```java
package model;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.geom.Point2D; //test//
import java.util.ArrayList;
import java.util.Random;

public class Car {
        private static int carsCreated = 0;

        private Point2D.Float coordinate;
        private final int id;
        private float desiredSpeed;
        private float currentSpeed;
        private Random rng = new Random();
        private Road currentRoad;
        private Lane currentLane;
        /*
         * targetLane: not used to its full purpose in current version. Was supp
osed
         * to be part of inner road lane change
         */
        private Lane targetLane;
        /*
         * target connection: connection it aims to make a turn at. Can be null
if
         * the car decides to reach an exit point
         */
        private Connection targetConnection;
        private CarWorld cWorld;

        /*
         * distanceTravelled: distance travelled on the current road, required f
or
         * perception sequence
         */
        private float distanceTravelled;

        public Car(Point2D.Float coordinate, float ds, Lane initialLane,
                        CarWorld cWorld, Point2D.Float entryPoint) {
                this.id = Car.carsCreated;
                Car.carsCreated++;
                this.coordinate = coordinate;
                this.desiredSpeed = ds;
                this.enterLane(initialLane, entryPoint);
                this.cWorld = cWorld;

        }

        public static int totalCar() {
                return Car.carsCreated;
        }

        public Road getRoad() {
                return this.currentRoad;
        }

        public float getTravelled() {

                return this.distanceTravelled;
        }

        public static void setCarsCreated(int i) {

                carsCreated = i;

        }
```

```java
        /*
         * checkCourse: not used to its full purpose in current version. Was
         * supposed to be part of inner road lane change. Currently being used a
s a
         * part of speed decision when making a turn
         */
        private boolean checkCourse() {

                return (this.currentLane.equals(this.targetLane));
        }

        public int getId() {
                return this.id;
        }

        public void setTravelled(float f) {
                this.distanceTravelled = f;
        }

        public Point2D.Float getCoordinate() {
                return coordinate;
        }

        public void setCoordinate(Point2D.Float coordinate) {
                this.coordinate = coordinate;
        }

        public float getDesiredSpeed() {
                return desiredSpeed;
        }

        public void setDesiredSpeed(float ds) {
                this.desiredSpeed = ds;
        }

        public float getCurrentSpeed() {
                return currentSpeed;
        }

        public void setCurrentSpeed(float currentSpeed) {
                this.currentSpeed = currentSpeed;
        }

        public Lane getCurrentLane() {
                return currentLane;
        }

        public void setCurrentLane(Lane currentLane) {
                this.currentLane = currentLane;
        }

        /*
         * enterLane: lane entering logic. If the road of the lanes change,
         * enterRoad will be called as well
         */
        public void enterLane(Lane lane, Point2D.Float entryPoint) {
                if (this.currentLane != null) {
                        this.currentLane.carLeaves(this);
                }
                if (lane instanceof StraightLane) {
                        // entering a straightlane
                        this.setTravelled(lane.findDistance(this));
                } else if (lane instanceof Connection) {
                        // entering a connection
                        this.setTravelled(1);
                        this.targetConnection = null;

                }
                this.coordinate = entryPoint;
```

```
		Road tRoad = lane.getRoad();
		Lane previousLane = this.currentLane;
		Road previousRoad;
		if (previousLane == null) {
			previousRoad = null;
		} else {
			previousRoad = previousLane.getRoad();
		}
		this.currentLane = lane;
		lane.carEnters(this);
		if (!tRoad.equals(previousRoad)) {
			System.out.println("Entering road");
			enterRoad(tRoad);
		}

	}

	private void enterRoad(Road tRoad) {

		/* gets the whole connections available on */
		ArrayList<Connection> connections = currentLane.getSameConnectio
ns();
		/* sanitizes the list to have connections that are ahead of the
cars */
		ArrayList<Connection> legalConnections = new ArrayList<Connectio
n>();
		for (int i = 0; i < connections.size(); i++) {
			Connection cc = connections.get(i);
			if (Car.distance(this.currentLane.getStart(), cc.getStar
t()) >= this.distanceTravelled) {
				legalConnections.add(cc);
			}
		}
		boolean ending = currentLane.isEnding();
		/* dummy: Connection representing an exit point */
		Connection dummy = new Connection();
		if (ending) {
			legalConnections.add(dummy);
		}

		int random = rng.nextInt(legalConnections.size());
		Connection chosen = legalConnections.get(random);

		/*
		 * if the dummy connection is randomly chosen it will aim to rea
ch the
		 * exit point of the car. Other wise appropriate state changes o
ccur for
		 * making turns
		 */
		if (chosen.equals(dummy)) {
			ArrayList<Lane> sameLanes = currentLane.getSameLanes();
			int nr = rng.nextInt(sameLanes.size());
			Lane chosenLane = sameLanes.get(nr);
			this.targetLane = chosenLane;
			this.targetConnection = null;
		} else {

			this.targetLane = chosen.getStartLane();
			if (Car.distance(this.currentLane.getStart(), chosen.get
Start()) >= this.distanceTravelled) {
				this.targetConnection = chosen;
			} else {
				this.targetConnection = null;
			}
		}
		this.currentRoad = tRoad;
	}
```

```
	/* move: decision making as well as updating its coordinate */
	public void move() {

		// perception sequence: START

		Car frontCar = this.currentLane.getFrontCar(this);
		TrafficLight tfl = this.currentLane.getNextTrafficLight(this);
		float cd = 100;
		float td = 100;
		if (frontCar != null) {
			cd = Car.distance(frontCar.getCoordinate(), this.getCoor
dinate());
		}
		if (tfl != null) {
			td = Car.distance(tfl.getCoordinate(), this.getCoordinat
e());
		}
		// current---car---trafficlight(red/green)----100m :: react to t
he car
		// current----trafficlight(green)---car--100m :: react to the ca
r
		// current---trafficlight(red)----car ---100m :: react to the li
ght
		// current -----100m----x-----y :: free run
		if (cd >= 100 && td >= 100) {
			// free run
			accelerate();
		} else if (td < cd && td < 100 && tfl.getStatus().equals("red"))
 {
			// react to the red light
			if (td < 15)
				this.setCurrentSpeed(0);
			else if (td < 70)
				this.setCurrentSpeed(td);
			else if (td < 100) {
				if (this.currentSpeed > td) {
					decelerate();
				}
			}

		} else if ((td < cd && td < 100 && tfl.getStatus().equals("green"
))) {
			// react to the green light-> ignore the light and react
 to the
			// closest car or slow down if making turn
			if (this.targetConnection != null
					&& checkCourse()
					&& Car.distance(this.coordinate,
							this.targetConnection.ge
tStart()) < 100) {
				if (td < 70)
					if (currentSpeed < 60)
						this.accelerate();
					else if (currentSpeed > 80) {
						this.decelerate();
					}
			} else {
				if (cd < 15) {
					this.setCurrentSpeed(0);
				} else if (cd < 70) {
					this.setCurrentSpeed(cd);
				} else if (cd < 100) {
					if (this.currentSpeed > cd) {
						decelerate();
					}
				} else {
					accelerate();
				}
```

```
                                }
                } else if ((cd < td && cd < 100)) {
                        // react to the car
                        if (cd < 15) {
                                this.setCurrentSpeed(0);
                        } else if (cd < 70) {
                                this.setCurrentSpeed(cd);
                        } else if (cd < 100) {
                                if (this.currentSpeed > cd)
                                        decelerate();
                        }
                }
                // perception sequence: END

                // action sequence: START
