```
 1: package genetic.algorithm;
 2:
 3: import java.util.ArrayList;
 4: import java.util.Collections;
 5:
 6: public class Generation extends ArrayList<Turtle> {
 7:
 8:         private Turtle currentTurtle;
 9:         private int index = 0;
10:
11:         public Generation(Generation parents) {
12:                 super();
13:                 if (parents == null) {
14:                         randomGen();
15:                 } else {
16:                         newGenFromParents(parents);
17:                 }
18:                 currentTurtle = this.get(0);
19:         }
20:
21:         public Turtle getCurrentTurtle() {
22:                 currentTurtle = this.get(index);
23:                 return currentTurtle;
24:         }
25:
26:         public void nextTurtle() {
27:                 index++;
28:         }
29:
30:         public boolean done() {
31:                 return index > size() - 1;
32:         }
33:
34:         public int getIndex() {
35:                 return index;
36:         }
37:
38:         private void randomGen() {
39:                 for (int i = 0; i < World.GEN_SIZE; i++) {
40:                         Turtle t = new Turtle();
41:                         t.fillLocationList();
42:                         this.add(t);
43:                 }
44:         }
45:
46:         private void newGenFromParents(Generation parents) {
47:
48:                 Collections.sort(parents);
49:
50:                 // highest path turtles are removed
51:                 parents.killTheWeak();
52:
53:                 // pair up into twos
54:                 Collections.shuffle(parents);
55:
56:                 for (int i = 0; i < (parents.size()); i++) {
57:                         Turtle mom = parents.get(i);
58:                         Turtle dad = parents.get(parents.size() - 1 - i);
59:                         makeBabies(this, mom.clone(), dad.clone(), parents);
60:                 }
61:
62:         }
63:
64:         private void killTheWeak() {
65:                 int cutoff = (int) (0.5 * size());
66:                 if (cutoff < this.size() && cutoff > 0) {
67:                         removeRange(0, cutoff);
68:                 }
69:         }
70:
71:         private static void makeBabies(Generation babies, Turtle mom, Turtle dad,
72:                         Generation parentGen) {
73:
74:                 // mutate parents
75:                 mom.mutate();
76:                 dad.mutate();
77:                 // cross over
78:                 crossOver(babies, mom, dad, parentGen);
79:                 // optimize
80:
81:                 babies.optimize();
82:         }
83:
84:         private void optimize() {
85:                 for (Turtle t : this) {
86:                         t.optimize();
87:                 }
88:         }
89:
90:         private static void crossOver(Generation babies, Turtle mother,
91:                         Turtle father, Generation parentGen) {
92:                 Turtle.crossOver(babies, mother, father, parentGen);
93:         }
94:
95:         public double getAvgFitness() {
96:                 double sum = 0;
97:                 for (Turtle t : this) {
98:                         sum += t.fitness;
99:                 }
100:                return sum / size();
101:        }
102:
103:        @Override
104:        public String toString() {
105:                String turtles = "[...";
106:                for (Turtle t : this) {
107:                        turtles += t.fitness + ", " + t.toString() + " \r\n ";
108:                }
109:                turtles += "...]";
110:                return getAvgFitness() + " :  " + turtles;
111:        }
112: }
```

```
 1: package genetic.algorithm;
 2:
 3: public class Location {
 4:
 5:         public int x, y;
 6:         public LocationList nearbyLocations;
 7:
 8:         public Location(int x, int y) {
 9:                 this.x = x;
10:                 this.y = y;
11:         }
12:
13:         public void setNearbyLocations() {
14:                 for (Location l : Master.getInstance().locations) {
15:                         if (Location.lengthBetween(l, this) < 6) {
16:                                 l.nearbyLocations.add(l);
17:                         }
18:                 }
19:         }
20:
21:         public boolean invalid() {
22:                 return x >= World.W || x < 0 || y < 0 || y >= World.H;
23:         }
24:
25:         @Override
26:         public String toString() {
27:                 return x + "," + y;
28:         }
29:
30:         public static double lengthBetween(Location l, Location last) {
31:                 double dx = last.x - l.x;
32:                 double dy = last.y - l.y;
33:                 return Math.sqrt(dx * dx + dy * dy);
34:         }
35:
36:         @Override
37:         public Location clone() {
38:                 Location newLoc = new Location(this.x, this.y);
39:                 return newLoc;
40:         }
41: }
```

```
    1: package genetic.algorithm;
    2:
    3: import java.util.ArrayList;
    4: import java.util.Collections;
    5: import java.util.Random;
    6:
    7: public class LocationList extends ArrayList<Location> {
    8:
    9:         public static double chanceOfMutation = 0.15, chanceOfOpt2 = 0.2,
   10:                         chanceOfNeighborPreference = 0.9;
   11:         private static final Random R = new Random();
   12:         private static final LocationList masterList = Master.getInstance().location
s;
   13:
   14:         public LocationList() {
   15:                 super();
   16:         }
   17:
   18:         public void shuffle() {
   19:                 Collections.shuffle(this);
   20:         }
   21:
   22:         public void fillFromMaster() {
   23:                 for (Location l : masterList) {
   24:                         add(l);
   25:                 }
   26:                 Collections.shuffle(this);
   27:         }
   28:
   29:         public void mutate() {
   30:                 // randomly flip locations
   31:                 for (int i = 0; i < size() - 1; i++) {
   32:                         if (R.nextDouble() < LocationList.chanceOfMutation) {
   33:                                 Collections.swap(this, i, i + 1);
   34:                         }
   35:                 }
   36:         }
   37:
   38:         public LocationList crossWith(LocationList mateList, int crossPoint) {
   39:                 return this;
   40:         }
   41:
   42:         public void optimize() {
   43:                 for (int i = 0; i < this.size() - 4; i++) {
   44:                         Location A = this.get(i);
   45:                         Location B = this.get(i + 1);
   46:                         Location C = this.get(i + 2);
