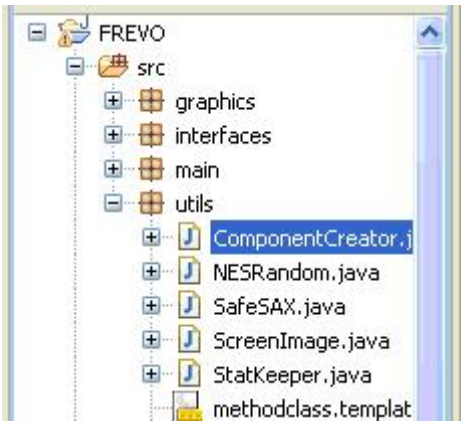
FREVO Tutorial - New Problem

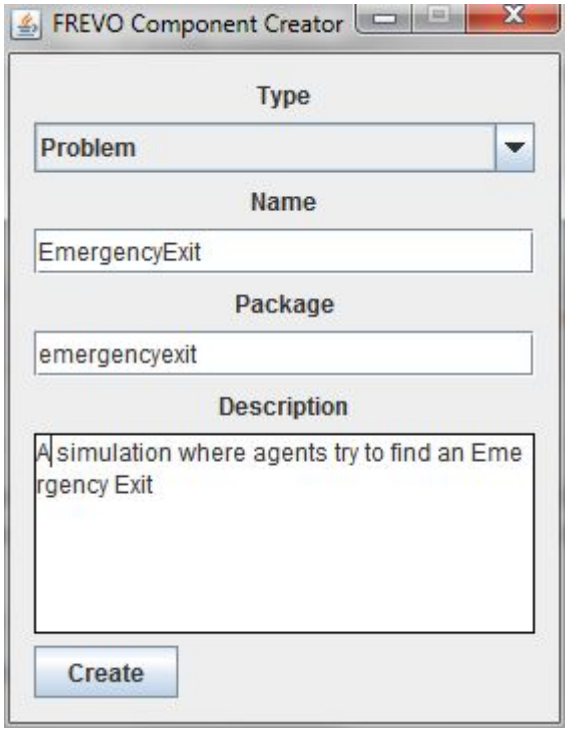
This tutorial explains how to model and implement a new problem in FREVO. FREVO (see www.frevotool.tk) is an open-source framework developed in Java to help engineers and scientists in evolutionary design or optimization tasks. The major feature of FREVO is the componentwise decomposition and separation of the key building blocks for each optimization tasks. We identify these as the problem definition, solution representation and the optimization method.

In this tutorial we will implement a simulation where agents try to find an emergency exit. Let's start:

- Run ComponentCreator.java. (ComponentCreator.java can be found, as you see in the picture, in the package utils)



- A window opens. Here you have to select the type of the component you want to create (Problem, Method, Representation, Ranking). For a simulation the right choice is Problem. Then you have to enter a name and a short description.



Click on "Create" and follow the instructions that are shown. A new folder, containing your *.java file, and an *.xml file will be generated generatet. (in this case "EmergencyExit.java" and "EmergencyExit.xml")

- The generated *.java file looks like this:

```
package emergencyexit;

import interfaces.IProblem;
import interfaces.IRepresentation;

public class EmergencyExit2 extends IProblem {

	@Override
	public double getResult(IRepresentation[] candidates) {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public void replayWithVisualization(IRepresentation[] candidates) {
		// TODO Auto-generated method stub

	}

}
```

  -
  - getResult() is called to simulate without visualization.
  - replayWithVisualization() is (as the name says) called to replay the simulation with visualization.

- Implement your simulation. (it’s useful to extract it in an own function. So you can call it from getResult() and replayWithVisualization).