                if (this.currentSpeed == 0) {
                        if (frontCar != null) {
                                System.out.println("front car : " + frontCar.coordina
te);
                        }
                } else {

                        float tempDistance = this.currentSpeed * 0.02f;

                        Point2D.Float nextPosition = this.currentLane.nextPositi
on(this,
                                        tempDistance, this.distanceTravelled);

                        // state change sequence: Additional perception and chan
ge in
                        // states: START
                        Point2D.Float nextDisplacement = new Point2D.Float(
                                        Math.abs(nextPosition.x - this.coordinat
e.x),
                                        Math.abs(nextPosition.y - this.coordinat
e.y));

                        Point2D.Float dToEnd = new Point2D.Float(Math.abs(curren
tLane
                                        .getEnd().x - this.getCoordinate().x), M
ath.abs(currentLane
                                        .getEnd().y - this.getCoordinate().y));

                        if ((this.currentLane instanceof StraightLane)
                                        && (dToEnd.x < nextDisplacement.x || dTo
End.y < nextDisplacement.y)) {
                                // reached an exit point
                                this.coordinate = this.currentLane.getEnd();
                                this.remove();
                        } else if ((this.currentLane instanceof Connection)) {
                                Point2D.Float lastPoint = this.currentLane.getEn
d();
                                if (Car.distance(this.coordinate, lastPoint) < 5
) {
                                        this.enterLane(
                                                        ((Connection) this.curre
ntLane).getTargetLane(),
                                                        ((Connection) this.curre
ntLane).getEnd());
                                }
                        }
                        this.coordinate = nextPosition;

                        if (this.targetConnection != null) {

                                if (Car.distance(this.currentLane.getStart(),
                                                targetConnection.getStart()) <=
this.distanceTravelled
                                                && checkCourse()) {
```

```
                                        this.enterLane(targetConnection,
                                                        targetConnection.getStar
t());
                                }

                        }

                }
                // State change sequence: END
                // action sequence: END
        }

        private void decelerate() {
                // TODO Auto-generated method stub
                this.currentSpeed -= 500 * 0.02;
        }

        private void accelerate() {
                // TODO Auto-generated method stub
                if (this.currentSpeed < this.desiredSpeed) {
                        this.currentSpeed += 30 * 0.02;
                }
        }

        public void remove() {
                // TODO Auto-generated method stub
                this.cWorld.removeCar(this);
                this.currentLane.removeCar(this);
                System.gc();

        }

        public void paint(Graphics g) {
                if (this.currentSpeed >= 160) {
                        g.setColor(Color.RED);
                } else if (this.currentSpeed < 160 && this.currentSpeed >= 120)
{
                        g.setColor(Color.ORANGE);
                } else if (this.currentSpeed < 120 && this.currentSpeed >= 80) {
                        g.setColor(Color.YELLOW);
                } else if (this.currentSpeed < 80 && this.currentSpeed >= 40) {
                        g.setColor(Color.MAGENTA);
                } else if (this.currentSpeed < 40 && this.currentSpeed > 0) {
                        g.setColor(Color.BLUE);
                } else if (this.currentSpeed == 0) {
                        g.setColor(Color.BLACK);
                }
                g.fillOval((int) coordinate.x - 4, (int) coordinate.y - 4, 8, 8)
;

        }

        public static float distance(Point2D.Float p1, Point2D.Float p2) {
                return (float) Math.sqrt(Math.pow((p1.x - p2.x), 2.0)
                                + Math.pow(p1.y - p2.y, 2.0));
        }

}
```

```
package model;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.geom.Point2D;
import java.util.Random;

public class CarPark {
        public static final int START = 0;
        public static final int END = 1;
        private int parkId;
        private int type;
        private Point2D.Float coordinate;
        private static int parksCreated = 0;
        private Random rng = new Random();
        // higher the spawn rate, more frequent car spawn
        private double spawnRate;
        private Lane lane;
        private CarWorld world;
        // attribute to check that the park does not spawn another car too soon.
        // distance is checked between the car park and the previous car it spaw
ned.
        private Car previousCar = null;

        public CarPark(Lane bLane, int type, CarWorld cWorld) {
                this.parkId = parksCreated;
                this.lane = bLane;
                this.spawnRate = 0.003d;
                this.type = type;
                this.world = cWorld;
                if (type == START) {
                        this.coordinate = lane.getStart();
                } else if (type == END) {
                        this.coordinate = lane.getEnd();
                }
                parksCreated++;
        }

        public int getId() {
                return this.parkId;
        }

        public void setSpawn(int i) {
                this.spawnRate = ((double) i / (double) 1000);
        }

        public Point2D.Float getCoordinate() {
                return this.coordinate;
        }

        public Lane getLane() {
                return this.lane;
        }

        public void reset() {
                // TODO Auto-generated method stub
                this.previousCar = null;

        }

        public static void setParksCreated(int i) {
                // TODO Auto-generated method stub
                parksCreated = i;
        }

        public void update() {

                if (this.type == START) {
                        double range = this.spawnRate / 2;
```

```
                        double dice = Math.random();
                        int dspeed = rng.nextInt((180 - 160) + 1) + 160;
                        if (dice >= 0.5d - range && dice <= 0.5d + range
                                        && this.world.getCars().size() < 80) {
                                if (previousCar == null) {
                                        Car c = new Car(this.lane.getStart(), ds
peed, this.lane,
                                                        this.world, this.lane.ge
tStart());
                                        c.setCurrentSpeed(100);
                                        previousCar = c;
                                        this.world.addCar(c);
                                } else {
                                        if (Car.distance(previousCar.getCoordina
te(),
                                                        this.coordinate) > 15) {
                                                Car c = new Car(this.lane.getSta
rt(), dspeed,
                                                                this.lane, this.
world, this.lane.getStart());
                                                c.setCurrentSpeed(100);
                                                previousCar = c;
                                                this.world.addCar(c);
                                        }
                                }
                        }
                }
        }

        public void paint(Graphics g) {
                // TODO Auto-generated method stub
                g.setColor(Color.BLACK);
                g.fillOval(
                                (int) (coordinate.x - Math.sqrt(2 * (Math.pow(7.
5 / 2, 2)))),
                                (int) (coordinate.y - Math.sqrt(2 * (Math.pow(7.