   47:                         Location D = this.get(i + 3);
   48:
   49:                         double option1 = Location.lengthBetween(A, B)
   50:                                         + Location.lengthBetween(C, D);
   51:                         double option2 = Location.lengthBetween(A, C)
   52:                                         + Location.lengthBetween(B, D);
   53:
   54:                         // if the second choice is better, make the flip
   55:                         if (option2 < option1) {
   56:                                 this.remove(i + 1);
   57:                                 this.remove(i + 1);
   58:                                 this.add(i + 1, C);
   59:                                 this.add(i + 2, B);
   60:                         }
   61:                 }
   62:         }
   63:
   64:         public void optimize2() {
   65:                 for (int i = 0; i < this.size(); i++) {
   66:                         Location A = this.get(i);
   67:                         // find closest locations
   68:                         Location closestLoc = locationClosestTo(A);
   69:                         if (closestLoc != null) {
   70:                                 if (R.nextDouble() < LocationList.chanceOfOpt2) {
   71:                                         Collections.swap(this, i, this.indexOf(close
stLoc));
   72:                                 }
   73:                         }
   74:                 }
   75:         }
   76:
   77:         private Location locationClosestTo(Location A) {
   78:                 Location B = null;
   79:                 double min = Double.MAX_VALUE;
   80:                 for (Location loc : this) {
   81:                         double len = Location.lengthBetween(A, loc);
   82:                         if (loc.x != A.x || loc.y != A.y) {
   83:                                 if (len < min) {
   84:                                         min = len;
   85:                                         B = loc;
   86:                                 }
   87:                         }
   88:                 }
   89:                 return B;
   90:         }
   91:
   92:         @Override
   93:         public String toString() {
   94:                 String str = "";
   95:                 for (Location l : this) {
   96:                         str += "[" + l.toString() + "] ";
   97:                 }
   98:                 return str;
   99:         }
  100:
  101:         public LocationList sub(LocationList list, int start, int end) {
  102:                 LocationList sub = new LocationList();
  103:                 for (int i = start; i < end; i++) {
  104:                         sub.add(list.get(i));
  105:                 }
  106:                 return sub;
  107:         }
  108:
  109:         @Override
  110:         public LocationList clone() {
  111:                 LocationList newList = new LocationList();
  112:                 for (Location l : this) {
  113:                         newList.add(l.clone());
  114:                 }
  115:                 return newList;
  116:         }
  117: }
```

```
 1: package genetic.algorithm;
 2:
 3: import gui.ImageButton;
 4: import gui.ResourceLoader;
 5: import gui.WelcomeFrame;
 6: import gui.WelcomePanel;
 7:
 8: import javax.imageio.ImageIO;
 9: import javax.swing.JOptionPane;
10: import javax.swing.SwingUtilities;
11:
12: public class Main {
13:
14:         public static void main(String[] args) {
15:
16:                 loadPics();
17:
18:                 SwingUtilities.invokeLater(new Runnable() {
19:                         public void run() {
20:                                 new WelcomeFrame();
21:                         }
22:                 });
23:         }
24:
25:         public static void loadPics() {
26:                 try {
27:                         // images
28:                         ImageButton.exitImg = ImageIO.read(ResourceLoader
29:                                         .loadImage("exit.png"));
30:                         ImageButton.exitImgPressed = ImageIO.read(ResourceLoader
31:                                         .loadImage("exitPressed.png"));
32:                         ImageButton.settingsImg = ImageIO.read(ResourceLoader
33:                                         .loadImage("settings.png"));
34:                         ImageButton.settingsImgPressed = ImageIO.read(ResourceLoader
35:                                         .loadImage("settingsPressed.png"));
36:                         WelcomePanel.welcomeBackground = ImageIO.read(ResourceLoader
37:                                         .loadImage("background.jpg"));
38:                         ImageButton.buttonImg = ImageIO.read(ResourceLoader
39:                                         .loadImage("buttonBackground.png"));
40:                         ImageButton.buttonImgPressed = ImageIO.read(ResourceLoader
41:                                         .loadImage("buttonBackground.png"));
42:
43:                 } catch (Exception e) {
44:                         JOptionPane.showMessageDialog(null,
45:                                         e.getMessage() + " " + e.getCause());
46:                 }
47:         }
48:
49: }
```

```
    1: package genetic.algorithm;
    2:
    3: import gui.ResourceLoader;
    4:
    5: import java.io.BufferedReader;
    6: import java.io.File;
    7: import java.io.FileReader;
    8: import java.net.URI;
    9: import java.util.ArrayList;
   10:
   11: import javax.swing.JOptionPane;
   12:
   13: public class Master {
   14:
   15:         private static Master master = new Master();
   16:
   17:         public static Master getInstance() {
   18:                 return Master.master;
   19:         }
   20:
   21:         public LocationList locations;
   22:
   23:         private Master() {
   24:                 locations = new LocationList();
   25:                 loadFromFile();
   26:         }
   27:
   28:         private void loadFromFile() {
   29:                 URI uri;
   30:                 try {
   31:                         uri = ResourceLoader.loadData(ResourceLoader.MASTER_FILENAME
)
   32:                                         .toURI();
   33:                         File masterFile = new File(uri);
   34:                         BufferedReader bfr = new BufferedReader(new FileReader(maste
rFile));
   35:                         String line = "";
   36:                         int y = 0;
   37:                         while ((line = bfr.readLine()) != null) {
   38:                                 for (int x = 0; x < World.W && x < line.length(); x+
+) {
   39:                                         if (line.charAt(x) == 'X') {
