```
/**
 * Created by Stephen Sheridan on 16/02/14.
 */
// Inverted Pendulum simulation
//   by Chuck Anderson, 1998, with code from O'Reilly's "Java by Example"

import java.applet.*;
import java.awt.*;
import java.awt.event.*;

import com.fuzzylite.*;
import com.fuzzylite.defuzzifier.*;
import com.fuzzylite.norm.s.*;
import com.fuzzylite.norm.t.*;
import com.fuzzylite.rule.*;
import com.fuzzylite.term.*;
import com.fuzzylite.variable.*;

public class Pole2 extends Applet implements Runnable {


        int delay; //Frame rate control
   Thread animatorThread;
   //next three are for double-buffering
   Dimension offDimension;
   Image offImage;
   Graphics offGraphics;

   // Vars to store current pole and cart position and previous positions
   double action;
   double pos, posDot, angle, angleDot;

   // Constants used for physics
   public static final double cartMass=1.;
   public static final double poleMass=0.1;
   public static final double poleLength=1.;
   public static final double forceMag=10.;
   public static final double tau=0.02;
   public static final double fricCart=0.00005;
   public static final double fricPole=0.005;
   public static final double totalMass = cartMass + poleMass;
   public static final double halfPole = 0.5 * poleLength;
   public static final double poleMassLength = halfPole * poleMass;
   public static final double fourthirds = 4./3.;

   // TODO
        // Declare the Fuzzy logic global variables here
```

```
        InputVariable inputVariable1 = new InputVariable();
        InputVariable inputVariable2 = new InputVariable();

// Define the Engine
        Engine engine = new Engine();

// Define InputVariable1 Theta(t) {angle with perpendicular}
// Define InputVariable1 x(t) {angular velocity}
// OutputVariable {force to be applied}
        OutputVariable outputVariable = new OutputVariable();

// Define the RuleBlock
        RuleBlock ruleBlock = new RuleBlock();

public void init() {
   String str;
   int fps = 20;

   // Initialize pole state.
   pos = 0.;
   posDot = 0.;
   angle = .2; // Pole starts off at an angle
   angleDot = 0.;
   action = 0;

   // Setup the Fuzzy Controller
   FuzzyControllerSetup();

   // Set up animation timing.
   //How many milliseconds between frames?
   str = getParameter("fps");
   try {
      if (str != null) {
         fps = Integer.parseInt(str);
      }
   } catch (Exception e) {}
   delay = (fps > 0) ? (1000 / fps) : 100;

   // Handle keyboard events
   // LEFT KEY = push the cart left APPLY FORCE FROM RIGHT
   // RIGHT KEY = push the cart right APPLY FORCE FROM LEFT
   // SPACE KEY = Reset pole to centre
   this.addKeyListener(new KeyAdapter() {
      public void keyPressed(KeyEvent e) {
         if (e.getKeyCode() == KeyEvent.VK_LEFT)
            action = -1;
         else if (e.getKeyCode() == KeyEvent.VK_RIGHT)
```

```
            action = 1;
          else if (e.getKeyChar() == ' ') {
            action = 0;
            resetPole();
          }
        }
    });

}

public void start() {
    //Start animating!
    if (animatorThread == null) {
        animatorThread = new Thread(this);
    }
    animatorThread.start();
}

public void stop() {
    //Stop the animating thread.
    animatorThread = null;
    //Get rid of the objects necessary for double buffering.
    offGraphics = null;
    offImage = null;
}

public void run() {
    //Remember the starting time.
    long startTime = System.currentTimeMillis();

    //This is the animation loop.
    while (Thread.currentThread() == animatorThread) {

                            //Update the state of the pole;
        // First calc derivatives of state variables
        double force = forceMag * action;
        //double force = action;
        double sinangle = Math.sin(angle);
        double cosangle = Math.cos(angle);
        double angleDotSq = angleDot * angleDot;
        double common = (force + poleMassLength * angleDotSq * sinangle
              - fricCart * (posDot<0 ? -1 : 0)) / totalMass;
        double angleDDot = (9.8 * sinangle - cosangle * common
              - fricPole * angleDot / poleMassLength) /
              (halfPole * (fourthirds - poleMass * cosangle * cosangle /
                    totalMass));
        double posDDot = common - poleMassLength * angleDDot * cosangle /
```

```
            totalMass;

        //Now update current state.
        pos += posDot * tau;
        posDot += posDDot * tau;
        angle += angleDot * tau;
        angleDot += angleDDot * tau;

        // TODO
                        // Above values represent current state of the cart and pole
        // Control system should take these values and make decision.
