText());
                              return;
                        }
                        if(sc <= 0){
                              stateCountTxt.setText("Bad:
"+stateCountTxt.getText());
                              return;
                        }
                        try{
                              mp = Double.parseDouble(mutationTxt.getText());
                        }catch(Throwable ex){
                              mutationTxt.setText("Bad:
"+mutationTxt.getText());
                              return;
                        }
                        if(mp < 0.0 || mp > 1.0){
                              mutationTxt.setText("Bad:
"+mutationTxt.getText());
                              return;
                        }
                        algorithm = new SimpleGeneticAlgorithm(gc,sc,mp);
                        startBtn.setEnabled(false);
                        stopBtn.setEnabled(true);
                        algorithm.start();
                        timer = new Timer(1000,new ActionListener(){

                              @Override
                              public void actionPerformed(ActionEvent e) {
                                    // TODO Auto-generated method stub
                                    if(algorithm != null){
                                          setTitle("Conquest Ant. Generation:
"+Integer.toString(algorithm.getGenerationNumber()));
                                    }
                              }
                        });
                        timer.start();
                  }
```

```
            });
            b.add(startBtn);

            stopBtn = new Button("Stop");
            stopBtn.setMinimumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            stopBtn.setMaximumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            stopBtn.setPreferredSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            stopBtn.setSize(defaultComponentWidth, defaultComponentHeight);
            stopBtn.setEnabled(false);
            stopBtn.addActionListener(new ActionListener(){

                  @Override
                  public void actionPerformed(ActionEvent e) {
                        // TODO Auto-generated method stub
                        stopBtn.setEnabled(false);

                        Automation[] g = algorithm.getGeneration();
                        Automation[] b = algorithm.getBestIndividuals();

                        algorithm.stop();
                        timer.stop();
                        timer = null;
                        algorithm = null;

                        generation = new Automation[g.length + b.length];

                        for(int i = 0; i < g.length; ++i){
                              generation[i] = g[i];
                        }
                        for(int i = 0; i < b.length; ++i){
                              generation[g.length + i] = b[i];
                        }

                        wc = Mover.getWinCountArray(generation);

                        int n = wc.length;
                      for (int pass = 1; pass < n; pass++) {
                         for (int i=0; i < n-pass; i++) {
                             if (wc[i] < wc[i+1]) {
                                 int temp1 = wc[i];  wc[i] = wc[i+1];
wc[i+1] = temp1;
                                 Automation temp2 = generation[i];
generation[i] = generation[i+1];  generation[i+1] = temp2;
                             }
                         }
                      }
                      for(int i = 0; i < wc.length; ++i){
                              antsList.add(Integer.toString(wc[i]));
                        }
                        startBtn.setEnabled(true);
                  }

            });
            b.add(stopBtn);

            Label gl = new Label("Generation size:");
            gl.setMinimumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            gl.setMaximumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
```

```
            gl.setPreferredSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            gl.setSize(defaultComponentWidth, defaultComponentHeight);
            b.add(gl);

            generationSizeTxt = new TextField("100");
            generationSizeTxt.setMinimumSize(new
Dimension(defaultComponentWidth, defaultComponentHeight));
            generationSizeTxt.setMaximumSize(new
Dimension(defaultComponentWidth, defaultComponentHeight));
            generationSizeTxt.setPreferredSize(new
Dimension(defaultComponentWidth, defaultComponentHeight));
            generationSizeTxt.setSize(defaultComponentWidth,
defaultComponentHeight);
            b.add(generationSizeTxt);

            Label scl = new Label("State count:");
            scl.setMinimumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            scl.setMaximumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            scl.setPreferredSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            scl.setSize(defaultComponentWidth, defaultComponentHeight);
            b.add(scl);

            stateCountTxt = new TextField("12");
            stateCountTxt.setMinimumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            stateCountTxt.setMaximumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            stateCountTxt.setPreferredSize(new
Dimension(defaultComponentWidth, defaultComponentHeight));
            stateCountTxt.setSize(defaultComponentWidth,
defaultComponentHeight);
            b.add(stateCountTxt);