   40:                                                 Location l = new Location(x, y);
   41:                                                 locations.add(l);
   42:                                         }
   43:                                 }
   44:                                 y++;
   45:                         }
   46:                 } catch (Exception e) {
   47:                         JOptionPane.showMessageDialog(null, "ERROR " + e.getMessage(
));
   48:                         // for (StackTraceElement ste : e.getStackTrace()) {
   49:                         // System.out.println(ste.toString());
   50:                         // }
   51:                 }
   52:         }
   53: }
```

```
 1: package genetic.algorithm;
 2:
 3: public class Turtle implements Comparable<Turtle> {
 4:
 5:         public LocationList locations;
 6:         public double fitness = 0;
 7:         private int index = 0;
 8:
 9:         public Turtle() {
10:                 locations = new LocationList();
11:         }
12:
13:         public Location getNextLocation() {
14:                 return locations.get(index + 1);
15:         }
16:
17:         public Location getCurrentLocation() {
18:                 return locations.get(index);
19:         }
20:
21:         public int getIndex() {
22:                 return index;
23:         }
24:
25:         public void nextPath() {
26:                 index++;
27:         }
28:
29:         public void fillLocationList() {
30:                 locations.fillFromMaster();
31:         }
32:
33:         public void calculateFitness() {
34:                 fitness += Location.lengthBetween(getCurrentLocation(),
35:                                 getNextLocation());
36:                 nextPath();
37:         }
38:
39:         public boolean done() {
40:                 return index > locations.size() - 2;
41:         }
42:
43:         public void mutate() {
44:                 locations.mutate();
45:         }
46:
47:         public static void crossOver(Generation babies, Turtle mom, Turtle dad,
48:                         Generation parents) {
49:
50:                 Turtle kid1 = new Turtle();
51:                 Turtle kid2 = new Turtle();
52:
53:                 // split at random point
54:                 int i = mom.locations.size() / 2;
55:
56:                 kid1.locations = mom.locations;// .crossWith(this.locations, i);
57:                 kid2.locations = dad.locations;// .crossWith(mate.locations, i);
58:
59:                 babies.add(kid1);
60:                 babies.add(kid2);
61:         }
62:
63:         public void optimize() {
64:                 locations.optimize();
65:                 locations.optimize2();
66:         }
67:
68:         @Override
69:         public int compareTo(Turtle t) {
70:                 if (this.fitness < t.fitness) {
71:                         return 1;
72:                 } else if (t.fitness < this.fitness) {
73:                         return -1;
74:                 } else {
75:                         return 0;
76:                 }
77:         }
78:
79:         @Override
80:         public Turtle clone() {
81:                 Turtle t = new Turtle();
82:                 t.locations = this.locations.clone();
83:                 t.fitness = this.fitness;
84:                 return t;
85:
86:         }
87:
88:         @Override
89:         public String toString() {
90:                 return fitness + " " + this.locations.toString();
91:         }
92: }
```

```
 1: package genetic.algorithm;
 2:
 3: import java.util.ArrayList;
 4:
 5: import javax.swing.SwingUtilities;
 6:
 7: public class World {
 8:
 9:         public static int W = 20, H = 20, GEN_SIZE = 500, GEN_LIM = 50;
10:
11:         private static final World theWorld = new World();
12:
13:         public static World getInstance() {
14:                 return theWorld;
15:         }
16:
17:         private ArrayList<Generation> generations;
18:         private Generation currentGen;
19:         protected int index;
20:
21:         private World() {
22:                 generations = new ArrayList<Generation>();
23:                 generations.add(new Generation(null));
24:                 currentGen = generations.get(0);
25:                 index = 0;
26:         }
27:
28:         public void reset() {
29:                 generations.clear();
30:                 generations.add(new Generation(null));
31:                 currentGen = generations.get(0);
32:                 index = 0;
33:         }
34:
35:         public Generation getCurrentGen() {
36:                 currentGen = generations.get(index);
37:                 return this.currentGen;
38:         }
39:
40:         public int getIndex() {
41:                 return index;
42:         }
43:
44:         public int getSize() {
45:                 return generations.size();
46:         }
47:
48:         public void nextGen() {
49:                 // add new gen
50:                 Generation newGen = new Generation(getCurrentGen());
51:                 generations.add(newGen);
52:                 index++;
53:
54:         }
55: }
```

```
    1: package gui;
    2:
    3: import java.awt.event.MouseEvent;
    4: import java.awt.event.MouseListener;
    5:
    6: import javax.swing.JFrame;
    7:
    8: public class ClickableField extends Field implements MouseListener {
    9:
   10:         private static final ClickableField theClickableField = new ClickableField()
;
   11:
   12:         public static ClickableField getInstance() {
   13:                 return theClickableField;
   14:         }
   15:
   16:         private ClickableField() {
   17:                 super(CreateMasterFrame.getInstance());
   18:                 this.addMouseListener(this);
   19:         }
   20:
   21:         @Override
   22:         public void mouseClicked(MouseEvent me) {
   23:                 int nearestX = Math.round(me.getX() / Pixel.S);
   24:                 int nearestY = Math.round(me.getY() / Pixel.S);
   25:                 Pixel p = Field.pixels[nearestY][nearestX];
   26:                 p.on = !p.on;
   27:                 super.repaint();
   28:         }
   29:
   30:         @Override
   31:         public void mouseEntered(MouseEvent me) {
   32:         }
   33:
   34:         @Override
   35:         public void mouseExited(MouseEvent me) {
   36:         }
   37:
   38:         @Override
   39:         public void mousePressed(MouseEvent arg0) {
   40:                 // TODO Auto-generated method stub
   41:
   42:         }
   43:
   44:         @Override
   45:         public void mouseReleased(MouseEvent arg0) {
   46:                 // TODO Auto-generated method stub
   47:
   48:         }
   49: }
```

```java
package gui;