        // We are interested in angle and angleDot;
        // So we need a function here to set the state of the action for the next
        // update in time.

                        double output = FuzzyControllerRun(angle, angleDot);
        action = output;

        //Display it.
        repaint();

        //Delay depending on how far we are behind.
        try {
           startTime += delay;
           Thread.sleep(Math.max(0,
                startTime-System.currentTimeMillis()));
        } catch (InterruptedException e) {
           break;
        }
     }
}

public void paint(Graphics g) {
   update(g);
}

public void update(Graphics g) {
   Dimension d = getSize();
   Color cartColor = new Color(0,20,255);
   Color arrowColor = new Color(255,255,0);
   Color trackColor = new Color(100,100,50);

   //Create the off-screen graphics context, if no good one exists.
   if ( (offGraphics == null)
         || (d.width != offDimension.width)
         || (d.height != offDimension.height) ) {
      offDimension = d;
```

```
        offImage = createImage(d.width, d.height);
        offGraphics = offImage.getGraphics();
    }

    //Erase the previous image.
    offGraphics.setColor(getBackground());
    offGraphics.fillRect(0,0,d.width,d.height);

    //Draw Track.
    double xs[] = {-2.5, 2.5, 2.5, 2.3, 2.3, -2.3, -2.3, -2.5};
    double ys[] = {-0.4, -0.4, 0., 0., -0.2, -0.2, 0, 0};
    int pixxs[] = new int[8], pixys[] = new int[8];
    for (int i = 0; i<8; i++) {
        pixxs[i] = pixX(d,xs[i]);
        pixys[i] = pixY(d,ys[i]);
    }
    offGraphics.setColor(trackColor);
    offGraphics.fillPolygon(pixxs,pixys,8);

    //Draw message
    // String msg = "Left Mouse Button: push left    Right Mouse Button: push right
Middle Button: PANIC";
    String msg = "Position = " + pos + " Angle = " + angle + " angleDot = " + angleDot;
    offGraphics.drawString(msg,20,d.height-20);

    //Draw cart.
    offGraphics.setColor(cartColor);
    offGraphics.fillRect(pixX(d,pos-0.2), pixY(d,0), pixDX(d,0.4), pixDY(d,-0.2));

    //Draw pole.
    //    offGraphics.setColor(cartColor);
    offGraphics.drawLine(pixX(d,pos),pixY(d,0),
          pixX(d,pos+Math.sin(angle)*poleLength),
          pixY(d,poleLength*Math.cos(angle)));

    //Draw action arrow.
    if (action != 0) {
        int signAction = (action > 0 ? 1 : (action < 0) ? -1 : 0);
        int tipx = pixX(d,pos+0.2*signAction);
        int tipy = pixY(d,-0.1);
        offGraphics.setColor(arrowColor);
        offGraphics.drawLine(pixX(d,pos),pixY(d,-0.1),tipx,tipy);
        offGraphics.drawLine(tipx,tipy,tipx-4*signAction,tipy+4);
        offGraphics.drawLine(tipx,tipy,tipx-4*signAction,tipy-4);
    }
```

```
    //Last thing: Paint the image onto the screen.