            Label mpl = new Label("Mutation probability:");
            mpl.setMinimumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            mpl.setMaximumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            mpl.setPreferredSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            mpl.setSize(defaultComponentWidth, defaultComponentHeight);
            b.add(mpl);

            mutationTxt = new TextField("0.6");
            mutationTxt.setMinimumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            mutationTxt.setMaximumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            mutationTxt.setPreferredSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            mutationTxt.setSize(defaultComponentWidth,
defaultComponentHeight);
            b.add(mutationTxt);

            Button watch = new Button("Watch");
            watch.setMinimumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            watch.setMaximumSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
```

```
            watch.setPreferredSize(new Dimension(defaultComponentWidth,
defaultComponentHeight));
            watch.setSize(defaultComponentWidth, defaultComponentHeight);
            watch.addActionListener(new ActionListener(){

                  @Override
                  public void actionPerformed(ActionEvent e) {
                        // TODO Auto-generated method stub
                        int[] selected = antsList.getSelectedIndexes();

                        if(selected.length < 2){
                              return;
                        }
                        int[] fitness = {wc[selected[0]],wc[selected[1]]};
                        Automation[] a =
{generation[selected[0]],generation[selected[1]]};
                        viewer.watch(a, fitness);
                  }
            });
            b.add(watch);

            antsList = new List(0,true);
            antsList.setSize(defaultComponentWidth, this.getHeight() -
2*defaultComponentHeight);
            b.add(antsList);

            this.add(b,BorderLayout.WEST);

            viewer = new AntViewer();
            this.add(viewer,BorderLayout.CENTER);
      }
}
```

Файл *Transition.java*

```
package ru.ifmo.rain.phenotype;


public class Transition {
      public int endState;
      public AntAction action;

      public Transition(int endState, AntAction action)
    {
            this.endState = endState;
        this.action = action;
    }

    public String toString()
    {
        return "-{"+action+"}->{"+endState+"}";
    }
}
```

Файл *AntAction.java*

```
package ru.ifmo.rain.phenotype;

public enum AntAction {
      Left,
    Right,
    Move
```

```
}
```

Файл *Cell.java*

```
package ru.ifmo.rain.phenotype;

public class Cell {
      public int x;
      public int y;

      public Cell(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public Cell next(Direction direction)
    {
        switch (direction)
        {
            case Left:
                return new Cell((x + 31)%32, y);
            case Up:
                return new Cell(x, (y + 1)%32);
            case Right:
                return new Cell((x + 1)%32, y);
            case Down:
                return new Cell(x, (y + 31)%32);
            default:
                throw new AssertionError("bad direction");
        }
    }
}
```

Файл *CellType.java*

```
package ru.ifmo.rain.phenotype;

public enum CellType {
      Empty,
    Black,
    White
}
```

Файл *Direction.java*

```
package ru.ifmo.rain.phenotype;

public enum Direction {
      Up,
    Right,
    Down,
      Left
}
```

Файл *Automation.java*

```
package ru.ifmo.rain.problem;

import ru.ifmo.rain.phenotype.Transition;

public class Automation implements Comparable<Automation> {
      private Transition[][] transitions;
```

```
      private int initialState;
      private int statesCount;

      public Automation(int initialState, int statesCount)
    {
        this.initialState = initialState;
        this.statesCount = statesCount;
        transitions = new Transition[statesCount][3];
    }

      public int getInitialState(){
            return initialState;
      }
      public void setInitialState(int initialState){
            this.initialState = initialState;
      }
      public int getStatesCount(){
            return statesCount;
      }
      public void setStatesCount(int statesCount){
            this.statesCount = statesCount;
      }
      public Transition getTransition(int index, int condition)
    {
        return transitions[index][condition];
    }

    public void setTransition(int index, int condition, Transition
transition)
    {
        transitions[index][condition] = transition;
    }

    public Automation clone(){
      Automation res = new Automation(initialState, statesCount);
      for (int i = 0; i < res.getStatesCount(); i++)
        {
            Transition t = getTransition(i, 2);
            res.setTransition(i, 2, new Transition(t.endState, t.action));
            t = getTransition(i, 1);
            res.setTransition(i, 1, new Transition(t.endState, t.action));
            t = getTransition(i, 0);
            res.setTransition(i, 0, new Transition(t.endState, t.action));
        }
      return res;
    }