import genetic.algorithm.LocationList;
import genetic.algorithm.World;

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.util.Hashtable;

import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.SwingUtilities;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class ControlPanel extends JPanel implements ActionListener,
                ChangeListener {

        int w = 100, h = 40;
        protected World world = World.getInstance();
        protected static ImageButton reset, pause, play;
        protected static JLabel turtleNumber, genNumber, mutationLabel,
                        bestTurtleFitness;
        protected static JCheckBox continuous, visualize;
        protected static JSlider mutationRate;
        WorldFrame parentWorldFrame;
        protected static BufferedImage resetPic, pausePic, playPic,
                        resetPicPressed, pausePicPressed, playPicPressed;

        public static boolean pauseBetweenTrials;

        public ControlPanel(WorldFrame parentWorldFrame) {
                super();
                this.parentWorldFrame = parentWorldFrame;

                reset = new ImageButton(w, h, resetPic, resetPicPressed, "Reset");
                pause = new ImageButton(w, h, pausePic, pausePicPressed, "Pause");
                play = new ImageButton(w, h, playPic, playPicPressed, "Play");

                bestTurtleFitness = new JLabel("Shortest Path: ");
                turtleNumber = new JLabel("turtle: 0");
                genNumber = new JLabel("gen: 0");
                mutationLabel = new JLabel("mut: 0");

                turtleNumber.setPreferredSize(new Dimension(60, 30));
                genNumber.setPreferredSize(new Dimension(60, 30));

                mutationRate = new JSlider(0, 100, 5);
                mutationRate.setValue(20);

                continuous = new JCheckBox();
                continuous.setSelected(true);
                visualize = new JCheckBox();

                this.add(bestTurtleFitness);
                this.add(reset);
                this.add(mutationRate);
                this.add(mutationLabel);
                this.add(continuous);
                this.add(pause);
                this.add(play);
                this.add(turtleNumber);
                this.add(genNumber);

                reset.addActionListener(this);
                pause.addActionListener(this);
                play.addActionListener(this);
                mutationRate.addChangeListener(this);

                int width = Math.max(World.W * Pixel.S, 1100);
                this.setPreferredSize(new Dimension(width, 60));
        }

        @Override
        public void actionPerformed(ActionEvent ae) {
                if (ae.getSource() == play) {
                        play();
                        TurtlePlayer.getInstance().execute();
                } else if (ae.getSource() == pause) {
                        pause();
                        TurtlePlayer.getInstance().cancel(true);
                } else if (ae.getSource() == reset) {
                        pause();
                        World.getInstance().reset();
                }
        }

        @Override
        public void stateChanged(ChangeEvent ce) {
                JSlider source = (JSlider) ce.getSource();
                if (source.equals(mutationRate)) {
                        double mut = source.getValue() / 100.0;
                        LocationList.chanceOfMutation = mut;
                        mutationLabel.setText("mut: " + mut);
                }
        }

        public static void play() {
                play.setEnabled(false);
                pause.setEnabled(true);

        }

        public static void pause() {
                pause.setEnabled(false);
                play.setEnabled(true);
        }
}
```

```
    1: package gui;
    2:
    3: import genetic.algorithm.World;
    4:
    5: import java.awt.Color;
    6: import java.awt.Dimension;
    7: import java.awt.event.ActionEvent;
    8: import java.awt.event.ActionListener;
    9: import java.awt.event.ItemEvent;
   10: import java.awt.event.ItemListener;
   11: import java.awt.image.BufferedImage;
   12:
   13: import javax.swing.JCheckBox;
   14: import javax.swing.JPanel;
   15: import javax.swing.JSlider;
   16: import javax.swing.event.ChangeEvent;
   17: import javax.swing.event.ChangeListener;
   18:
   19: public class CreateMasterControlPanel extends JPanel implements ActionListener,
   20:                 ItemListener, ChangeListener {
   21:
   22:         int w = 100, h = 40;
   23:         ImageButton save;
   24:         CreateMasterFrame parentFrame;
   25:         public static BufferedImage resetPic, nextPic, pausePic, playPic,
   26:                         resetPicPressed, nextPicPressed, pausePicPressed, playPicPre
ssed;
   27:
   28:         public static boolean pauseBetweenTrials;
   29:
   30:         public CreateMasterControlPanel(CreateMasterFrame parentFrame) {
   31:                 super();
   32:                 this.parentFrame = parentFrame;
   33:                 save = new ImageButton(w, h, resetPic, resetPicPressed, "Save");
   34:                 this.add(save);
   35:                 save.addActionListener(this);
   36:                 this.setPreferredSize(new Dimension((World.W + 2) * Pixel.S, 60));
   37:         }
   38:
   39:         @Override
   40:         public void actionPerformed(ActionEvent ae) {
   41:                 if (ae.getSource() == save) {
   42:                         // save field
   43:                         ClickableField.getInstance().save();
   44:                 }
   45:         }
   46:
   47:         @Override
   48:         public void stateChanged(ChangeEvent ce) {
   49:         }
   50:
   51:         @Override
   52:         public void itemStateChanged(ItemEvent ie) {
   53:         }
   54: }
```

```
    1: package gui;
    2:
    3: import java.awt.BorderLayout;
    4: import java.awt.Dimension;
    5: import java.awt.Toolkit;
    6:
    7: import javax.swing.JFrame;
    8:
    9: public class CreateMasterFrame extends JFrame {
   10:
   11:         private static final CreateMasterFrame theCreateMasterFrame = new CreateMast
erFrame();
   12:
   13:         public static CreateMasterFrame getInstance() {
   14:                 return theCreateMasterFrame;
   15:         }
   16:
   17:         private Dimension screenSize;
   18:         private CreateMasterControlPanel controlPanel;
   19:         private Menu menu;
   20:
   21:         private CreateMasterFrame() {
   22:                 super();
   23:                 this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   24:                 this.setTitle("Create Master");
   25:                 this.setResizable(false);