For this tutorial I started with a very simple simulation:

```
  int steps;
  int xpositionofEmergencyExit = 0;
  int ypositionofEmergencyExit = 0;
  int width;
  int height;
  int xpositionofAgent;
  int ypositionofAgent;
  IRepresentation[] c;

void calcSim(){

    xpositionofEmergencyExit =
Integer.parseInt(getProperties().get("xpositionofEmergencyExit").getValue()
);
    ypositionofEmergencyExit =
Integer.parseInt(getProperties().get("ypositionofEmergencyExit").getValue()
);
    xpositionofAgent =
Integer.parseInt(getProperties().get("xpositionofAgent").getValue());
    ypositionofAgent =
Integer.parseInt(getProperties().get("ypositionofAgent").getValue());

    for (int step = 0; step < steps; step++) {
      Vector<Float> input = new Vector<Float>();
      input.add((float) (xpositionofEmergencyExit - xpositionofAgent));
      input.add((float) (ypositionofEmergencyExit - ypositionofAgent));

      Vector<Float> output = c[0].getOutput(input);

      float xVelocity = output.get(0).floatValue()*2.0f-1.0f;
      float yVelocity = output.get(1).floatValue()*2.0f-1.0f;

      if /* */(xVelocity >= 1.0 && xpositionofAgent < width - 1)
xpositionofAgent += 1;
      else if (xVelocity <= -1.0 && xpositionofAgent > 0 /*   */)
xpositionofAgent -= 1;
      if /* */(yVelocity >= 1.0 && ypositionofAgent < height - 1)
ypositionofAgent += 1;
      else if (yVelocity <= -1.0 && ypositionofAgent > 0 /*    */)
ypositionofAgent -= 1;
    }
  }
```

The position of the emergency exit and the agent are read from the *.xml file which is accessed getProperties().get(name).getValue(). Where name represents the name of the value in the *.xml file. The value of "steps", "width" and "height" are written in the functions getResult() and replayWithVisualization(). The main function of FREVO is to find the best way how to connect the input and the output. So you just have to collect all the inputs and the representation (here it is c[0]) will return the output. It’s important that all the inputs and all the outputs are always in the same order. The output is always a float value between 0.0 and 1.0. You have to decide how to handle these outputs. In this simulation the output describes how the agent moves.

- Now the code of the simulation is finished but it still has to be called by getResult().

```
public double getResult(IRepresentation[] candidates) {
    steps = Integer.parseInt(getProperties().get("steps").getValue());
```

```
    width = Integer.parseInt(getProperties().get("width").getValue());
    height = Integer.parseInt(getProperties().get("height").getValue());
    c = candidates;

    calcSim();

    return -Math.sqrt(Math.pow((xpositionofEmergencyExit -
xpositionofAgent), 2)
    /*            */+ Math.pow((ypositionofEmergencyExit -
ypositionofAgent), 2));
  }
```

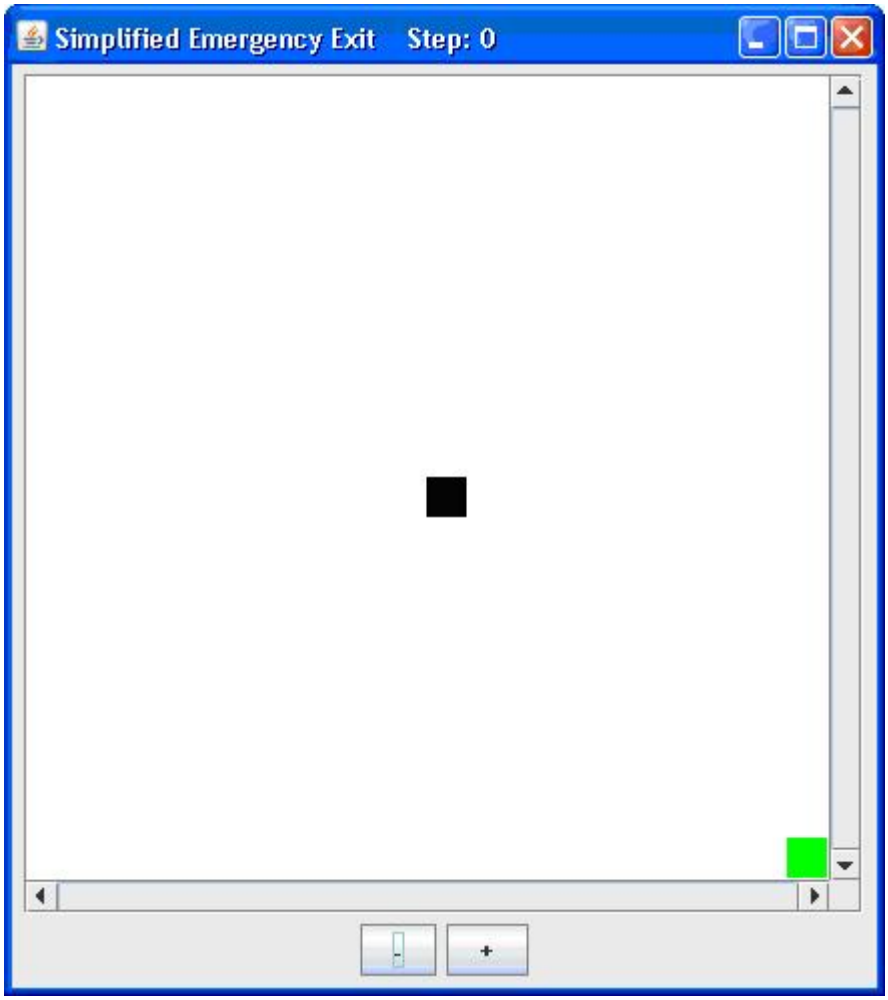
As we said before the values of “steps”, “width” and “height” have to be written in this function. They are read from the *.xml file. The return value of this function says how good this representation was. It says if this value is high, the connection of input and output is good. For the emergencyExit simulation this value is the negative distance between the agent and the emergency exit. So if the agent reaches the emergency exit within the amount of steps, the distance will be 0 and so it says it is a good way of connecting input and output.