    g.drawImage(offImage, 0, 0, this);

}

public int pixX(Dimension d, double v) {
    return (int) Math.round((v + 2.5) / 5.0 * d.width);
}

public int pixY(Dimension d, double v) {
    return (int) Math.round(d.height - (v + 2.5) / 5.0 * d.height);
}

public int pixDX(Dimension d, double v) {
    return (int) Math.round(v / 5.0 * d.width);
}

public int pixDY(Dimension d, double v) {
    return (int) Math.round(-v / 5.0 * d.height);
}

public void resetPole() {
    pos = 0.;
    posDot = 0.;
    angle = 0.;
    angleDot = 0.;
}



public void FuzzyControllerSetup()
{
        // STEP 1
                // Create the engine & set the name
        engine.setName("CartEngine");

    // STEP 2
                // Create and setup the first input variable [Theta(t)]
    inputVariable1.setEnabled(true);
    inputVariable1.setName("angle");
    inputVariable1.setRange(-4.000, 4.000);
    inputVariable1.addTerm(new Trapezoid("N", -4.000, -4.000, -1.000, 0.000));
    inputVariable1.addTerm(new Triangle("Z", -2.000, 0.000, 2.000));
    inputVariable1.addTerm(new Trapezoid("P", 0.000, 1.000, 4.000, 4.000));
    engine.addInputVariable(inputVariable1);

    // STEP 3
```

```
                // Create and setup the second input variable [x(t)]
        inputVariable2.setEnabled(true);
        inputVariable2.setName("angular_velocity");
        inputVariable2.setRange(-8.000, 8.000);
        inputVariable2.addTerm(new Trapezoid("N", -8.000, -8.000, -3.000, -2.000));
        inputVariable2.addTerm(new Trapezoid("Z", -3.000, -2.000, 2.000, 3.000));
        inputVariable2.addTerm(new Trapezoid("P", 2.000, 3.000, 8.000, 8.000));
        engine.addInputVariable(inputVariable2);

        // STEP 4
                // Create and setup the output variable [force]
        outputVariable.setEnabled(true);
        outputVariable.setName("force");
        outputVariable.setRange(-32.000, 32.000);
        outputVariable.fuzzyOutput().setAccumulation(new Maximum());
        outputVariable.setDefuzzifier(new Centroid(200));
        outputVariable.setDefaultValue(Double.NaN);
        outputVariable.setLockValidOutput(false);
        outputVariable.setLockOutputRange(false);
        outputVariable.addTerm(new Trapezoid("N", -32.000, -32.000, -5.000, -1.000));
        outputVariable.addTerm(new Trapezoid("Z", -3.000, -1.000, 1.000, 3.000));
        outputVariable.addTerm(new Trapezoid("P", 1.000, 5.000, 32.000, 32.000));
        engine.addOutputVariable(outputVariable);

        // STEP 5
                // Create the rule block and add the fuzzy rules
        ruleBlock.setEnabled(true);
        ruleBlock.setName("");
        ruleBlock.setConjunction(new Minimum());
        ruleBlock.setDisjunction(new Maximum());
        ruleBlock.setActivation(new Minimum());
        ruleBlock.addRule(Rule.parse("if angle is N and angular_velocity is N then force is N",
engine));
        ruleBlock.addRule(Rule.parse("if angle is Z and angular_velocity is N then force is N",
engine));
        ruleBlock.addRule(Rule.parse("if angle is P and angular_velocity is N then force is Z",
engine));
        ruleBlock.addRule(Rule.parse("if angle is N and angular_velocity is Z then force is N",
engine));
        ruleBlock.addRule(Rule.parse("if angle is Z and angular_velocity is Z then force is Z",
engine));
        ruleBlock.addRule(Rule.parse("if angle is P and angular_velocity is Z then force is P",
engine));
        ruleBlock.addRule(Rule.parse("if angle is N and angular_velocity is P then force is Z",
engine));
        ruleBlock.addRule(Rule.parse("if angle is Z and angular_velocity is P then force is P",
engine));
```

```
      ruleBlock.addRule(Rule.parse("if angle is P and angular_velocity is P then force is P",
engine));
      engine.addRuleBlock(ruleBlock);
   }

   public double FuzzyControllerRun(double angle, double angleDot)
   {
      // STEP 1
                  // Use inputVariable1.setInputValue(double) to set the value for the first
input variable [Theta(t)]
          inputVariable1.setInputValue(angle);
          // STEP 2
                  // Use inputVariable2.setInputValue(double) to set the value for the second
input variable [x(t)]
                  inputVariable2.setInputValue(angleDot);
                  // STEP 3
      // Call engine.process();
          engine.process();
      // STEP 4
                  // return outputVariable.defuzzify();
          return outputVariable.defuzzify();
   }

}
```