      @Override
      public int compareTo(Automation o) {
            // TODO Auto-generated method stub
            Automation[] mas = new Automation[2];
            mas[0] = this;
            mas[1] = o;

            Mover m = new Mover(mas);
            int winner = m.getWinner();
            if(winner == 0){
                  return 1;
            }else{
                  return -1;
            }
      }
}
```

Файл *Mover.java*

```
package ru.ifmo.rain.problem;

import ru.ifmo.rain.phenotype.Ant;
import ru.ifmo.rain.phenotype.AntAction;
import ru.ifmo.rain.phenotype.CellType;
import ru.ifmo.rain.phenotype.Direction;

public class Mover {
	private Ant antBlack;
	private Ant antWhite;

	private CellType[][] field;

	private int step = 0;

	private int blackCells = 1;
	private int whiteCells = 1;

	private Automation[] automationSource;

	public static final int MAX_STEP = 5000;

	public Mover(Automation[] a){
		automationSource = a.clone();
		reset();
	}
	public void reset(){
		field = new CellType[32][32];

		step = 0;

		blackCells = 1;
		whiteCells = 1;

		for(int i = 0; i < 32; ++i){
			for(int j = 0; j < 32; ++j){
				field[i][j] = CellType.Empty;
			}
		}
```

```
            antWhite = new
Ant(automationSource[0].clone(),28,28,Direction.Left);
            antBlack = new
Ant(automationSource[1].clone(),3,3,Direction.Right);


            field[3] [3]  = CellType.Black;
            field[28][28] = CellType.White;
      }
      public void moveTo(int stepTo){
            reset();
            int i = 1;
            while(move() && i < stepTo){
                  ++i;
            }
      }
      public boolean move(){
            if(step >= MAX_STEP){
                  return false;
            }
            AntAction black = antBlack.Move(field, CellType.Black);
            if(black == AntAction.Move){
                  if(field[antBlack.getX()][antBlack.getY()] ==
CellType.White){
                        //Imposible, but...
                        throw new Error("Ant want occupy other ant's cell!!!
O_o");
                  }
                  if(field[antBlack.getX()][antBlack.getY()] ==
CellType.Empty){
                        ++blackCells;
                  }
                  field[antBlack.getX()][antBlack.getY()] = CellType.Black;
            }
            AntAction white = antWhite.Move(field, CellType.White);
            if(white == AntAction.Move){
                  if(field[antWhite.getX()][antWhite.getY()] ==
CellType.Black){
                        //Imposible, but...
                        throw new Error("Ant want occupy other ant's cell!!!
O_o");
                  }
                  if(field[antWhite.getX()][antWhite.getY()] ==
CellType.Empty){
                        ++whiteCells;
                  }
```

```
            field[antWhite.getX()][antWhite.getY()] = CellType.White;
      }
      ++step;
      return step < MAX_STEP;
}
public Ant getAntBlack(){
      return antBlack;
}
public Ant getAntWhite(){
      return antWhite;
}
public int getWhiteCellsCount(){
      return whiteCells;
}
public int getBlackCellsCount(){
      return blackCells;
}
public CellType[][] getField(){
      return field;
}
public int getStep(){
      return step;
}
public void toTheEnd(){
      while(move()){}
}
public int getWinner(){
      toTheEnd();
      return whiteCells > blackCells ? 0 : 1;
}
public static int[] getWinCountArray(Automation[] generation){
      int[] winCount = new int[generation.length];
      for(int i = 0; i < generation.length; ++i){
            for(int j = i + 1; j < generation.length; ++j){
                  Automation[] toFight = new Automation[2];
                  toFight[0] = generation[i];
                  toFight[1] = generation[j];