5 / 2, 2)))),
                                15, 15);

        }

}
```

```
package view;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;

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

import control.ParkController;
import control.TrafficLightController;
import control.WorldController;

public class CarSimView extends JFrame {

        private Container mainContainer;
        private WorldController wControl;
        private TrafficLightController tlc;
        private ParkController pc;
        private JButton stopButton, startButton;
        private MainMenu mainMenu;
        private SimulationPanel simPanel;
        private ConsolePanel consolePanel;
        private TrafficLightPanel tlp;
        private UserHelpPanel usrHelpPanel;
        private boolean addingLight = false;

        public CarSimView(String title, WorldController wControl,
                        TrafficLightController tlc, ParkController pc) {
                super(title);
                mainContainer = this.getContentPane();
                BorderLayout borderLayout = new BorderLayout();
                // FlowLayout experimentLayout = new FlowLayout();
                this.setLayout(borderLayout);
                this.wControl = wControl;
                this.tlc = tlc;
                this.pc = pc;
                simPanel = new SimulationPanel(this.wControl, tlc, this);
                mainMenu = new MainMenu(this.wControl, this);
                consolePanel = new ConsolePanel(this.wControl, this, pc);

                usrHelpPanel = new UserHelpPanel(this.wControl, this);

                this.setResizable(false);
                this.setSize(1280, 670);
                mainContainer.setBackground(new Color(0,150,0));
                consolePanel.setPreferredSize(new Dimension(200, 660));
                mainMenu.setPreferredSize(new Dimension(1000, 660));
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                this.setLocationRelativeTo(null);
                this.setVisible(true);

                mainContainer.removeAll();
                mainContainer.add(mainMenu, BorderLayout.CENTER);
                this.repaint();
                this.revalidate();

        }

        public void TrafficPanel(int id) {
                mainContainer.removeAll();
                mainContainer.add(new TrafficLightPanel(this.wControl, tlc, this
, id));
                mainContainer.revalidate();
                mainContainer.repaint();
        }

        // THIS IS NEW
```

```
        public void HelpPanel() {
                mainContainer.removeAll();
                mainContainer.add(usrHelpPanel);
                mainContainer.revalidate();
                mainContainer.repaint();
        }

        public void mainMenu() {
                // TODO Auto-generated method stub
                // change panel main menu
                mainContainer.removeAll();
                mainContainer.add(mainMenu);
                mainContainer.revalidate();
                mainContainer.repaint();

        }

        public void simulationView() {
                // TODO Auto-generated method stub
                mainContainer.removeAll();
                mainContainer.add(simPanel, BorderLayout.CENTER);
                mainContainer.add(consolePanel, BorderLayout.WEST);
                mainContainer.revalidate();
                mainContainer.repaint();

        }

        public DynamicChart getDynamicChart() {

                return consolePanel.getDynamicChart();
        }

        public void setAddingLight(boolean b) {
                // TODO Auto-generated method stub

                addingLight = b;
                if (b == true) {
                        this.consolePanel.disableButtons();
                } else {
                        this.consolePanel.enableButtons();
                }

        }

        public boolean getAddingLight() {
                return this.addingLight;
        }

}
```

```
package model;

import java.awt.Point;
import java.awt.Rectangle;
import java.awt.geom.Point2D;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map.Entry;

public class CarWorld {
        private String status = "paused";
        private int height;
        private int width;
        private ArrayList<Road> roads = new ArrayList<Road>();
        private HashMap<Integer, Car> cars = new HashMap<Integer, Car>();
        private ArrayList<TrafficLight> lights = new ArrayList<TrafficLight>();
        private ArrayList<CarPark> parks = new ArrayList<CarPark>();
        private QuadTree quad;

        public CarWorld() {
                this.height = 1000;
                this.width = 800;
                quad = new QuadTree(0, new Rectangle(this.width, this.height));
        }

        public CarWorld(int height, int width) {
                this.height = height;
                this.width = width;
        }

        public int getHeight() {
                return height;
        }

        public int getWidth() {
                return width;
        }

        public void addRoad(Road road) {
                roads.add(road);
        }

        public void addCar(Car car) {
                cars.put(car.getId(), car);
        }

        public void removeRoad(String roadId) {
                roads.remove(roadId);
        }

        public String toString() {
                return "Number of roads: " + roads.size() + "Number of cars"
                                + cars.size();
        }

        public void setStatus(String s) {
                this.status = s;

        }

        public String getStatus() {
                return this.status;
        }

        public HashMap<Integer, Car> getCars() {

                return cars;
        }
```

```

        public void removeCar(Car car) {
                cars.remove(car.getId());
        }

        public Road getRoad(int roadId) {
                Road tRoad = null;
                for (int i = 0; i < roads.size(); i++) {
                        Road cRoad = roads.get(i);
                        if (cRoad.getId() == roadId) {
                                tRoad = cRoad;
                                break;
                        }
                }
                return tRoad;
        }

        public ArrayList<Road> getRoads() {

                return roads;
        }

        public ArrayList<TrafficLight> getLights() {

                return lights;
        }

        // resetting the state of the world for a new road network
        public void flush() {
                TrafficLight.setTotalLights(0);
                Car.setCarsCreated(0);
                CarPark.setParksCreated(0);
                roads = new ArrayList<Road>();
                cars = new HashMap<Integer, Car>();
                parks = new ArrayList<CarPark>();
                this.lights = new ArrayList<TrafficLight>();

        }

        public void addPark(CarPark cp) {
                System.out.println("adding car park");
                parks.add(cp);
        }

        public ArrayList<CarPark> getParks() {
                // TODO Auto-generated method stub
                return this.parks;
        }

        public void addLight(TrafficLight light) {
                // TODO Auto-generated method stub
                this.lights.add(light);

        }

        public ArrayList<Car> checkCollision() {
                quad.clear();
                ArrayList<Car> collided = new ArrayList<Car>();
                Iterator<Entry<Integer, Car>> carIt = cars.entrySet().iterator()
;
                while (carIt.hasNext()) {
                        quad.insert(carIt.next().getValue());
                }
                ArrayList<Car> returnObjects = new ArrayList<Car>();
                carIt = cars.entrySet().iterator();

                while (carIt.hasNext()) {
                        returnObjects.clear();
                        Car currentCar = carIt.next().getValue();
                        quad.retrieve(returnObjects, currentCar);
                        for (int i = 0; i < returnObjects.size(); i++) {
                                if ((!returnObjects.get(i).equals(currentCar))
                                                && Car.distance(returnObjects.ge
t(i).getCoordinate(),
                                                                currentCar.getCo
ordinate()) < 7) {
                                        this.status = "paused";
                                        collided.add(currentCar);
                                        collided.add(returnObjects.get(i));
                                        return collided;

                                }
                        }
                }
                return collided;
        }

        public void reset() {
                // returning to the initial state of the current simulation

                for (int i = 0; i < roads.size(); i++) {

                        Road cr = roads.get(i);

                        Iterator<Entry<Integer, Lane>> lIt = cr.getLanes().entry
Set()
                                        .iterator();
                        while (lIt.hasNext()) {
                                Lane currentLane = lIt.next().getValue();
                                currentLane.getCarsInLane().clear();
                                Iterator<Entry<Point2D.Float, ConnectionPoint>>
cpIt = currentLane
                                                .getConnectionPoints().entrySet(
).iterator();
                                while (cpIt.hasNext()) {
                                        Iterator<Entry<Lane, Connection>> conIt