   26:                 screenSize = Toolkit.getDefaultToolkit().getScreenSize();
   27:                 this.setLocation((screenSize.width - WelcomePanel.w) / 2,
   28:                                 (screenSize.height - WelcomePanel.h) / 2);
   29:
   30:                 menu = new Menu(this);
   31:                 controlPanel = new CreateMasterControlPanel(this);
   32:
   33:                 this.add(controlPanel, BorderLayout.SOUTH);
   34:                 this.add(ClickableField.getInstance(), BorderLayout.CENTER);
   35:                 this.add(menu, BorderLayout.NORTH);
   36:
   37:                 this.pack();
   38:         }
   39: }
```

```
   1: package gui;
   2:
   3: import genetic.algorithm.Location;
   4: import genetic.algorithm.Turtle;
   5: import genetic.algorithm.World;
   6:
   7: import java.awt.Color;
   8: import java.awt.Dimension;
   9: import java.awt.Graphics;
  10: import java.io.File;
  11: import java.io.PrintStream;
  12: import java.net.URI;
  13: import java.util.ArrayList;
  14:
  15: import javax.swing.JFrame;
  16: import javax.swing.JOptionPane;
  17: import javax.swing.JPanel;
  18:
  19: public abstract class Field extends JPanel {
  20:
  21:         public static final Color turtlePathColor = new Color(30, 30, 200, 140),
  22:                         turtleNoPathColor = new Color(255, 255, 255, 100),
  23:                         masterPathColor = new Color(255, 0, 0, 70);
  24:         public static Pixel[][] pixels = new Pixel[World.H][World.W];
  25:
  26:         JFrame parentFrame;
  27:
  28:         public Field(JFrame parentFrame) {
  29:                 super();
  30:                 this.parentFrame = parentFrame;
  31:                 // init pixels matrix
  32:                 for (int x = 0; x < World.W; x++) {
  33:                         for (int y = 0; y < World.H; y++) {
  34:                                 pixels[y][x] = new Pixel(x, y);
  35:
  36:                         }
  37:                 }
  38:                 Dimension screenSize = new Dimension(Pixel.S * World.W, Pixel.S
  39:                                 * World.H);
  40:                 this.setPreferredSize(screenSize);
  41:         }
  42:
  43:         @Override
  44:         public void paintComponent(Graphics g) {
  45:                 super.paintComponent(g);
  46:                 draw(g);
  47:         }
  48:
  49:         public void draw(Graphics g) {
  50:                 g.setColor(Color.black);
  51:                 g.fillRect(0, 0, World.W * Pixel.S, World.H * Pixel.S);
  52:
  53:                 for (Pixel[] row : pixels) {
  54:                         for (Pixel p : row) {
  55:                                 // draw current turtle path if on
  56:                                 if (p.on) {
  57:                                         g.setColor(Field.turtlePathColor);
  58:                                         g.fillRect(p.x * Pixel.S, p.y * Pixel.S, Pix
el.S, Pixel.S);
  59:                                 }
  60:                                 // draw outline unless square is tiny
  61:                                 if (Pixel.S > 8) {
  62:                                         g.setColor(Color.gray);
  63:                                         g.drawRect(p.x * Pixel.S, p.y * Pixel.S, Pix
el.S, Pixel.S);
  64:                                 }
  65:                         }
  66:                 }
  67:
  68:         }
  69:
  70:         public void clear() {
  71:                 for (Pixel[] pixelRow : pixels) {
  72:                         for (Pixel p : pixelRow) {
  73:                                 p.on = false;
  74:                                 p.visited = false;
  75:                                 p.used = false;
  76:                         }
  77:                 }
  78:         }
  79:
  80:         public void save() {
  81:                 try {
  82:                         URI uri = ResourceLoader.loadData(ResourceLoader.MASTER_FILE
NAME)
  83:                                         .toURI();
  84:                         PrintStream ps = new PrintStream(new File(uri));
  85:                         for (Pixel[] pixelRow : pixels) {
  86:                                 for (Pixel p : pixelRow) {
  87:                                         if (p.on) {
  88:                                                 ps.print('X');
  89:                                         } else {
  90:                                                 ps.print('O');
  91:                                         }
  92:                                 }
  93:                                 ps.println();
  94:                         }
  95:                         ps.flush();
  96:                         ps.close();
  97:                 } catch (Exception e) {
  98:                         JOptionPane.showMessageDialog(null,
  99:                                         getClass().getName() + " " + e.getMessage())
;
 100:                 }
 101:         }
 102: }
```

```
    1: package gui;
    2:
    3: import java.awt.Color;
    4: import java.awt.Dimension;
    5: import java.awt.Font;
    6: import java.awt.Graphics;
    7: import java.awt.image.BufferedImage;
    8:
    9: import javax.swing.BorderFactory;
   10: import javax.swing.JButton;
   11: import javax.swing.SwingConstants;
   12:
   13: public class ImageButton extends JButton {
   14:
   15:         private static final long serialVersionUID = 6316886668679040393L;
   16:         public int w, h;
   17:         public static BufferedImage buttonImg, buttonImgPressed, settingsImg,
   18:                         settingsImgPressed, exitImg, exitImgPressed;
   19:         public BufferedImage img, imgPressed;
   20:         Font buttonFont;
   21:
   22:         ImageButton(int w, int h, BufferedImage img, BufferedImage imgPressed,
   23:                         String label) {
   24:                 super(label);
   25:                 this.w = w;
   26:                 this.h = h;
   27:                 if (img == null || imgPressed == null) {
   28:                         this.img = ImageButton.buttonImg;
   29:                         this.imgPressed = ImageButton.buttonImgPressed;
   30:                 } else {
   31:                         this.img = img;
   32:                         this.imgPressed = imgPressed;
   33:                 }
   34:                 this.setOpaque(false);
   35:                 this.setContentAreaFilled(false);
   36:                 this.setBorderPainted(false);
   37:                 buttonFont = new Font("", Font.BOLD, 20);
   38:                 this.setPreferredSize(new Dimension(w, h));
   39:                 this.setHorizontalTextPosition(SwingConstants.CENTER);