                  Mover m = new Mover(toFight);

                  int winner = m.getWinner();
```

```
                        if(winner == 0){
                              ++winCount[i];
                        }else{
                              ++winCount[j];
                        }
                  }
            }
            return winCount;
      }
      public static int getWinnerIndex(Automation[] generation){
            int[] winCount = getWinCountArray(generation);

            int max = winCount[0];
            int maxId = 0;
            for(int i = 1; i < winCount.length;++i){
                  if(winCount[i] > max){
                        max = winCount[i];
                        maxId = i;
                  }
            }
            return maxId;
      }
      public static int getLooserIndex(Automation[] generation){
            int[] winCount = getWinCountArray(generation);

            int min = winCount[0];
            int minId = 0;
            for(int i = 1; i < winCount.length;++i){
                  if(winCount[i] > min){
                        min = winCount[i];
                        minId = i;
                  }
            }
            return minId;
      }
}
```

Файл *SearchOperatorOnAutomation.java*

```
package ru.ifmo.rain.problem;

import ru.ifmo.rain.phenotype.AntAction;
```

```
import ru.ifmo.rain.phenotype.Transition;

public class SearchOperatorOnAutomation {
      public static Automation createRandom(int stateCount)
    {
        Automation automation = new Automation(0, stateCount);
        for (int i = 0; i < stateCount; i++){
            for (int j = 0; j < 3; j++){
                  if(j == 2){
                        //нельзя заходить на чужое
                        automation.setTransition(i, j,new
Transition((int)Math.floor(Math.random()*stateCount), getRandomAction(2)));
                  }else{
                        automation.setTransition(i, j,new
Transition((int)Math.floor(Math.random()*stateCount), getRandomAction(3)));
                  }
            }
        }

        return automation;
    }
      public static AntAction getRandomAction(int actions)
    {
        AntAction action = AntAction.Left;
        switch ((int)Math.floor(Math.random()*actions))
        {
            case 0:
                action = AntAction.Left;
                break;
            case 1:
                action = AntAction.Right;
                break;
            case 2:
                action = AntAction.Move;
                break;
        }
        return action;
    }
      public static Automation Mutate(Automation oldAutomation)
    {
        Automation aa = oldAutomation;
        if (aa == null)
            return oldAutomation;
```

```
        int state = (int)Math.floor(Math.random()*aa.getStatesCount());
        int c = (int)Math.floor(Math.random()*2);

        int iss = aa.getInitialState();
        if ((int)Math.floor(Math.random()*2) == 1)
            iss = (int)Math.floor(Math.random()*aa.getStatesCount());;

        Automation res = new Automation(iss, aa.getStatesCount());

        for (int i = 0; i < res.getStatesCount(); i++)
        {
            Transition t = aa.getTransition(i, 2);
            res.setTransition(i, 2, new Transition(t.endState, t.action));
            t = aa.getTransition(i, 1);
            res.setTransition(i, 1, new Transition(t.endState, t.action));
            t = aa.getTransition(i, 0);
            res.setTransition(i, 0, new Transition(t.endState, t.action));
        }

        int es = (int)Math.floor(Math.random()*res.getStatesCount());

        if(c == 2){
            res.setTransition(state, c, new Transition(es,
getRandomAction(2)));
        }else{
            res.setTransition(state, c, new Transition(es,
getRandomAction(3)));
        }

        return res;
    }

    public static Automation[] Crossover(Automation[] Automations)
    {
        Automation aa1 = Automations[0];
        Automation aa2 = Automations[1];

        if (aa1 == null || aa2 == null)
            return Automations;

        Automation[] s = new Automation[2];
        for (int i = 0; i < 2; i++)
            s[i] = new Automation(0, aa1.getStatesCount());
```