= cpIt.next()
                                                        .getValue().getConnectio
ns().entrySet().iterator();
                                        while (conIt.hasNext()) {
                                                conIt.next().getValue().getCarsI
nLane().clear();
                                        }
                                }
                        }
                }
                this.cars.clear();
                for (int i = 0; i < lights.size(); i++) {
                        lights.get(i).reset();
                }
                Car.setCarsCreated(0);
                for (int i = 0; i < parks.size(); i++) {
                        parks.get(i).reset();
                }

        }

        public ArrayList<Lane> getLanes() {
                ArrayList<Lane> laneReturn = new ArrayList<Lane>();
                for (int i = 0; i < this.roads.size(); i++) {
                        Road cr = roads.get(i);
                        Iterator<Entry<Integer, Lane>> lIt = cr.getLanes().entry
Set()
                                        .iterator();
                        while (lIt.hasNext()) {
                                laneReturn.add(lIt.next().getValue());
                        }
                }
```

```
                return laneReturn;
        }

        public void addNewLight(Integer selectedLane, Point point) {

                Lane sl = null;
                for (int i = 0; i < this.getLanes().size(); i++) {
                        if (this.getLanes().get(i).getLaneId() == selectedLane)
{
                                sl = this.getLanes().get(i);
                                break;
                        }
                }
                Point2D.Float closest = getClosestPointOnSegment(sl.getStart(),
                                sl.getEnd(), point);
                TrafficLight tl = new TrafficLight(sl, "green", 5f, 5f, 1f, close
st);
                sl.addTrafficLight(tl);

        }

        public static Point2D.Float getClosestPointOnSegment(Point2D.Float ss,
                        Point2D.Float se, Point p) {
                return getClosestPointOnSegment(ss.x, ss.y, se.x, se.y, p.x, p.y
);
        }

        /**
         * used to find coordinate for the new traffic light From
         * http://www.java2s.com
         * /Code/Java/2D-Graphics-GUI/Returnsclosestpointonsegmenttopoint.htm
         **/
        public static Point2D.Float getClosestPointOnSegment(float sx1, float sy
1,
                        float sx2, float sy2, int px, int py) {
                double xDelta = sx2 - sx1;
                double yDelta = sy2 - sy1;

                if ((xDelta == 0) && (yDelta == 0)) {
                        throw new IllegalArgumentException(
                                        "Segment start equals segment end");
                }

                double u = ((px - sx1) * xDelta + (py - sy1) * yDelta)
                                / (xDelta * xDelta + yDelta * yDelta);

                final Point2D.Float closestPoint;
                if (u < 0) {
                        closestPoint = new Point2D.Float(sx1, sy1);
                } else if (u > 1) {
                        closestPoint = new Point2D.Float(sx2, sy2);
                } else {
                        closestPoint = new Point2D.Float((float) Math.round(sx1
+ u
                                        * xDelta), (float) Math.round(sy1 + u *
yDelta));
                }

                return closestPoint;
        }

}
```

```java
package model;

import java.awt.Graphics;
import java.awt.geom.Point2D;
import java.awt.geom.Point2D.Float;

public class ChangingLane extends Lane {
        // special lane that will be created dynamically when ever a car changes
        //NOT IN USE IN CURRENT VERSION OF THE SYSTEM
        private Lane startLane;
        private Lane targetLane;
        private Point2D.Float startPoint, endPoint;

        public ChangingLane(Lane sl, Lane tl, Road road, Point2D.Float sp) {
                super();
                this.contained=road;
                this.startLane = sl;
                this.targetLane = tl;
                this.startPoint = sp;
                this.endPoint = setUpChangingLane(sl, tl, sp);
        }

        private Float setUpChangingLane(Lane sl, Lane tl, Float sp) {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public float calculateLaneSpan() {
                // TODO Auto-generated method stub
                return 0;
        }

        @Override
        public Float nextPosition(Car car, float targetDistance,
                        float distanceTravelled) {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public Point2D.Float getStart() {
                return this.startPoint;
        }

        @Override
        public Point2D.Float getEnd() {
                return this.endPoint;

        }

        @Override
        public void paint(Graphics g) {
                // TODO Auto-generated method stub

        }

        @Override
        public Road getRoad() {
                // TODO Auto-generated method stub
                return this.contained;
        }

        @Override
        public float findDistance(Car car) {
                // TODO Auto-generated method stub
                return 0;
        }
```

```java
        @Override
        public Car getFrontCar(Car car) {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public TrafficLight getNextTrafficLight(Car car) {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public void paintBorders(Graphics g) {
                // TODO Auto-generated method stub

        }

}
```

```
package tests;

import java.awt.geom.Point2D;
import java.awt.geom.Point2D.Float;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

import model.Car;
import model.Connection;
import model.ConnectionPoint;
import model.Lane;
import model.Road;
import model.StraightRoad;
import model.UnknownConnectionError;

public class ConnectTest {
        public static void main(String[] args) throws InterruptedException {
                Road sr = new StraightRoad(new Point2D.Float(200, 200),
                                new Point2D.Float(400, 200), 3, 1);
                System.out.println(sr.getBilateral());
                Road sr2 = new StraightRoad(new Point2D.Float(300, 1000),
                                new Point2D.Float(300, 200), 1, 0);
                Car car= new Car();
                System.out.println(sr2.getBilateral());
                try {
                        Road.connectLane(sr2, 0, sr, 1);

                        Lane testLane = sr2.getLanes().get(0);
                        System.out.println(testLane);
                        HashMap<Point2D.Float, ConnectionPoint> cpMap = testLane
                                        .getConnectionPoints();
                        Iterator<Entry<Point2D.Float, ConnectionPoint>> it = cpM
ap
                                        .entrySet().iterator();
                        while (it.hasNext()) {
                                Map.Entry pair = (Map.Entry) it.next();
                                System.out.println(pair.getKey() + " = " + pair.g
etValue());
                                ConnectionPoint testCp = (ConnectionPoint) pair.
getValue();
                                HashMap<Lane, Connection> connections = testCp.g
etConnections();
                                Iterator<Entry<Lane, Connection>> it1 = connecti
ons.entrySet()
                                                .iterator();
                                while (it1.hasNext()) {
                                        System.out.println("printing connection");
                                        Map.Entry pair1 = (Map.Entry) it1.next()
;
                                        Connection c1 = (Connection) pair1.getVa
lue();
                                        System.out.println("Next position test:"+ c1.n
extPosition(car, 0.02f, 0.34f));
                                        System.out.println(c1.getStart());
                                        System.out.println(c1.getIntersection())
;
                                        System.out.println(c1.getEnd());
                                }

                        }
                } catch (UnknownConnectionError e) {
                        // TODO Auto-generated catch block
                        System.out.println("error");
                }
                Car c1 = new Car();
                c1.setCurrentSpeed(10);
                c1.enterLane(sr2.getLanes().get(0), sr2.getLanes().get(0).getSta
```

```
rt());
                while (true) {
                        c1.move();
                        Thread.sleep(20);
                }

        }
}
```

```
package model;

import java.awt.Graphics;
import java.awt.RenderingHints;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.awt.geom.Point2D.Float;
import java.math.BigDecimal;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;
//this is quadratic bezier curve that car will move onto while changing
// lane

public class Connection extends Lane {

        private Road sRoad, tRoad;
        private Lane sLane, tLane;
        private Point2D.Float interStartPoint;
        private Point2D.Float interEndPoint;
        private Point2D.Float intersectingPoint;
        private ConnectionPoint cp;
        private float[] bezierDistanceTable = new float[1001];
        private Point2D.Float[] bezierPointTable = new Point2D.Float[1001];

        public Connection() {
                // dummy: used to represent an exit point.

