```java
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.border.*;
import java.util.*;


/**
 * RobotPlanning - 1030 GUI Demonstration.
 *
 * @author William Soukoreff
 */
public class RobotPlanning extends JFrame implements ActionListener
{
   public static void main(String[] args)
   {
      RobotPlanning jframe = new RobotPlanning();
      jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      jframe.setTitle("Robot Path Planning");
      jframe.pack();
      jframe.setResizable(false);
      jframe.setVisible(true);
   }


   ////////////////////////////////////////////////////////////
   //               This Part Handles the GUI
   ////////////////////////////////////////////////////////////


   /*
    * GUI components we need to define
    */

   private JButton go;
   private JButton clear;
   private JButton exit;


   /*
    * some constants
    */
   static final int PANEL_WIDTH  = 500;
   static final int PANEL_HEIGHT = 400;


   // this is the JPanel where we'll draw the universe
   DrawPanel drawpanel;


   /*
    * the constructor for our JFrame object
    */
   public RobotPlanning()
   {
      /*
       * construct and configure GUI components
       */
      go    = new JButton("Go");
      clear = new JButton("Clear");
      exit  = new JButton("Exit");
      drawpanel = new DrawPanel();
      drawpanel.setBackground(Color.WHITE);
      drawpanel.setPreferredSize(new Dimension(PANEL_WIDTH, PANEL_HEIGHT));
      drawpanel.setMaximumSize(new Dimension(PANEL_WIDTH, PANEL_HEIGHT));

      /*
       * add listeners
       */
      go.addActionListener(this);
      clear.addActionListener(this);
      exit.addActionListener(this);

      /*
       * arrange components
       */
      JPanel leftPanel = new JPanel();
      leftPanel.setLayout(new BoxLayout(leftPanel, BoxLayout.Y_AXIS));
      leftPanel.add(drawpanel);

      JPanel rightPanel = new JPanel();
      rightPanel.setLayout(new BoxLayout(rightPanel, BoxLayout.Y_AXIS));
      rightPanel.add(go);
      rightPanel.add(Box.createRigidArea(new Dimension(0, 10)));
      rightPanel.add(clear);
      rightPanel.add(Box.createRigidArea(new Dimension(0, 10)));
      rightPanel.add(exit);
      rightPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

      leftPanel.setAlignmentY(Component.TOP_ALIGNMENT);
      rightPanel.setAlignmentY(Component.TOP_ALIGNMENT);

      JPanel p = new JPanel();
      p.setLayout(new BoxLayout(p, BoxLayout.X_AXIS));
      p.add(leftPanel);
      p.add(rightPanel);
      p.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

      // make paint panel this JFrame's content pane
      setContentPane(p);
   }


   /*
    * implement ActionListener method
    *
    * this handles clicks from the
    * three buttons - go, clear, and exit
    */

   public void actionPerformed(ActionEvent ae)
   {
      Object source = ae.getSource();

      if(source == go)
      {
         // the important part of the following code
         // is that it calls "findPath()"
         ( new Thread() {
            public void run()
            {
               Node head = findPath(new Node(startX, startY, null));
               Node pointer = head;
               System.out.println("A Path was Found!");
               while(pointer != null)
               {
                  System.out.println("  (" + pointer.x + ", " + pointer.y + ")");
                  pointer = pointer.next;
               }
            }
         } ).start();
      }

      else if(source == clear)
      {
         // any locations in the universe that have been
         // visited are cleared
         for(int x = 0; x < universe[0].length; x++)
            for(int y = 0; y < universe.length; y++)
               if(universe[y][x] == VISITED)
                  universe[y][x] = EMPTY;
         drawpanel.repaint();
      }

      else if(source == exit)
         System.exit(0);
   }


   ////////////////////////////////////////////////////////////
   //               This Part Handles the Display
   ////////////////////////////////////////////////////////////


   /*
    * This inner class extends JPanel
    *
    * this is where we do our drawing...
    */
   class DrawPanel extends JPanel
   {
      // constuctor
      public DrawPanel()
      {
         super();
      }


      // do our drawing...