```
if ((int)Math.floor(Math.random()*2) == 1)
{
    s[0].setInitialState(aa2.getInitialState());
    s[1].setInitialState(aa1.getInitialState());
}
else
{
    s[0].setInitialState(aa1.getInitialState());
    s[1].setInitialState(aa2.getInitialState());
}

for (int i = 0; i < aa1.getStatesCount(); i++)
{
    int flag = (int)Math.floor(Math.random()*8);
    switch (flag)
    {
        case 0:
            s[0].setTransition(i, 0, aa2.getTransition(i, 0));
            s[0].setTransition(i, 1, aa2.getTransition(i, 1));
            s[0].setTransition(i, 2, aa2.getTransition(i, 2));
            s[1].setTransition(i, 0, aa1.getTransition(i, 0));
            s[1].setTransition(i, 1, aa1.getTransition(i, 1));
            s[1].setTransition(i, 2, aa1.getTransition(i, 2));
            break;
        case 1:
            s[0].setTransition(i, 0, aa2.getTransition(i, 0));
            s[0].setTransition(i, 1, aa2.getTransition(i, 1));
            s[0].setTransition(i, 2, aa1.getTransition(i, 2));
            s[1].setTransition(i, 0, aa1.getTransition(i, 0));
            s[1].setTransition(i, 1, aa1.getTransition(i, 1));
            s[1].setTransition(i, 2, aa2.getTransition(i, 2));
            break;
        case 2:
            s[0].setTransition(i, 0, aa1.getTransition(i, 0));
            s[0].setTransition(i, 1, aa1.getTransition(i, 1));
            s[0].setTransition(i, 2, aa1.getTransition(i, 2));
            s[1].setTransition(i, 0, aa2.getTransition(i, 0));
            s[1].setTransition(i, 1, aa2.getTransition(i, 1));
            s[1].setTransition(i, 2, aa2.getTransition(i, 2));
            break;
        case 3:
```

```
                s[0].setTransition(i, 0, aa1.getTransition(i, 0));
                s[0].setTransition(i, 1, aa1.getTransition(i, 1));
                s[0].setTransition(i, 2, aa2.getTransition(i, 2));
                s[1].setTransition(i, 0, aa2.getTransition(i, 0));
                s[1].setTransition(i, 1, aa2.getTransition(i, 1));
                s[1].setTransition(i, 2, aa1.getTransition(i, 2));
                break;
            case 4:
                s[0].setTransition(i, 0, aa2.getTransition(i, 0));
                s[0].setTransition(i, 1, aa1.getTransition(i, 1));
                s[0].setTransition(i, 2, aa1.getTransition(i, 2));
                s[1].setTransition(i, 0, aa2.getTransition(i, 0));
                s[1].setTransition(i, 1, aa1.getTransition(i, 1));
                s[1].setTransition(i, 2, aa2.getTransition(i, 2));
                break;
            case 5:
                s[0].setTransition(i, 0, aa2.getTransition(i, 0));
                s[0].setTransition(i, 1, aa1.getTransition(i, 1));
                s[0].setTransition(i, 2, aa2.getTransition(i, 2));
                s[1].setTransition(i, 0, aa2.getTransition(i, 0));
                s[1].setTransition(i, 1, aa1.getTransition(i, 1));
                s[1].setTransition(i, 2, aa1.getTransition(i, 2));
                break;
            case 6:
                s[0].setTransition(i, 0, aa1.getTransition(i, 0));
                s[0].setTransition(i, 1, aa2.getTransition(i, 1));
                s[0].setTransition(i, 2, aa2.getTransition(i, 2));
                s[1].setTransition(i, 0, aa1.getTransition(i, 0));
                s[1].setTransition(i, 1, aa2.getTransition(i, 1));
                s[1].setTransition(i, 2, aa1.getTransition(i, 2));
                break;
            case 7:
                s[0].setTransition(i, 0, aa1.getTransition(i, 0));
                s[0].setTransition(i, 1, aa2.getTransition(i, 1));
                s[0].setTransition(i, 2, aa1.getTransition(i, 2));
                s[1].setTransition(i, 0, aa1.getTransition(i, 0));
                s[1].setTransition(i, 1, aa2.getTransition(i, 1));
                s[1].setTransition(i, 2, aa2.getTransition(i, 2));
                break;
        }
    }
```