- At the end you have to implement a visualization for your simulation:

```
  WhiteBoard whiteboard;
  Display display;

  @Override
  public void replayWithVisualization(IRepresentation[] candidates) {
    steps = 0;
    c = candidates;
    width = Integer.parseInt(getProperties().get("width").getValue());
    height = Integer.parseInt(getProperties().get("height").getValue());
    display = new Display(440, 495, "SimplifiedEmergencyExit");
    display.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    whiteboard = new WhiteBoard(400, 400, width, height, 1);
    whiteboard.addColorToScale(0, Color.WHITE);
    whiteboard.addColorToScale(1, Color.BLACK);
    whiteboard.addColorToScale(2, Color.GREEN);
    JButton minusbutton = new JButton("-");
    JButton plusbutton = new JButton("+");
    display.add(whiteboard);
    display.add(minusbutton);
    display.add(plusbutton);
    minusbutton.addActionListener(new ActionListener() {

      @Override
      public void actionPerformed(ActionEvent e) {
        if (steps > 0) steps--;
        calcSim();
        displayResult();
        display.setTitle("Simplified Emergency Exit    Step: " + steps);
      }
    });
    plusbutton.addActionListener(new ActionListener() {

      @Override
      public void actionPerformed(ActionEvent e) {
        steps++;
        calcSim();
        displayResult();
```

```
        display.setTitle("Simplified Emergency Exit    Step: " + steps);
      }
    });
    display.setVisible(true);
    calcSim();
    displayResult();
    display.setTitle("Simplified Emergency Exit    Step: " + steps);
  }

  private void displayResult() {
    int[][] data = new int[width][height];
    for (int x = 0; x < width; x++) {
      for (int y = 0; y < height; y++) {
        if /* */(x == xpositionofEmergencyExit && y ==
ypositionofEmergencyExit) data[x][y] = 2;
        else if (x == xpositionofAgent /*    */&& y == ypositionofAgent) /*
*/data[x][y] = 1;
        else /*
*/data[x][y] = 0;
      }
    }
    whiteboard.setData(data);
    whiteboard.repaint();
  }
```