      public void paintComponent(Graphics g)
      {
         super.paintComponent(g);

         final int minX = 10;
         final int minY = 10;
         final int maxX = 490;
         final int maxY = 390;
         final int rowWidth  = (maxX - minX) / 10;
         final int colHeight = (maxY - minY) / 10;

         int x, y, i, j;

         // first we draw the map
         for(x= 0; x < 10; x++)
            for(y = 0; y < 10; y++)
            {
               if(universe[y][x] == EMPTY)
                  ;
```

```java
            else if(universe[y][x] == GOAL)
            {
               g.setColor(Color.GREEN);
               g.fillRect(minX + x*rowWidth, minY + y*colHeight,
                     rowWidth, colHeight);
            }

            else if(universe[y][x] == START)
            {
               g.setColor(Color.RED);
               g.fillRect(minX + x*rowWidth, minY + y*colHeight,
                     rowWidth, colHeight);
            }

            else if(universe[y][x] == WALL)
            {
               g.setColor(Color.BLUE);
               g.fillRect(minX + x*rowWidth, minY + y*colHeight,
                     rowWidth, colHeight);
            }

            else if(universe[y][x] == VISITED)
            {
               g.setColor(Color.GRAY);
               g.fillRect(minX + x*rowWidth, minY + y*colHeight,
                     rowWidth, colHeight);
            }
         }


      // next we draw the grid
      g.setColor(Color.BLACK);
      for(i = 0; i <= 10; i++)
      {
         g.drawLine(minX + i*rowWidth, minY, minX + i*rowWidth, maxY);
         g.drawLine(minX, minY + i*colHeight, maxX, minY + i*colHeight);
      }

   }
}


////////////////////////////////////////////////////////////
//                 This Part Handles the AI
////////////////////////////////////////////////////////////

/*
 * For this problem our 'universe' will consist of a
 * 10 x 10 space.  The geometry of the obstacle is
 * described using the array below
 */

// These describe the states that locations in our
// universe can be in
static final int EMPTY   = 0;
static final int GOAL    = 1;
static final int START   = 2;
static final int WALL    = 3;
static final int VISITED = 4;


// the universe consists of a 10 x 10 area
// (we have filled-in the obstacle (walls)
   // directly into the array
   static final int[][] universe = {
    //  0  1  2  3  4  5  6  7  8  9
      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 0
      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 1,
      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 2,
      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 3,
      { 0, 0, 3, 3, 3, 3, 3, 3, 0, 0 },  // 4,
      { 0, 0, 3, 0, 0, 0, 0, 3, 0, 0 },  // 5,
      { 0, 0, 3, 0, 0, 0, 0, 3, 0, 0 },  // 6,
      { 0, 0, 3, 0, 0, 0, 0, 3, 0, 0 },  // 7,
      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  // 8,
      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }   // 9,
   };


   // these define the start and goal locations
   static final int startX = 4;
   static final int startY = 6;
   static final int goalX = 4;
   static final int goalY = 2;

   // this inserts the start and goal into the
   // universe array
   static
   {
      universe[startY][startX] = START;
      universe[goalY][goalX]   = GOAL;
   }


   // we are going to use a linked-list to
   // remember the path from start to goal
   static class Node
   {
      int  x;
      int  y;
      Node next;
      Node(int x, int y, Node next)
         { this.x = x; this.y = y; this.next = next; }
   }


   // this important function checks to see
   // whether a proposed movement is valid
   //
   // it checks:
   // 1) whether the move would exit the
   //    universe (which is not allowed)
   // 2) whether the move would hit the
   //    obstacle (also not allowed)
   // 3) whether the move would go to
   //    somewhere we have already been
   //    (which would be inefficient)
   boolean checkPath(int x, int y, Node head)
   {
      // 1) move out of the universe?
      if(x < 0 || x >= universe[0].length)
         return false;

      // 1) move out of the universe?
      if(y < 0 || y >= universe.length)
         return false;
      // 2) move would hit the obstacle or
      //    go where we've already been?
      if(universe[y][x] > GOAL)
         return false;

      // mark the location as visited
      if(universe[y][x] == EMPTY)
         universe[y][x] = VISITED;

      // redraw the display so we can watch
      // the algorithm's progress
      drawpanel.repaint();

      // delay for 100 ms, so it doesn't
      // all happen too quickly
      try {
         Thread.sleep(100);
      } catch(Exception e) {};

      return true;
   }


   Node findPath(Node head)
   {
      // if our location is the goal, then
      // because we're done
      // (our return value is the head of
      // the linked-list of locations)
      if(head.x == goalX && head.y == goalY)
         return head;

      else
      {
         // try to go left:  x = x - 1
         Node foundit = null;
         if(checkPath(head.x - 1, head.y, head))
         {
            foundit = findPath(new Node(head.x - 1, head.y, head));
            if(foundit != null)
               return foundit;
         }

         // try to go right:  x = x + 1
         if(checkPath(head.x + 1, head.y, head))
         {
            foundit = findPath(new Node(head.x + 1, head.y, head));
            if(foundit != null)
               return foundit;
         }

         // try to go up:  y = y - 1
         if(checkPath(head.x, head.y - 1, head))
         {
            foundit = findPath(new Node(head.x, head.y - 1, head));
            if(foundit != null)
               return foundit;
         }

         // try to go down:  y = y + 1
         if(checkPath(head.x, head.y + 1, head))
         {
            foundit = findPath(new Node(head.x, head.y + 1, head));
            if(foundit != null)
```

```
               return foundit;
         }

         // didn't find the solution
         return null;
      }
   }
}
```