        }

        public Connection(Road sRoad, Lane sLane, Road tRoad, Lane tLane,
                        ConnectionPoint cp, Point2D.Float interStartPoint,
                        Point2D.Float interEndPoint, Point2D.Float intersectingP
oint)
                        throws UnknownConnectionError {
                super();
                this.sRoad = sRoad;
                this.sLane = sLane;
                this.tRoad = tRoad;
                this.tLane = tLane;
                this.cp = cp;
                this.interStartPoint = interStartPoint;
                this.interEndPoint = interEndPoint;
                this.intersectingPoint = intersectingPoint;
                setupBezier(interStartPoint, intersectingPoint, interEndPoint);

        }

        @Override
        public float calculateLaneSpan() {
                // TODO Auto-generated method stub
                return bezierDistanceTable[1000];
        }

        @Override
        public Point2D.Float nextPosition(Car car, float targetDistance,
                        float distanceTravelled) {

                // bezier curve. Only up the table for coordinate according to
                // distanceTravelled+targetDistance;

                float finalDistance = targetDistance + distanceTravelled;

                float fdRound = round(finalDistance).floatValue();

                int index = binarySearch(this.bezierDistanceTable, finalDistance
,
                                targetDistance);
                if (index == -1) {
```

```
                        System.out.println("something went wrong");
                        System.exit(0);
                } else if (index > 1000) {
                        index = 1000;
                }
                car.setTravelled(finalDistance);
                return bezierPointTable[index];
        }

        @Override
        public Point2D.Float getStart() {
                return this.interStartPoint;
        }

        @Override
        public Point2D.Float getEnd() {
                return this.interEndPoint;

        }

        public Point2D.Float getIntersection() {
                return this.intersectingPoint;
        }

        public Road getTargetRoad() {
                return this.tRoad;
        }

        public Lane getTargetLane() {
                return this.tLane;
        }

        public Lane getStartLane() {
                return this.sLane;
        }

        public Road getRoad() {
                return this.sRoad;
        }

        private Point2D.Float calculateBezier(float t) {

                float x = (1 - t) * (1 - t) * this.interStartPoint.x + 2 * (1 -
t) * t
                                * this.intersectingPoint.x + t * t * this.interE
ndPoint.x;
                float y = (1 - t) * (1 - t) * this.interStartPoint.y + 2 * (1 -
t) * t
                                * this.intersectingPoint.y + t * t * this.interE
ndPoint.y;
                return new Point2D.Float(x, y);
        }

        private void setupBezier(Point2D.Float startingPoint,
                        Point2D.Float controlPoint, Point2D.Float endPoint) {
                float t = 0.000f;
                float x;
                float y;
                float cumulativeDistance = 0;
                int i = 0;
                Point2D.Float bezierPoint;
                Point2D.Float currentPoint = startingPoint;
                for (; t <= 1; i++) {

                        x = (1 - t) * (1 - t) * startingPoint.x + 2 * (1 - t) *
t
                                        * controlPoint.x + t * t * endPoint.x;
                        y = (1 - t) * (1 - t) * startingPoint.y + 2 * (1 - t) *
t
```

```
                                * controlPoint.y + t * t * endPoint.y;
                        bezierPoint = new Point2D.Float(x, y);
                        bezierPointTable[i] = bezierPoint; // t for point
                        cumulativeDistance += (float) Math.sqrt(Math.pow(
                                        (bezierPoint.x − currentPoint.x), 2.0)
                                        + Math.pow(bezierPoint.y − currentPoint.
y, 2.0));
                        bezierDistanceTable[i] = cumulativeDistance;
                        currentPoint = bezierPoint;
                        t += 0.001f;
                }
                System.out.println("bezier setup");
                System.out.println("Total connection distance = "
                                + bezierDistanceTable[1000]);

        }

        private static BigDecimal round(float finalDistance) {

                BigDecimal bd = new BigDecimal(finalDistance);
                bd.setScale(4, BigDecimal.ROUND_HALF_UP);
                return bd;

        }

        private int binarySearch(float[] table, float target, float errorBound)
{
                int firstKey = (table.length / 2);
                if (Math.abs(table[firstKey] − target) < errorBound) {

                        return firstKey;
                } else if (table[firstKey] > target) {
                        // look for the values smaller

                        return binarySearch(table, 0, firstKey − 1, target, erro
rBound);
                } else if (table[firstKey] < target) {
                        // look for the values larger

                        return binarySearch(table, firstKey + 1, table.length, t
arget,
                                        errorBound);
                }
                return −1;
        }

        private int binarySearch(float[] table, int lowerBound, int upperBound,
                        float target, float errorBound) {

                int halfPoint = (lowerBound + upperBound) / 2;
                if (lowerBound == upperBound) {
                        return halfPoint;

                }

                else if (Math.abs(table[halfPoint] − target) < errorBound) {

                        return halfPoint;

                } else if (table[halfPoint] > target) {

                        return binarySearch(table, lowerBound, halfPoint − 1, ta
rget,
                                        errorBound);
                } else if (table[halfPoint] < target) {
                        // look for bigger half

                        return binarySearch(table, halfPoint + 1, upperBound, ta
rget,
```

```
                                        errorBound);
                }
                return −1;
        }

        @Override
        public void paint(Graphics g) {

        }

        @Override
        public float findDistance(Car car) {
                // not needed for this class
                return 0;
        }

        @Override
        public Car getFrontCar(Car car) {

                float closest = 100f;
                Car closestCar = null;

                float dtp = Car.distance(this.cp.getPointCoordinate(),
                                this.sLane.getStart());
                // needs to see cars in front of it currently in originating lan
e, cars
                // in current connection
                // as well as cars after the connection

                // in originating lane
                Iterator<Entry<Integer, Car>> olCars = this.sLane.getCarsInLane(
)
                                .entrySet().iterator();
                while (olCars.hasNext()) {
                        Car oCar = olCars.next().getValue();
                        if (oCar.getTravelled() > car.getTravelled() + dtp
                                        && (oCar.getTravelled() − (car.getTravel
led() + dtp)) < 100f) {
                                closest = Car.distance(oCar.getCoordinate(),
                                                car.getCoordinate());
                                closestCar = oCar;
                        }
                }

                // in ending lane
                Iterator<Entry<Integer, Car>> elCars = this.tLane.getCarsInLane(
)
                                .entrySet().iterator();
                while (elCars.hasNext()) {
                        Car eCar = elCars.next().getValue();
                        if (eCar.getTravelled() > Car.distance(new Point2D.Float
(
                                        intersectingPoint.x
                                                        − (interEndPoint.x − int
ersectingPoint.x),
                                        intersectingPoint.y
                                                        − (interEndPoint.y − int
ersectingPoint.y)),
                                        this.tLane.getStart())
                                        && Car.distance(eCar.getCoordinate(), ca
r.getCoordinate()) < 100f