Therefore the class WhiteBoard can be used. It is an extension of JPanel which represents a two- or three-dimensional grid of data in form of colors or pictures in a grid. It only has to be initialized and added to a JFrame or an extension of JFrame. Here the class Display is used. It is an extension of JFrame with a new constructor and a few settings that have already been done. The window of this visualization contains the WhiteBoard and two buttons, which are used to go through the simulation step by step. The simulation always does as much steps as the value of “steps” says. The simulation with visualization always works with the same representation and the same starting conditions. So it is possible to increase the value of “steps” and start a simulation from the beginning by clicking on plusbutton without having any differences in the agent’s behaviour. You will just see the next step of the simulation. For displaying the result the position of the agent and the emergency exit have to be converted into a two-dimensional array. Also the color-scale of the WhiteBoard has to be set. The conversion is done by displayResult(). The setting of the colorscale is done in replayWithVisualization by the function addColorToScale(int lowerLimit, Color c). All values within lowerLimit and the next lowerLimit or, if there is no next lowerLimit, the top, have the color c. As soon as this has been done you have to set the data which should be shown by the WhiteBoard. The data is set with the function setData(int[][] data). The call of repaint() will force the WhiteBoard to visualize the data which will look like this:



The black square represents the agent and the green square the emergency exit.

- A form of this simulation which has got a few more complexity is this:

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;
import java.util.Vector;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTextField;

import GridVisualization.*;

import interfaces.IProblem;
import interfaces.IRepresentation;

public class EmergencyExit extends IProblem {

  int               steps;
  int               width;
  int               height;
  int               numberofAgents;
  int               numberofExits;
  int               seed;
```

```
  agent[]           agents;
  Exit[]            EmergencyExits;
  IRepresentation[] c;
  JTextField        seedTextField;

  // this function is called to simulate without visualization. It is used
to find the best Representation
  @Override
  public double getResult(IRepresentation[] candidates) {
    // read config from xml file
    steps = Integer.parseInt(getProperties().get("steps").getValue());
    width = Integer.parseInt(getProperties().get("width").getValue());
    height = Integer.parseInt(getProperties().get("height").getValue());
    seed =
Integer.parseInt(getProperties().get("seedforEmergencyExits").getValue());
    c = candidates;
    double Fitness = 0;
    for (int s = seed; s < seed + 10; s++) {
      setupField(s);
      Fitness += calcSim();
    }

    return (Fitness / 10);
  }

  WhiteBoard whiteboard;
  Display    display;

  @Override
  public void replayWithVisualization(IRepresentation[] candidates) {
    c = candidates;
    // read config from xml file
    width = Integer.parseInt(getProperties().get("width").getValue());
    height = Integer.parseInt(getProperties().get("height").getValue());
    seed =
Integer.parseInt(getProperties().get("seedforEmergencyExits").getValue());
    display = new Display(840, 695, "EmergencyExit");
    display.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    whiteboard = new WhiteBoard(600, 600, width, height, 1);
    whiteboard.addColorToScale(0, Color.WHITE);
    // you can decide whether to take images or colors to represent things
on the whiteboard
    //whiteboard.addColorToScale(1, Color.BLACK);
    whiteboard.addImageToScale(1,
"Components\\Problems\\EmergencyExit\\agent.png");
    //whiteboard.addColorToScale(2, Color.GREEN);
    whiteboard.addImageToScale(2,
"Components\\Problems\\EmergencyExit\\EmergencyExit.png");
    JButton minusbutton = new JButton("<--");
    JButton plusbutton = new JButton("-->");
    seedTextField = new JTextField("" + seed);
    seedTextField.setPreferredSize(new Dimension(50, 20));
    JButton startButton = new JButton("Change seed");
    display.add(whiteboard);
    display.add(minusbutton);
    display.add(plusbutton);
    display.add(seedTextField);
    display.add(startButton);
    minusbutton.addActionListener(new ActionListener() {

      @Override
      public void actionPerformed(ActionEvent e) {
```

```
        if (steps > 0) steps--;
        setupField(seed);
        double Fitness = calcSim();
        displayResult();
        int agentsleft = 0;
        for (agent a : agents) {
          if (!a.hasReachedExit) agentsleft++;
        }
        String FitnessString = String.format("%.2f", Fitness);
        display.setTitle("Emergency Exit    Step: " + steps + "   Fitness:
" + FitnessString + "  Number of Agents left: " + agentsleft);
      }
    });
    plusbutton.addActionListener(new ActionListener() {