   40:                 this.setVerticalTextPosition(SwingConstants.CENTER);
   41:                 this.setForeground(Color.black);
   42:                 this.setBorder(BorderFactory.createMatteBorder(5, 5, 5, 5, Color.bla
ck));
   43:                 this.setFont(buttonFont);
   44:         }
   45:
   46:         @Override
   47:         protected void paintComponent(Graphics g) {
   48:                 if (getModel().isPressed()) {
   49:                         g.drawImage(imgPressed, 0, 0, w, h, this);
   50:                 } else {
   51:                         g.drawImage(img, 0, 0, w, h, this);
   52:                 }
   53:                 super.paintComponent(g);
   54:         }
   55:
   56: }
```

```
 1: package gui;
 2:
 3: import genetic.algorithm.Location;
 4: import genetic.algorithm.Master;
 5: import genetic.algorithm.World;
 6:
 7: import java.awt.Color;
 8: import java.awt.Graphics;
 9:
10: import javax.swing.JFrame;
11:
12: public class MasterField extends Field {
13:
14:         private static final MasterField theMasterField = new MasterField();
15:
16:         public static MasterField getInstance() {
17:                 return theMasterField;
18:         }
19:
20:         private MasterField() {
21:                 super(null);
22:         }
23:
24:         public void setOwnedByWorldFrame() {
25:                 super.parentFrame = WorldFrame.getInstance();
26:                 this.repaint();
27:         }
28:
29:         @Override
30:         public void draw(Graphics g) {
31:                 g.setColor(Color.black);
32:                 g.fillRect(0, 0, World.W * Pixel.S, World.H * Pixel.S);
33:                 for (Location l : Master.getInstance().locations) {
34:                         g.setColor(Color.white);
35:                         g.drawRect(Pixel.S * l.x, Pixel.S * l.y, Pixel.S, Pixel.S);
36:                 }
37:         }
38: }
```

```
 1: package gui;
 2:
 3: import java.awt.event.ActionEvent;
 4: import java.awt.event.ActionListener;
 5:
 6: import javax.swing.JFrame;
 7: import javax.swing.JMenu;
 8: import javax.swing.JMenuBar;
 9: import javax.swing.JMenuItem;
10:
11: public class Menu extends JMenuBar implements ActionListener {
12:
13:         JMenu file;
14:         JMenuItem returnToMain;
15:         JFrame parent;
16:
17:         Menu(JFrame parent) {
18:                 super();
19:                 this.parent = parent;
20:                 file = new JMenu("File");
21:                 returnToMain = new JMenuItem("return the main screen");
22:                 file.add(returnToMain);
23:                 this.add(file);
24:                 returnToMain.addActionListener(this);
25:         }
26:
27:         @Override
28:         public void actionPerformed(ActionEvent e) {
29:                 if (e.getSource() == returnToMain) {
30:                         parent.dispose();
31:                         new WelcomeFrame();
32:                 }
33:         }
34: }
```

```
    1: package gui;
    2:
    3: import genetic.algorithm.Location;
    4:
    5: import java.awt.Color;
    6: import java.awt.Graphics;
    7:
    8: public class Path {
    9:
   10:         Location from, to;
   11:         double length;
   12:
   13:         Path(Location from, Location to) {
   14:                 this.from = from;
   15:                 this.to = to;
   16:                 length = Location.lengthBetween(from, to);
   17:         }
   18:
   19:         public void draw(Graphics g) {
   20:                 g.setColor(Color.blue);
   21:                 g.drawLine(Pixel.S * from.x + Pixel.S / 2, Pixel.S * from.y + Pixel.
S
   22:                                 / 2, Pixel.S * to.x + Pixel.S / 2, Pixel.S * to.y +
Pixel.S / 2);
   23:                 g.setColor(Color.gray);
   24:                 g.fillRect(Pixel.S * from.x, Pixel.S * from.y, Pixel.S, Pixel.S);
   25:                 g.setColor(Color.red);
   26:                 g.fillRect(Pixel.S * to.x, Pixel.S * to.y, Pixel.S, Pixel.S);
   27:         }
   28:
   29:         @Override
   30:         public String toString() {
   31:                 return from.x + "," + from.y + " -> " + to.x + "," + to.y;
   32:         }
   33:
   34: }
```

```
 1: package gui;
 2:
 3:
 4: public class Pixel {
 5:
 6:         public static final int S = 20;
 7:         public boolean on = false, visited = false, used = false;
 8:         public int x, y;
 9:
10:         // x ranges from 0 to World.W
11:         // y ranges from 0 to World.H
12:
13:         public Pixel(int x, int y) {
14:                 this.x = x;
15:                 this.y = y;
16:                 on = false;
17:         }
18: }
```

```
    1: package gui;
    2:
    3: import java.io.FileWriter;
    4: import java.net.URL;
    5:
    6: import javax.swing.JOptionPane;
    7:
    8: final public class ResourceLoader {
    9:
   10:         public static final String MASTER_FILENAME = "master.txt";
   11:
   12:         public static URL loadImage(String filePath) {
   13:                 try {
   14:                         return ResourceLoader.class.getResource("/images/" + filePat
h);
   15:                 } catch (Exception e) {
   16:                         JOptionPane.showMessageDialog(null,
   17:                                         "Resource loader error " + e.getMessage());
   18:                         return null;
   19:                 }
   20:         }
   21:
   22:         public static URL loadData(String filePath) {
   23:                 try {
   24:                         return ResourceLoader.class.getResource("/data/" + filePath)
;
   25:                 } catch (Exception e) {
   26:                         JOptionPane.showMessageDialog(null,
   27:                                         "Resource loader error " + e.getMessage());
   28:                         return null;
   29:                 }
   30:         }
   31:
   32:         public static void writeToLog(String s) {
   33:                 try {
   34:                         FileWriter fw = new FileWriter("ELTRUT_2_LOG.txt");
   35:                         fw.append(s + "\r\n");
   36:                         fw.flush();
   37:                         fw.close();
   38:                 } catch (Exception e) {
   39:                         JOptionPane.showMessageDialog(null,
   40:                                         e.getMessage() + " " + e.getCause());
   41:                 }
   42:         }
   43: }
```

```
  1: package gui;
  2:
  3: import genetic.algorithm.LocationList;
  4: import genetic.algorithm.World;
  5:
  6: import java.awt.BorderLayout;
  7: import java.awt.Dimension;
  8: import java.awt.GridLayout;
  9: import java.awt.Toolkit;