```
        return s;
    }
}
```

Файл *SimpleGeneticAlgorithm.java*

```
package ru.ifmo.rain.problem;


public class SimpleGeneticAlgorithm extends Thread{
	private int generationSize;
	private int stateCount;
	private double mutationProbability;

	private Automation[] generation;

	private Automation currentBestIndividual;

	private Automation[] bestIndividuals;

	private int generationNumber = 1;

	public SimpleGeneticAlgorithm(int generationSize, int stateCount, double
mutationProbability){
		this.generationSize = generationSize;
		this.stateCount = stateCount;
		this.mutationProbability = mutationProbability;
		this.generation = new Automation[this.generationSize];
		this.bestIndividuals = new Automation[this.generationSize/10];

		for(int i = 0; i < generation.length; ++i){
			generation[i] =
SearchOperatorOnAutomation.createRandom(this.stateCount);
		}
		for(int i = 0; i < bestIndividuals.length; ++i){
			bestIndividuals[i] = generation[i].clone();
		}

		currentBestIndividual = generation[0].clone();
	}
	@Override
	public void run() {
		// TODO Auto-generated method stub
		while(true){
			nextGeneration();
		}
	}
	public int getGenerationNumber(){
		return generationNumber;
	}
	public Automation[] getGeneration(){
		return generation;
	}
	public Automation[] getBestIndividuals(){
		return bestIndividuals;
	}
	private void setBestIndividual(Automation bi){
		currentBestIndividual = bi.clone();
		int looser = Mover.getLooserIndex(bestIndividuals);
		if(currentBestIndividual.compareTo(bestIndividuals[looser]) > 0){
			bestIndividuals[looser] = currentBestIndividual;
		}
	}
```

```
      public void nextGeneration() {
            //используем турнирный отбор
            Automation[] newGeneration = new Automation[generation.length];
            for(int i = 0; i < newGeneration.length; ++i){
                  int index1 =
(int)Math.floor(Math.random()*generation.length);
                  int index2 =
(int)Math.floor(Math.random()*generation.length);

                  Automation[] toFight = new Automation[2];

                  toFight[0] = generation[index1];
                  toFight[1] = generation[index2];

                  Mover m = new Mover(toFight);

                  int winnerIndex = m.getWinner();

                  newGeneration[i] = toFight[winnerIndex];
            }
            for(int i = 0; i < newGeneration.length / 2; ++i){
                  int index1 = (int) Math.floor(Math.random() *
newGeneration.length);
                  int index2 = (int) Math.floor(Math.random() *
newGeneration.length);

                  Automation[] toCrossOver = new Automation[2];

                  toCrossOver[0] = newGeneration[index1];
                  toCrossOver[1] = newGeneration[index2];

                  Automation[] crossOvered = SearchOperatorOnAutomation
                              .Crossover(toCrossOver);

                  newGeneration[index1] = crossOvered[0];
                  newGeneration[index2] = crossOvered[1];
            }
            for(int i = 0; i < newGeneration.length; ++i){
                  if(Math.random() < mutationProbability){
                        newGeneration[i] =
SearchOperatorOnAutomation.Mutate(newGeneration[i]);
                  }
            }
            generation = newGeneration;
            ++generationNumber;
            if(generationNumber % 100 == 0){
                  int index = Mover.getWinnerIndex(generation);
                  if(currentBestIndividual.compareTo(generation[index]) < 0){
                        setBestIndividual(generation[index]);
                  }
                  BigMutation();
            }
      }
      private void BigMutation()
    {
        for (int i = 0; i < generation.length; i++){
            generation[i] = SearchOperatorOnAutomation.Mutate(generation[i]);
        }
    }
}
```

Файл *AntViewer.java*

```
package ru.ifmo.rain.components;
```

```
import java.awt.Button;
import java.awt.Dimension;
import java.awt.Label;
import java.awt.Panel;
import java.awt.Scrollbar;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.AdjustmentEvent;
import java.awt.event.AdjustmentListener;

import javax.swing.Box;

import ru.ifmo.rain.problem.Automation;
import ru.ifmo.rain.problem.Mover;

public class AntViewer extends Panel {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	private Label desc;

	private Label score;

	private Label steps;

	private Field field;

	private Scrollbar history;

	private Mover m;

	public AntViewer(){
		this.setSize(new Dimension(480,600));
		this.setMinimumSize(new Dimension(480, 600));
		this.setMaximumSize(new Dimension(480, 600));
		this.setPreferredSize(new Dimension(480, 600));
		Box vb = Box.createVerticalBox();
```