      @Override
      public void actionPerformed(ActionEvent e) {
        steps++;
        setupField(seed);
        double Fitness = calcSim();
        displayResult();
        int agentsleft = 0;
        for (agent a : agents) {
          if (!a.hasReachedExit) agentsleft++;
        }
        String FitnessString = String.format("%.4f", Fitness);
        display.setTitle("Emergency Exit    Step: " + steps + "   Fitness:
" + FitnessString + "  Number of Agents left: " + agentsleft);
      }
    });
    startButton.addActionListener(new ActionListener() {

      @Override
      public void actionPerformed(ActionEvent e) {
        steps = 0;
        seed = Integer.parseInt(seedTextField.getText());
        setupField(seed);
        double Fitness = calcSim();
        displayResult();
        int agentsleft = 0;
        for (agent a : agents) {
          if (!a.hasReachedExit) agentsleft++;
        }
        String FitnessString = String.format("%.4f", Fitness);
        display.setTitle("Emergency Exit    Step: " + steps + "   Fitness:
" + FitnessString + "  Number of Agents left: " + agentsleft);
      }
    });
    display.setVisible(true);
    setupField(seed);
    double Fitness = calcSim();
    displayResult();
    int agentsleft = 0;
    for (agent a : agents) {
      if (!a.hasReachedExit) agentsleft++;
    }
    String FitnessString = String.format("%.4f", Fitness);
    display.setTitle("Emergency Exit    Step: " + steps + "   Fitness: " +
FitnessString + "  Number of Agents left: " + agentsleft);
  }

  /**
   * Displays the result of the last Simulation
```

```
   */
  private void displayResult() {
    int[][] data = new int[width][height];
    for (int x = 0; x < width; x++) {
      for (int y = 0; y < height; y++) {
        data[x][y] = 0;
      }
    }
    for (agent a : agents) {
      data[a.xpos][a.ypos] = 1;
    }
    for (Exit e : EmergencyExits) {
      data[e.xpos][e.ypos] = 2;
    }

    whiteboard.setData(data);
    whiteboard.repaint();
  }

  void setupField(int s) {
    // read config from xml file
    numberofAgents /**/=
Integer.parseInt(getProperties().get("NumberofAgents").getValue());
    numberofExits /* */=
Integer.parseInt(getProperties().get("NumberofEmergencyExits").getValue());

    numberofAgents /**/= Math.min(numberofAgents, Math.max(width, height));
    numberofExits /* */= Math.min(numberofExits, width * height);
    agents /*        */= new agent[numberofAgents];
    EmergencyExits /**/= new Exit[numberofExits];
    // it is important to reset the Representation. Otherwise there would
sometimes be simulation mistakes because the Representation wouldn't start
from the same seed
    c[0].reset();
    // create the agents and place them in a straight line from the upper
left to the lower right corner

    for (int i = 0; i < agents.length; i++) {
      agents[i] = new agent(c[0].clone(), (i + 1) * width / (agents.length
+ 1), (i + 1) * height / (agents.length + 1), false);
    }
    Random positionGenerator = new Random(s);
    for (int i = 0; i < EmergencyExits.length; i++) {
      EmergencyExits[i] = new Exit();
      boolean PositionExistsAlready = false;
      do {
        EmergencyExits[i].xpos = positionGenerator.nextInt(width);
        EmergencyExits[i].ypos = positionGenerator.nextInt(width);
        PositionExistsAlready = false;
        for (int j = 0; j < i; j++) {
          if (EmergencyExits[i].xpos == EmergencyExits[j].xpos &&
EmergencyExits[i].ypos == EmergencyExits[j].ypos) PositionExistsAlready =
true;
        }
      } while (PositionExistsAlready);
    }
  }

  /**
   * Calculates one Simulation whit a certain amount of steps, which has to
be defined before calling this method
   *
```

```
   * @return Returns the negative Sum of the distances between the agents
and the Emergency Exit
   */
  double calcSim() {

    for (int step = 0; step < steps; step++) {
      for (int i = 0; i < agents.length; i++) {
        agent a = agents[i];
        if (!a.hasReachedExit) {