 10: import java.awt.event.ActionEvent;
 11: import java.awt.event.ActionListener;
 12:
 13: import javax.swing.JButton;
 14: import javax.swing.JDialog;
 15: import javax.swing.JLabel;
 16: import javax.swing.JOptionPane;
 17: import javax.swing.JPanel;
 18: import javax.swing.JTextField;
 19:
 20: public class SettingsDialog extends JDialog implements ActionListener {
 21:
 22:         JPanel p;
 23:         JLabel genSizeLabel, genLimLabel, gridWLabel, gridHLabel, opt1RateLabel,
 24:                         opt2RateLabel;
 25:         JTextField genSize, genLim, gridW, gridH, opt1Rate, opt2Rate;
 26:         JButton save;
 27:
 28:         SettingsDialog() {
 29:                 super();
 30:
 31:                 genSizeLabel = new JLabel("Generation Size");
 32:                 genLimLabel = new JLabel("Generation Lim");
 33:                 gridWLabel = new JLabel("Grid Width");
 34:                 gridHLabel = new JLabel("Grid Height");
 35:                 opt1RateLabel = new JLabel("Rate of Linear Optimization");
 36:                 opt2RateLabel = new JLabel("Initial Population Greedyness");
 37:
 38:                 genSize = new JTextField("500");
 39:                 genLim = new JTextField("500");
 40:                 gridW = new JTextField("20");
 41:                 gridH = new JTextField("20");
 42:                 opt1Rate = new JTextField("0.6");
 43:                 opt2Rate = new JTextField("0.2");
 44:
 45:                 save = new JButton("Save");
 46:                 save.addActionListener(this);
 47:
 48:                 p = new JPanel();
 49:                 p.setLayout(new BorderLayout());
 50:
 51:                 JPanel subPanel = new JPanel();
 52:                 subPanel.setLayout(new GridLayout(6, 2));
 53:                 subPanel.add(genSizeLabel);
 54:                 subPanel.add(genSize);
 55:                 subPanel.add(genLimLabel);
 56:                 subPanel.add(genLim);
 57:                 subPanel.add(gridWLabel);
 58:                 subPanel.add(gridW);
 59:                 subPanel.add(gridHLabel);
 60:                 subPanel.add(gridH);
 61:                 subPanel.add(opt1RateLabel);
 62:                 subPanel.add(opt1Rate);
 63:                 subPanel.add(opt2RateLabel);
 64:                 subPanel.add(opt2Rate);
 65:
 66:                 p.add(subPanel, BorderLayout.CENTER);
 67:                 p.add(save, BorderLayout.SOUTH);
 68:                 p.setPreferredSize(new Dimension(400, 200));
 69:
 70:                 this.setResizable(false);
 71:                 Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
 72:                 this.setLocation((screenSize.width - 200) / 2,
 73:                                 (screenSize.height - 100) / 2);
 74:                 // this.setUndecorated(true);
 75:                 this.add(p);
 76:                 this.pack();
 77:                 this.setVisible(true);
 78:         }
 79:
 80:         @Override
 81:         public void actionPerformed(ActionEvent e) {
 82:                 if (e.getSource() == save) {
 83:                         try {
 84:                                 World.GEN_LIM = Integer.parseInt(genLim.getText());
 85:                                 World.GEN_SIZE = Integer.parseInt(genSize.getText())
;
 86:                                 World.W = Integer.parseInt(gridW.getText());
 87:                                 World.H = Integer.parseInt(gridH.getText());
 88:                                 LocationList.chanceOfOpt2 = Double
 89:                                                 .parseDouble(opt1Rate.getText());
 90:                                 LocationList.chanceOfNeighborPreference = Double
 91:                                                 .parseDouble(opt2Rate.getText());
 92:                         } catch (NumberFormatException ex) {
 93:                                 JOptionPane.showMessageDialog(null, "You must enter
integers!");
 94:                         }
 95:                 }
 96:         }
 97: }
```

```
 1: package gui;
 2:
 3: import genetic.algorithm.Location;
 4: import genetic.algorithm.Turtle;
 5:
 6: import java.awt.Graphics;
 7: import java.util.ArrayList;
 8:
 9: import javax.swing.SwingUtilities;
10:
11: public class TurtleField extends Field {
12:
13:         private static final TurtleField theField = new TurtleField();
14:         private double pathLength;
15:
16:         public static TurtleField getInstance() {
17:                 return theField;
18:         }
19:
20:         private ArrayList<Path> paths = new ArrayList<Path>();
21:
22:         private TurtleField() {
23:                 super(WorldFrame.getInstance());
24:
25:         }
26:
27:         @Override
28:         public void paintComponent(Graphics g) {
29:                 super.paintComponent(g);
30:                 // draw paths
31:                 for (Path p : paths) {
32:                         p.draw(g);
33:                 }
34:
35:                 paths.clear();
36:         }
37:
38:         @Override
39:         public void clear() {
40:                 super.clear();
41:                 pathLength = 0;
42:                 paths.clear();
43:         }
44:
45:         public void drawTurtle(Turtle bestTurtle) {
46:                 // add paths
47:                 for (int i = 0; i < bestTurtle.locations.size() - 1; i++) {
48:                         Location l1 = bestTurtle.locations.get(i);
49:                         Location l2 = bestTurtle.locations.get(i + 1);
50:                         Path p = new Path(l1, l2);
51:                         paths.add(p);
52:                 }
53:                 this.repaint();
54:         }
55:
56: }
```

```
 1: package gui;
 2:
 3: import genetic.algorithm.Turtle;
 4: import genetic.algorithm.World;
 5:
 6: import java.awt.event.ActionEvent;
 7: import java.awt.event.ActionListener;
 8: import java.util.ArrayList;
 9: import java.util.Collections;
10: import java.util.List;
11:
12: import javax.swing.JOptionPane;
13: import javax.swing.SwingWorker;
14: import javax.swing.Timer;
15:
16: public class TurtlePlayer extends SwingWorker<Turtle, Turtle> {
17:
18:         private static final TurtlePlayer theTurtlePlayer = new TurtlePlayer();
19:         private Turtle turtle, bestTurtle;
20:         private ArrayList<Turtle> bestTurtles = new ArrayList<Turtle>();
21:         private final World theWorld = World.getInstance();
22:         private final TurtleField theField = TurtleField.getInstance();
23:
24:         public static TurtlePlayer getInstance() {
25:                 return theTurtlePlayer;
26:         }
27:
28:         private TurtlePlayer() {
29:                 bestTurtle = new Turtle();
30:                 bestTurtle.fitness = Double.MAX_VALUE;