```
            desc = new Label("");
            desc.setMinimumSize(new Dimension(480, 20));
            desc.setMaximumSize(new Dimension(480, 20));
            desc.setPreferredSize(new Dimension(480, 20));
            desc.setSize(480, 20);
            desc.setAlignment(Label.CENTER);
            vb.add(desc);

            score = new Label("0-0");
            score.setMinimumSize(new Dimension(480, 20));
            score.setMaximumSize(new Dimension(480, 20));
            score.setPreferredSize(new Dimension(480, 20));
            score.setSize(480, 20);
            score.setAlignment(Label.CENTER);
            vb.add(score);

            steps = new Label("Steps: 0");
            steps.setMinimumSize(new Dimension(480, 20));
            steps.setMaximumSize(new Dimension(480, 20));
            steps.setPreferredSize(new Dimension(480, 20));
            steps.setSize(480, 20);
            steps.setAlignment(Label.CENTER);
            vb.add(steps);

            history = new
Scrollbar(Scrollbar.HORIZONTAL,0,20,1,Mover.MAX_STEP);
            history.setMinimumSize(new Dimension(400, 20));
            history.setMaximumSize(new Dimension(400, 20));
            history.setPreferredSize(new Dimension(400, 20));
            history.setSize(400, 20);
            vb.add(history);

            history.setEnabled(false);

            history.addAdjustmentListener(new AdjustmentListener(){

                  @Override
                  public void adjustmentValueChanged(AdjustmentEvent e) {
                        // TODO Auto-generated method stub
                        int stepTo = history.getValue();
                        System.out.println(stepTo);
                        m.moveTo(stepTo);
```

```
                        field.drawField(m.getField(), m.getAntWhite(),
m.getAntBlack());
                        steps.setText("Steps:
"+Integer.toString(m.getStep()));

      score.setText(Integer.toString(m.getWhiteCellsCount())+"-
"+Integer.toString(m.getBlackCellsCount()));
                  }});

            Label sep = new Label("");
            sep.setMinimumSize(new Dimension(480, 20));
            sep.setMaximumSize(new Dimension(480, 20));
            sep.setPreferredSize(new Dimension(480, 20));
            sep.setSize(480, 20);
            sep.setAlignment(Label.CENTER);
            vb.add(sep);

            field = new Field();
            field.setMinimumSize(new Dimension(480, 480));
            field.setMaximumSize(new Dimension(480, 480));
            field.setPreferredSize(new Dimension(480, 480));
            field.setSize(480, 480);
            vb.add(field);

            Box hb = Box.createHorizontalBox();

            Button nextStep = new Button("Next step");
            nextStep.setMinimumSize(new Dimension(140, 20));
            nextStep.setMaximumSize(new Dimension(140, 20));
            nextStep.setPreferredSize(new Dimension(140, 20));
            nextStep.setSize(140, 20);
            nextStep.addActionListener(new ActionListener(){

                  @Override
                  public void actionPerformed(ActionEvent e) {
                        // TODO Auto-generated method stub
                        nextStep();
                  }});
            hb.add(nextStep);

            Button toTheEnd = new Button("To the end");
            toTheEnd.setMinimumSize(new Dimension(140, 20));
            toTheEnd.setMaximumSize(new Dimension(140, 20));
```

```
            toTheEnd.setPreferredSize(new Dimension(140, 20));
            toTheEnd.setSize(140, 20);
            toTheEnd.addActionListener(new ActionListener(){