                                        && (Car.distance(eCar.getCoordinate(), t
his.interEndPoint) < Car
                                                        .distance(car.getCoordin
ate(), this.interEndPoint))) {
                                closest = Car.distance(eCar.getCoordinate(),
                                                car.getCoordinate());
                                closestCar = eCar;
                        }
                }
                // in current connection
                Iterator<Entry<Integer, Car>> cit = this.getCarsInLane().entrySe
t()
                                .iterator();

                while (cit.hasNext()) {
                        Map.Entry<Integer, Car> cPair = cit.next();
                        Car cCar = cPair.getValue();

                        if (cCar.getTravelled() > car.getTravelled()) {
                                // cCar is somewhere ahead of the car in the sam
e lane
                                if (((cCar.getTravelled() - car.getTravelled())
< closest)
                                                && !car.equals(cCar)) {
                                        closest = cCar.getTravelled() - car.getT
ravelled();
                                        closestCar = cCar;
                                }
                        }
                }
                // if closest car is null it means the parameter car is the lead
ing
                // car
                return closestCar;
        }

        @Override
        public TrafficLight getNextTrafficLight(Car car) {
                // not needed
                return null;
        }

        @Override
        public void paintBorders(Graphics g) {
                // not needed

        }

}
```

```
package model;

import java.awt.geom.Point2D;
import java.util.ArrayList;
import java.util.HashMap;

//object holding connections
public class ConnectionPoint {
        private HashMap<Lane, Connection> connections = new HashMap<Lane, Connec
tion>();
        private Point2D.Float pointCoordinate;
        private Road road;
        private Lane lane;

        public ConnectionPoint(Road road, Lane lane, Point2D.Float coordinate) {
                this.road = road;
                this.lane = lane;
                this.pointCoordinate = coordinate;

        }

        public Point2D.Float getPointCoordinate() {
                return pointCoordinate;
        }

        public void setPointCoordinate(Point2D.Float pointCoordinate) {
                this.pointCoordinate = pointCoordinate;
        }

        public Road getRoad() {
                return road;
        }

        public Lane getLane() {
                return lane;
        }

        public void addConnection(Connection cn) throws UnknownConnectionError {
                connections.put(cn.getTargetLane(), cn);
        }

        public HashMap<Lane, Connection> getConnections() {
                return connections;
        }
}
```

```
package view;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import control.ParkController;
import control.WorldController;

public class ConsolePanel extends JPanel implements ActionListener {

        private WorldController wController;
        private CarSimView mainFrame;
        private Font font = new Font("Tahoma", Font.BOLD, 20);
        private JButton startButton, stopButton, returnButton, trafficlightButto
n,
                        helpButton, resetButton, addLightBtn;
        private JLabel cntrlPanel_gfx, carSpawnLabel, teamLabel, carSpeedLabel;
        private JPanel bottomButtonPanel, topButtonPanel;
        private JSlider carSpawnSlider = new JSlider(0, 10, 3);
        private BorderLayout borderLayout = new BorderLayout();
        DynamicChart demo;
        private ImageIcon cntrlPanel_img, teamlogo_img, carspeed_img;

        public ConsolePanel(WorldController wController, CarSimView mainFrame,
                        final ParkController pc) {
                this.mainFrame = mainFrame;
                this.wController = wController;
                this.setPreferredSize(new Dimension(200, 800));

                this.setLayout(borderLayout);

                topButtonPanel = new JPanel();
                bottomButtonPanel = new JPanel();
                demo = new DynamicChart("", this.wController);

                Font font = new Font("Tahoma", Font.BOLD, 14);
                carSpawnLabel = new JLabel("Car Spawn Rate");
                carSpawnLabel.setFont(font);
                cntrlPanel_gfx = new JLabel();
                cntrlPanel_img = new ImageIcon(getClass().getResource(
                                File.separator + "gfx" + File.separator + "cntrl_gf
x.gif"));
                cntrlPanel_gfx.setIcon(cntrlPanel_img);
                teamLabel = new JLabel();
                teamlogo_img = new ImageIcon(getClass().getResource(
                                File.separator + "gfx" + File.separator + "teamNE
RD_img.gif"));
                teamLabel.setIcon(teamlogo_img);
                carSpeedLabel = new JLabel();
                carspeed_img = new ImageIcon(getClass().getResource(
                                File.separator + "gfx" + File.separator + "CSC.gif
"));
                carSpeedLabel.setIcon(carspeed_img);

                this.add(topButtonPanel, BorderLayout.NORTH);
                // topButtonPanel
```

```
                this.add(bottomButtonPanel, BorderLayout.SOUTH);

                // JButton setup!
                stopButton = new CustomJButton("Stop");
                stopButton.addActionListener(this);
                startButton = new CustomJButton("Start");
                startButton.addActionListener(this);
                addLightBtn = new CustomJButton("Add Traffic Light");
                addLightBtn.addActionListener(this);
                returnButton = new CustomJButton("Return To Main Menu");
                returnButton.addActionListener(this);
                trafficlightButton = new CustomJButton("Adjust Traffic Lights");
                trafficlightButton.addActionListener(this);
                helpButton = new CustomJButton("Help?");
                helpButton.addActionListener(this);
                resetButton = new CustomJButton("RESET");
                resetButton.addActionListener(this);

                JLabel controlTitleLabel = new JLabel("Control Panel");
                controlTitleLabel.setFont(font);
                JLabel trafficTitleLabel = new JLabel("Adjust Traffic Ligts");

                carSpawnSlider.setPaintTicks(true);
                carSpawnSlider.setPaintLabels(true);
                carSpawnSlider.setMinorTickSpacing(1);
                carSpawnSlider.setMajorTickSpacing(5);
                carSpawnSlider.addChangeListener(new ChangeListener() {

                        @Override
                        public void stateChanged(ChangeEvent e) {

                                JSlider source = (JSlider) e.getSource();
                                // red slider

                                int value = source.getValue();
                                pc.setSpawnChance(value);

                        }

                });

                topButtonPanel.add(cntrlPanel_gfx);
                topButtonPanel.add(stopButton);
                topButtonPanel.add(startButton);
                topButtonPanel.add(resetButton);

                topButtonPanel.add(trafficlightButton);

                topButtonPanel.add(addLightBtn);
                topButtonPanel.add(carSpawnLabel);
                topButtonPanel.add(carSpawnSlider);
                topButtonPanel.add(demo.content);
                topButtonPanel.add(carSpeedLabel);
                bottomButtonPanel.add(returnButton);
                bottomButtonPanel.add(helpButton);
                bottomButtonPanel.add(teamLabel);
                topButtonPanel.setPreferredSize(new Dimension(200, 520));
                bottomButtonPanel.setPreferredSize(new Dimension(200, 100));

                mainFrame.add(this);
        }

        @Override
        public void actionPerformed(ActionEvent e) {
                if (e.getSource() == stopButton) {
                        this.wController.pause();
                } else if (e.getSource() == startButton) {
                        this.wController.start();
                } else if (e.getSource() == returnButton) {
                        this.wController.pause();
                        mainFrame.setAddingLight(false);
                        System.out.println("Main Menu");
                        mainFrame.mainMenu();
                } else if (e.getSource() == trafficlightButton) {
                        System.out.println("Traffic Light Menu");