31:
32:         }
33:
34:         @Override
35:         protected Turtle doInBackground() throws Exception {
36:                 while (!theWorld.getCurrentGen().done()) {
37:
38:                         // get current turtle
39:                         turtle = theWorld.getCurrentGen().getCurrentTurtle();
40:                         // update display
41:
42:                         // run through path and calculate fitness
43:                         while (!turtle.done()) {
44:                                 turtle.calculateFitness();
45:                         }
46:
47:                         // check if its a new best
48:                         if (turtle.fitness < bestTurtle.fitness) {
49:                                 bestTurtle = turtle.clone();
50:                                 // show path/log/print fitness
51:                                 bestTurtles.add(bestTurtle);
52:                                 process(bestTurtles);
53:                                 bestTurtles.clear();
54:                         } else {
55:                                 process(null);
56:                         }
57:
58:                         theWorld.getCurrentGen().nextTurtle();
59:                 }
60:                 theWorld.nextGen();
61:                 if (theWorld.getIndex() < World.GEN_LIM) {
62:                         doInBackground();
63:                 }
64:                 return bestTurtle;
65:         }
66:
67:         @Override
68:         protected void done() {
69:                 JOptionPane.showMessageDialog(WorldFrame.getInstance(),
70:                                 "Generation Limit Reached: Path Optimization Complet
e");
71:         }
72:
73:         @Override
74:         protected void process(List<Turtle> turtles) {
75:                 if (turtles == null) {
76:                         ControlPanel.genNumber.setText("gen: " + theWorld.getIndex()
);
77:                 } else {
78:                         Collections.sort(turtles);
79:                         Turtle bestTurtle = turtles.get(0);
80:                         ControlPanel.genNumber.setText("gen: " + theWorld.getIndex()
);
81:                         ControlPanel.turtleNumber.setText("turtle: "
82:                                         + theWorld.getCurrentGen().getIndex());
83:                         ControlPanel.bestTurtleFitness.setText("Shortest Path: "
84:                                         + (int) (bestTurtle.fitness));
85:                         theField.drawTurtle(bestTurtle);
86:                 }
87:         }
88: }
```

```
 1: package gui;
 2:
 3: import java.awt.Dimension;
 4: import java.awt.Toolkit;
 5:
 6: import javax.swing.JFrame;
 7:
 8: public class WelcomeFrame extends JFrame {
 9:
10:         WelcomePanel welcomePanel;
11:         Dimension screenSize;
12:
13:         public WelcomeFrame() {
14:                 super();
15:                 this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16:                 this.setTitle("******** WELCOME********");
17:                 this.setResizable(false);
18:                 this.setUndecorated(true);
19:                 screenSize = Toolkit.getDefaultToolkit().getScreenSize();
20:                 this.setLocation((screenSize.width - WelcomePanel.w) / 2,
21:                                 (screenSize.height - WelcomePanel.h) / 2);
22:
23:                 welcomePanel = new WelcomePanel(this);
24:                 this.add(welcomePanel);
25:                 this.pack();
26:
27:                 this.setVisible(true);
28:
29:         }
30: }
```

```java
package gui;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;

import javax.swing.JPanel;

public class WelcomePanel extends JPanel implements ActionListener {

	Dimension screenSize;
	static int w = 960, h = 600;
	public static BufferedImage welcomeBackground;
	public static ImageButton createMaster, runWorld, settings, exit;
	WelcomeFrame parent;

	public WelcomePanel(WelcomeFrame parent) {
		super();

		this.parent = parent;
		this.setPreferredSize(new Dimension(w, h));
		exit = new ImageButton(40, 40, ImageButton.exitImg,
				ImageButton.exitImgPressed, "");
		settings = new ImageButton(40, 40, ImageButton.settingsImg,
				ImageButton.settingsImgPressed, "Settings");
		createMaster = new ImageButton(400, 80, null, null, "Create Master")
;

		runWorld = new ImageButton(400, 80, null, null, "Run World");
		this.add(exit);
		this.add(createMaster);
		this.add(runWorld);
		this.add(settings);
		exit.addActionListener(this);
		createMaster.addActionListener(this);
		settings.addActionListener(this);
		runWorld.addActionListener(this);
	}

	@Override
	public void paintComponent(Graphics g) {
		super.paintComponent(g);
		exit.setLocation(w - 40 - 2, 2);
		createMaster.setLocation((w - 400) / 2, 300);
		runWorld.setLocation((w - 400) / 2, 400);
		settings.setLocation(20, h - 60);
		g.drawImage(welcomeBackground, 0, 0, w, h, this);
	}

	@Override
	public void actionPerformed(ActionEvent e) {
		if (e.getSource() == exit) {
			System.exit(1);
		} else if (e.getSource() == createMaster) {
			CreateMasterFrame.getInstance().setVisible(true);
			parent.dispose();
		} else if (e.getSource() == runWorld) {
			WorldFrame.getInstance().setVisible(true);
			parent.dispose();
		} else if (e.getSource() == settings) {
			// open settings pop-up
			new SettingsDialog();
		}
	}

}
```

```
 1: package gui;
 2:
 3: import java.awt.BorderLayout;
 4: import java.awt.GridLayout;
 5:
 6: import javax.swing.JFrame;
 7: import javax.swing.JPanel;
 8:
 9: public class WorldFrame extends JFrame {
10:
11:         private static final WorldFrame theWorldFrame = new WorldFrame();
12:
13:         public static WorldFrame getInstance() {
14:                 return theWorldFrame;
15:         }
16:
17:         public ControlPanel controlPanel;
18:         private Menu menu;
19:
20:         private WorldFrame() {
21:                 super();
22:
23:                 this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24:                 this.setTitle("World");
25:
26:                 controlPanel = new ControlPanel(this);
27:
28:                 menu = new Menu(this);
29:                 this.add(menu, BorderLayout.NORTH);
30:                 JPanel fieldPanel = new JPanel();
31:                 fieldPanel.setLayout(new GridLayout(1, 2));
32:                 fieldPanel.add(TurtleField.getInstance());
33:                 fieldPanel.add(MasterField.getInstance());
34:                 this.add(fieldPanel, BorderLayout.CENTER);
35:                 this.add(controlPanel, BorderLayout.SOUTH);
36:                 this.setResizable(false);
37:                 this.pack();
38:
39:                 this.setVisible(true);
40:         }
41:
42: }
```