          Exit nearestExit = EmergencyExits[0];
          double minimumDistance =
Math.sqrt(Math.pow(EmergencyExits[0].xpos - a.xpos, 2) +
Math.pow(EmergencyExits[0].ypos - a.ypos, 2));
          for (int e = 0; e < EmergencyExits.length; e++) {
            double Distance = Math.sqrt(Math.pow(EmergencyExits[e].xpos -
a.xpos, 2) + Math.pow(EmergencyExits[e].ypos - a.ypos, 2));
            if (Distance < minimumDistance) {
              minimumDistance = Distance;
              nearestExit = EmergencyExits[e];
            }
          }

          // input[0] .. horizontal distance between the agent and the
nearest Emergency Exit
          // input[1] .. vertical distance between the agent and the
nearest Emergency Exit
          // input[2] .. field north      of the agent is occupied
          // input[3] .. field north-east of the agent is occupied
          // input[4] .. field east       of the agent is occupied
          // input[5] .. field south-east of the agent is occupied
          // input[6] .. field south      of the agent is occupied
          // input[7] .. field south-west of the agent is occupied
          // input[8] .. field west       of the agent is occupied
          // input[9] .. field north-west of the agent is occupied

          // determine which fields around the agent are occupied by
another agent
          boolean northoccupied /**/= false;
          boolean northeastoccupied = false;
          boolean eastoccupied /* */= false;
          boolean southeastoccupied = false;
          boolean southoccupied /**/= false;
          boolean southwestoccupied = false;
          boolean westoccupied /* */= false;
          boolean northwestoccupied = false;

          for (int j = 0; j < agents.length; j++) {
            agent ag = agents[j];
            if (!ag.hasReachedExit) { // If a agent has reached the
Emergency Exit he cannot occupy a field
              if (a.xpos /**/== ag.xpos && a.ypos - 1 == ag.ypos)
northoccupied /**/= true;
              if (a.xpos + 1 == ag.xpos && a.ypos - 1 == ag.ypos)
northeastoccupied = true;
              if (a.xpos + 1 == ag.xpos && a.ypos /**/== ag.ypos)
eastoccupied /* */= true;
              if (a.xpos + 1 == ag.xpos && a.ypos + 1 == ag.ypos)
southeastoccupied = true;
              if (a.xpos /**/== ag.xpos && a.ypos + 1 == ag.ypos)
southoccupied /**/= true;
              if (a.xpos - 1 == ag.xpos && a.ypos + 1 == ag.ypos)
southwestoccupied = true;
```

```
              if (a.xpos - 1 == ag.xpos && a.ypos /**/== ag.ypos)
westoccupied /* */= true;
              if (a.xpos - 1 == ag.xpos && a.ypos - 1 == ag.ypos)
northwestoccupied = true;
            }
          }

          Vector<Float> input = new Vector<Float>();
          input.add((float) (nearestExit.xpos - a.xpos));
          input.add((float) (nearestExit.ypos - a.ypos));
          input.add(northoccupied /**/? 1.0f : 0.0f);
          input.add(northeastoccupied ? 1.0f : 0.0f);
          input.add(eastoccupied /* */? 1.0f : 0.0f);
          input.add(southeastoccupied ? 1.0f : 0.0f);
          input.add(southoccupied /**/? 1.0f : 0.0f);
          input.add(southwestoccupied ? 1.0f : 0.0f);
          input.add(westoccupied /* */? 1.0f : 0.0f);
          input.add(northwestoccupied ? 1.0f : 0.0f);

          // output[0] .. horizontal velocity of the agent
          // output[1] .. vertical velocity of the agent
          Vector<Float> output = a.representation.getOutput(input);

          // the elements of output are float values between 0.0 and 1.0
          // for the simulation it is useful to format these values so that
you can see what each value means
          float xVfloat = output.get(0).floatValue() * 2.0f - 1.0f;
          float yVfloat = output.get(1).floatValue() * 2.0f - 1.0f;

          int xVelocity = Math.round(xVfloat); // -1 .. move one field in
negative horizontal direction
                                               //  0 .. do not move in any
horizontal direction
                                               //  1 .. move one field in
positive horizontal direction
          int yVelocity = Math.round(yVfloat); // -1 .. move one field in
negative vertical direction
                                               //  0 .. do not move in any
vertical direction
                                               //  1 .. move one field in
positive vertical direction