                  @Override
                  public void actionPerformed(ActionEvent e) {
                        // TODO Auto-generated method stub
                        toTheEnd();
                  }});
            hb.add(toTheEnd);

            vb.add(hb);

            this.add(vb);
      }
      public void watch(Automation[] a, int[] fitness){
            history.setEnabled(true);
            history.setValue(history.getMinimum());
            desc.setText("White Ant(Fitness: "+Integer.toString(fitness[0])+")
vs Black Ant(Fitness: "+Integer.toString(fitness[1])+")");
            m = new Mover(a);
            field.drawField(m.getField(), m.getAntWhite(), m.getAntBlack());
            steps.setText("Steps: "+Integer.toString(m.getStep()));
            score.setText(Integer.toString(m.getWhiteCellsCount())+"-
"+Integer.toString(m.getBlackCellsCount()));
      }
      public void nextStep(){
            if(m == null){
                  return;
            }
            m.move();
            field.drawField(m.getField(), m.getAntWhite(), m.getAntBlack());
            steps.setText("Steps: "+Integer.toString(m.getStep()));
            score.setText(Integer.toString(m.getWhiteCellsCount())+"-
"+Integer.toString(m.getBlackCellsCount()));
      }
      public void toTheEnd(){
            if(m == null){
                  return;
            }
            m.toTheEnd();
            field.drawField(m.getField(), m.getAntWhite(), m.getAntBlack());
            steps.setText("Steps: "+Integer.toString(m.getStep()));
```

```
            score.setText(Integer.toString(m.getWhiteCellsCount())+"-
"+Integer.toString(m.getBlackCellsCount()));
      }
}
```

Файл *Field.java*

```
package ru.ifmo.rain.components;

import java.awt.Canvas;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;

import ru.ifmo.rain.phenotype.Ant;
import ru.ifmo.rain.phenotype.CellType;

public class Field extends Canvas {

      /**
       *
       */
      private static final long serialVersionUID = 1L;

      public void paint(Graphics g){
            this.setBackground(Color.GREEN);
            this.setPreferredSize(new Dimension(480,480));
            this.setMinimumSize(new Dimension(480,480));
            this.setMaximumSize(new Dimension(480,480));
            this.setSize(new Dimension(480,480));
            drawFieldSceleton();
      }
      public void drawFieldSceleton(){
            Graphics g = this.getGraphics();
            g.setColor(Color.GREEN);
            g.fillRect(0, 0, 480, 480);
            g.setColor(Color.GRAY);
            for(int i = 0; i<=32; ++i){
                  g.drawLine(i*15, 0, i*15, 480);
                  g.drawLine( 0, i*15, 480, i*15);
            }
```

```
}
public void drawField(CellType[][] field, Ant whiteAnt, Ant blackAnt){
	Graphics g = this.getGraphics();
	g.setColor(Color.GREEN);
	g.fillRect(0, 0, 480, 480);
	for(int i = 0; i < 32; ++i){
		for(int j = 0; j < 32; ++j){
			switch(field[i][j]){
			case White:
				g.setColor(Color.WHITE);
				break;
			case Black:
				g.setColor(Color.BLACK);
				break;
			default:
				g.setColor(Color.GREEN);
			}
			g.fillRect(i*15, j*15, 15, 15);
		}
	}
	g.setColor(Color.GRAY);
	for(int i = 0; i<=32; ++i){
		g.drawLine(i*15, 0, i*15, 480);
		g.drawLine( 0, i*15, 480, i*15);
	}
	drawAnt(whiteAnt,g,Color.RED);
	drawAnt(blackAnt,g,Color.RED);
}
public void drawAnt(Ant ant, Graphics g, Color c){
	Color cOld = g.getColor();
	g.setColor(c);
	g.fillOval(ant.getX()*15 + 5, ant.getY()*15+ 5, 6, 6);
	switch(ant.getDirection()){
	case Down:
		g.fillRect(ant.getX()*15 + 7, ant.getY()*15 + 2, 3, 7);
		break;
	case Up:
		g.fillRect(ant.getX()*15 + 7, ant.getY()*15 + 7, 3, 7);
		break;
	case Right:
		g.fillRect(ant.getX()*15 + 7, ant.getY()*15 + 7, 7, 3);
		break;
```

```
            case Left:
                  g.fillRect(ant.getX()*15 + 2, ant.getY()*15 + 7, 7, 3);
                  break;
            }
            g.setColor(cOld);
      }
}
```