                        this.wController.pause();
                        mainFrame.TrafficPanel(0);
                } else if (e.getSource() == helpButton) {
                        this.wController.pause();
                        mainFrame.HelpPanel();
                } else if (e.getSource() == resetButton) {
                        // RESET
                        this.wController.reset();
                } else if (e.getSource() == addLightBtn) {
                        if (mainFrame.getAddingLight()) {
                                mainFrame.setAddingLight(false);

                        } else if (!mainFrame.getAddingLight()) {
                                wController.pause();
                                mainFrame.setAddingLight(true);

                        }
                        mainFrame.repaint();
                }
        }

        public DynamicChart getDynamicChart() {
                // TODO Auto-generated method stub
                return this.demo;
        }

        public void enableButtons() {
                // TODO Auto-generated method stub
                startButton.setEnabled(true);
                stopButton.setEnabled(true);
                helpButton.setEnabled(true);
                trafficlightButton.setEnabled(true);
                helpButton.setEnabled(true);
                resetButton.setEnabled(true);
        }

        public void disableButtons() {
                startButton.setEnabled(false);
                stopButton.setEnabled(false);
                helpButton.setEnabled(false);
                trafficlightButton.setEnabled(false);
                helpButton.setEnabled(false);
                resetButton.setEnabled(false);
        }
}
```

```java
package view;

import java.awt.Color;
import java.awt.Font;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.Shape;
import java.awt.geom.RoundRectangle2D;

import javax.swing.ButtonModel;
import javax.swing.JButton;
//The creation of a custom JButton that extends the normal java swing JButton
public class CustomJButton extends JButton {

        private GradientPaint gradientPaint; // provides a way to fill a Shape w
ith

// a linear color gradient pattern
        private Color deafultButtonColour = new Color(255, 255, 255);
        private final Color buttonColour1 = new Color(100, 100, 100);
        private final Color buttonColour2 = new Color(255, 0, 0);

        public CustomJButton(String text) {
                super(text);
                setFont(new Font("Tahoma", Font.BOLD, 12));
                setForeground(Color.WHITE);
                setFocusable(false);
                setContentAreaFilled(false);
                setBorderPainted(false);
        }

        @Override
        public void paint(Graphics g) {
                Graphics2D g2d = (Graphics2D) g.create();
                g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                                RenderingHints.VALUE_ANTIALIAS_ON);
                int height = getHeight();
                int width = getWidth();
                ButtonModel buttonModel = getModel();// The JButton Model

                gradientPaint = new GradientPaint(0, 0, deafultButtonColour, 0,
                                height / 2, buttonColour1, true);

                g2d.setPaint(gradientPaint);
                GradientPaint gradientPainter1;

                if (!buttonModel.isPressed()) {
                        gradientPainter1 = new GradientPaint(0, 0, new Color(100
, 100, 100), 0,
                                        height - 1, new Color(0, 0, 0));
                } else {
                        gradientPaint = new GradientPaint(0, 0, deafultButtonCol
our, 0,
                                        height / 2, buttonColour2, true);
                        g2d.setPaint(gradientPaint);
                        gradientPainter1 = new GradientPaint(0, 0, new Color(0,
0, 0), 0, height - 1,
                                        new Color(100, 100, 100));
                }
                RoundRectangle2D.Float r2d = new RoundRectangle2D.Float(0, 0,
                                width - 1, height - 1, 10, 10);
                Shape clip = g2d.getClip();
                g2d.clip(r2d);
                g2d.fillRect(0, 0, width, height);
                g2d.setClip(clip);
                g2d.setPaint(gradientPainter1);
                g2d.drawRoundRect(0, 0, width - 1, height - 1, 10, 10);
```

```java
                g2d.dispose();
                super.paintComponent(g);

        }
}
```

```java
package view;

import java.awt.BorderLayout;
import java.awt.Color;

import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.ValueAxis;
import org.jfree.chart.plot.XYPlot;
import org.jfree.data.time.Millisecond;
import org.jfree.data.time.TimeSeries;
import org.jfree.data.time.TimeSeriesCollection;
import org.jfree.data.xy.XYDataset;
import org.jfree.ui.ApplicationFrame;

import control.WorldController;

public class DynamicChart extends ApplicationFrame {
        private TimeSeries series;
        private double lastValue = 100.0;
        public JPanel content;
        private WorldController wc;

        public DynamicChart(final String title, WorldController wc) {

                super(title);
                this.series = new TimeSeries("", Millisecond.class);
                this.wc = wc;

                final TimeSeriesCollection dataset = new TimeSeriesCollection(
                                this.series);
                final JFreeChart chart = createChart(dataset);

                // timer.setInitialDelay(20);

                // Sets background color of chart
                chart.setBackgroundPaint(Color.LIGHT_GRAY);

                // Created JPanel to show graph on screen
                content = new JPanel(new BorderLayout());

                // Created Chartpanel for chart area
                final ChartPanel chartPanel = new ChartPanel(chart);

                // Added chartpanel to main panel
                content.add(chartPanel);

                // Sets the size of whole window (JPanel)
                chartPanel.setPreferredSize(new java.awt.Dimension(200, 140));

                // Puts the whole content on a Frame
                // setContentPane(content);

                // timer.start();

        }

        public void updateData() {

                this.lastValue = this.wc.getAverageSpeed();

                final Millisecond now = new Millisecond();
                this.series.addOrUpdate(new Millisecond(), this.lastValue);

        }
```

```java
        private JFreeChart createChart(final XYDataset dataset) {
                final JFreeChart result = ChartFactory.createTimeSeriesChart(
                                "Average Car Speed", "Time", "Speed", dataset, false,
 true, false);

                final XYPlot plot = result.getXYPlot();

                plot.setBackgroundPaint(new Color(255,255,255));
                plot.setDomainGridlinesVisible(true);
                plot.setDomainGridlinePaint(Color.lightGray);
                plot.setRangeGridlinesVisible(true);
                plot.setRangeGridlinePaint(Color.lightGray);


                ValueAxis xaxis = plot.getDomainAxis();
                xaxis.setAutoRange(true);

                // Domain axis would show data of 60 seconds for a time
                xaxis.setFixedAutoRange(3000.0); // 60 seconds
                // xaxis.setVerticalTickLabels(true);

                ValueAxis yaxis = plot.getRangeAxis();
                yaxis.setRange(0.0, 200.0);

                return result;
        }
}
```

```
package model;

import java.awt.Graphics;
import java.awt.RenderingHints;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.awt.geom.Point2D.Float;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

import javax.swing.JOptionPane;

//abstract class that all types of lanes extends
public abstract class Lane {

        private static final float SMALL_NUM = (float) 0.00000001;
        private int laneId; // its id
        private int laneKey;
        protected Road contained;
        protected float laneSpan;
        private HashMap<Point2D.Float, ConnectionPoint> connectionPoints = new H
ashMap<Point2D.Float, ConnectionPoint>();
        private HashMap<Integer, Car> carsInLane = new HashMap<Integer, Car>();
        private boolean hasPark = false;
        private CarWorld world;
        protected Point2D.Float startPoint;
        protected Point2D.Float endPoint;
        private boolean isEnding = false;
        private static int lanesCreated = 0;

        // store traffic lights that belong to this lane.