          // move the agent (only if the place, that he wants to move is
not occupied by another agent)
          if /*   */(xVelocity == 0/* */&& yVelocity == -
1/**/&& !northoccupied /**/&& /*                 */a.ypos > 0) {
            a.xpos += 0;
            a.ypos += -1;
          } else if (xVelocity == 1/* */&& yVelocity == -
1/**/&& !northeastoccupied && a.xpos < width - 1 && a.ypos > 0) {
            a.xpos += 1;
            a.ypos += -1;
          } else if (xVelocity == 1/* */&& yVelocity == 0/*
*/&& !eastoccupied /* */&& a.xpos < width - 1) {
            a.xpos += 1;
            a.ypos += 0;
          } else if (xVelocity == 1/* */&& yVelocity == 1/*
*/&& !southeastoccupied && a.xpos < width - 1 && a.ypos < height - 1) {
            a.xpos += 1;
            a.ypos += 1;
          } else if (xVelocity == 0/* */&& yVelocity == 1/*
*/&& !southoccupied /**/&& /*                 */a.ypos < height - 1) {
            a.xpos += 0;
```

```
            a.ypos += 1;
          } else if (xVelocity == -1/**/&& yVelocity == 1/*
*/&& !southwestoccupied && a.xpos > 0 /*    */&& a.ypos < height - 1) {
            a.xpos += -1;
            a.ypos += 1;
          } else if (xVelocity == -1/**/&& yVelocity == 0/*
*/&& !westoccupied /* */&& a.xpos > 0/*    */) {
            a.xpos += -1;
            a.ypos += 0;
          } else if (xVelocity == -1/**/&& yVelocity == -
1/**/&& !northwestoccupied && a.xpos > 0 /*    */&& a.ypos > 0) {
            a.xpos += -1;
            a.ypos += -1;
          }

          for (int n = 0; n < EmergencyExits.length && !a.hasReachedExit;
n++) {
            if (a.xpos == EmergencyExits[n].xpos && a.ypos ==
EmergencyExits[n].ypos) a.hasReachedExit = true;
            else /*
*/a.hasReachedExit = false;
          }
        }
      }
    }
    double Fitness = 0;
    for (agent a : agents) {
      double minimumDistance = Math.sqrt(Math.pow(EmergencyExits[0].xpos -
a.xpos, 2) + Math.pow(EmergencyExits[0].ypos - a.ypos, 2));
      for (int e = 0; e < EmergencyExits.length; e++) {
        double Distance = Math.sqrt(Math.pow(EmergencyExits[e].xpos -
a.xpos, 2) + Math.pow(EmergencyExits[e].ypos - a.ypos, 2));
        if (Distance < minimumDistance) {
          minimumDistance = Distance;
        }
      }
      Fitness += -minimumDistance / Math.sqrt(Math.pow(width, 2) +
Math.pow(height, 2));
    }

    return (Fitness / numberofAgents);
  }

  public class agent {
    public IRepresentation representation;
    public int             xpos;
    public int             ypos;
    public boolean         hasReachedExit;

    /**
     * @param r Representation for this Agent
     * @param x defines the x position of this Agent
     * @param y defines the y position of this Agent
     */
    public agent(IRepresentation r, int x, int y, boolean reachedExit) {
      representation = r;
      xpos = x;
      ypos = y;
      hasReachedExit = reachedExit;
    }
  }

  public class Exit {
```

```
    public int xpos;
    public int ypos;
  }
}
```

The main changes are that there are more than one agent and also more than on emergency exits. The agents are distributed over the field in a straight line from the upper left to the lower right corner of the field. The emergency exits are distributed by random with a seed. It also shows that the same connection of input and output should be able to start with different starting conditions and nevertheless to find the exit for all agents. This is done by adding a loop to getResult() which changes the starting seed of setupField().