        ArrayList<TrafficLight> trafficLights = new ArrayList<TrafficLight>();

        public Lane() {
                // test stub
        }

        public Lane(Point2D.Float start, Point2D.Float end, Road cRoad,
                        CarWorld cWorld, int lk) {
                this.startPoint = start;
                this.endPoint = end;
                this.contained = cRoad;
                this.laneId = lanesCreated;
                this.world = cWorld;
                lanesCreated++;
                this.laneKey = lk;

        }

        public abstract float calculateLaneSpan();

        public abstract Point2D.Float nextPosition(Car car, float targetDistance
,
                        float distanceTravelled);

        public float getLaneSpan() {
                return laneSpan;
        }

        public int getLaneId() {
                return this.laneId;
        }

        public abstract Road getRoad();

        public final void setLaneSpan(float laneSpan) {
```

```
                this.laneSpan = laneSpan;
        }

        // in case of roundabout this will return center point
        public Point2D.Float getStart() {
                return this.startPoint;
        }

        // in case of roundabout this will return center point
        public Point2D.Float getEnd() {

                return this.endPoint;
        }

        public void addTrafficLight(TrafficLight light) {

                ArrayList<TrafficLight> lights = light.getLane().trafficLights;
                for (int i = 0; i < lights.size(); i++) {
                        TrafficLight currentLight = lights.get(i);
                        if (Car.distance(currentLight.getCoordinate(),
                                        light.getCoordinate()) < 20) {
                                JOptionPane.showMessageDialog(null,
                                                "the light is too close to other lights in lane")
;
                                return;
                        }
                }
                this.trafficLights.add(light);
                this.world.addLight(light);

        }

        public void removeTrafficLight(int lid) {

                for (int i = 0; i < trafficLights.size(); i++) {
                        if (trafficLights.get(i).getId() == lid) {

                                for (int j = 0; j < this.world.getLights().size(
); j++) {
                                        TrafficLight jr = this.world.getLights()
.get(j);
                                        if (jr.getId() == lid) {
                                                this.world.getLights().remove(j)
;
                                                break;
                                        }
                                }
                                trafficLights.remove(i);
                                break;

                        }
                }
        }

        public abstract TrafficLight getNextTrafficLight(Car car);

        public float getSpan() {
                // TODO Auto-generated method stub
                return laneSpan;
        }

        public void addCar(Car car) {
                int carId = car.getId();
                carsInLane.put(carId, car);
        }

        public void removeCar(Car car) {
                int carId = car.getId();
                carsInLane.remove(carId);
        }

        public boolean addConnectionPoint(ConnectionPoint cp, Connection cn)
                        throws UnknownConnectionError {

                if (connectionPoints.get(new Point2D.Float((int) cp
                                .getPointCoordinate().x, (int) cp.getPointCoordi
nate().y)) == null) {

                        System.out
                                        .println("Connection point not found, adding new con
nection point");
                        cp.addConnection(cn);
                        connectionPoints.put(
                                        new Point2D.Float(((int) cp.getPointCoor
dinate().x),
                                                        ((int) cp.getPointCoordi
nate().y)), cp);
                        TrafficLight tl = new TrafficLight(cp.getLane(), "green",
 5f, 5f,
                                        1f, cp.getPointCoordinate());
                        cp.getLane().addTrafficLight(tl);

                        return true;
                } else {
                        System.out
                                        .println("connection point found, adding new connect
ion to the existing connection point");
                        ConnectionPoint existingCp = connectionPoints
                                        .get(new Point2D.Float((int) cp.getPoint
Coordinate().x,
                                                        (int) cp.getPointCoordin
ate().y));
                        existingCp.addConnection(cn);
                        return true;
                }
        }

        public HashMap<Point2D.Float, ConnectionPoint> getConnectionPoints() {
                return this.connectionPoints;
        }

        public abstract void paint(Graphics g);

        public void paintTrafficLights(Graphics g) {
                for (int i = 0; i < this.trafficLights.size(); i++) {
                        trafficLights.get(i).paint(g);
                }
        }

        public void carEnters(Car car) {
                // TODO Auto-generated method stub
                int carId = car.getId();
                carsInLane.put(carId, car);
        }

        public void carLeaves(Car car) {
                carsInLane.remove(car.getId());
        }

        public abstract Car getFrontCar(Car car);

        public int getLaneKey() {
                return this.laneKey;
        }

        // returns all the connections in the same direction lane
        public ArrayList<Connection> getSameConnections() {
```

```
                ArrayList<Connection> cal = new ArrayList<Connection>();
                Iterator<Entry<Integer, Lane>> lit = this.contained.getLanes()
                                .entrySet().iterator();
                while (lit.hasNext()) {
                        Map.Entry<Integer, Lane> lp = lit.next();
                        Lane currentLane = lp.getValue();
                        if (currentLane.laneKey % 2 == this.laneKey % 2) {
                                Iterator<Entry<Point2D.Float, ConnectionPoint>>
cit = currentLane
                                                .getConnectionPoints().entrySet(
).iterator();
                                while (cit.hasNext()) {
                                        Map.Entry<Point2D.Float, ConnectionPoint
> cp = cit.next();
                                        ConnectionPoint currentPoint = cp.getVal
ue();
                                        Iterator<Entry<Lane, Connection>> conIt
= currentPoint
                                                        .getConnections().entryS
et().iterator();
                                        while (conIt.hasNext()) {
                                                Map.Entry<Lane, Connection> conP
 = conIt.next();
                                                Connection currentConnection = c
onP.getValue();
                                                cal.add(currentConnection);
                                        }
                                }
                        }
                }
                return cal;

        }

        public void setHasPark(boolean b) {
                this.hasPark = b;
        }

        public boolean getHasPark() {
                return this.hasPark;
        }

        public abstract float findDistance(Car car);

        public void setEnding(boolean b) {
                this.isEnding = b;
        }

        public boolean isEnding() {
                return this.isEnding;
        }

        // return the list of the lanes of the same direction that are in the sa
me
        // road
        public ArrayList<Lane> getSameLanes() {
                int currentKey = this.getLaneKey();
                ArrayList<Lane> sal = new ArrayList<Lane>();
                Iterator<Entry<Integer, Lane>> lit = this.contained.getLanes()
                                .entrySet().iterator();
                while (lit.hasNext()) {
                        Map.Entry<Integer, Lane> lp = lit.next();
                        Lane currentLane = lp.getValue();
                        if (currentKey % 2 == this.laneKey % 2) {
                                sal.add(currentLane);
                        }
                }
                return sal;
        }
        public abstract void paintBorders(Graphics g);

        public HashMap<Integer, Car> getCarsInLane() {
                // TODO Auto−generated method stub
                return this.carsInLane;
        }
}
```

```
package view;

public class LightButton extends CustomJButton {

        int lightId;

        public LightButton(String text, int id) {
                super(text);
                // TODO Auto−generated constructor stub
                this.lightId = id;
        }

        public int getId() {
                return this.lightId;
        }
}
```

```java
package control;

import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import model.CarWorld;
import view.CarSimView;

public class Main {
        public static void main(String[] args) {

                WorldController wControl = new WorldController();
                CarWorld cWorld = wControl.createWorld();
                TrafficLightController tlc = new TrafficLightController(cWorld);
                ParkController pc= new ParkController(cWorld);
                JFrame frame = new CarSimView("carSim", wControl, tlc, pc);
                wControl.setView(frame);
                tlc.setView(frame);

                try {
                        wControl.simulate();
                        ((CarSimView) frame).mainMenu();
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        System.out.println("simulation error");
                        e.printStackTrace();
                }

        }

}
```

```
package view;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;

import control.WorldController;

public class MainMenu extends JPanel implements ActionListener {
        JButton tButton, interButton, fullSimulation;
        WorldController wControl;
        CarSimView mainFrame;
        private JPanel buttonJPanel;
        private ImageIcon title_img;

        public MainMenu(WorldController wControl, CarSimView mainFrame) {
                this.wControl = wControl;
                this.mainFrame = mainFrame;
                buttonJPanel = new JPanel();

                tButton = new CustomJButton("T Junction");
                tButton.addActionListener(this);
                interButton = new CustomJButton("Intersection");
                interButton.addActionListener(this);
                fullSimulation = new CustomJButton("Full Simulation");
                fullSimulation.addActionListener(this);

                title_img = new ImageIcon(getClass().getResource(File.separator
+ "gfx"
                                + File.separator + "title_gfx.gif"));
                JLabel titleSimulationTitle = new JLabel();
                titleSimulationTitle.setIcon(title_img);
                this.add(titleSimulationTitle);

                this.setBorder(BorderFactory.createMatteBorder(3, 3, 3, 3, Color
.BLACK));
                buttonJPanel.setBorder(BorderFactory.createMatteBorder(2, 2, 2,
2,
                                Color.BLACK));
                buttonJPanel.setBackground(Color.WHITE);
                buttonJPanel.setPreferredSize(new Dimension(400, 50));
                this.add(buttonJPanel);

                buttonJPanel.add(tButton);
                buttonJPanel.add(interButton);
                buttonJPanel.add(fullSimulation);

        }

        @Override
        public void actionPerformed(ActionEvent e) {
                // TODO Auto-generated method stub
                if (e.getSource().equals(tButton)) { // T Junction choosen by us
er
                        wControl.setTJunction();
                        mainFrame.simulationView();

                }else if (e.getSource().equals(interButton)) { // Intersection c
hooen

                        // by user
```

```
                        wControl.setIntersection();
                        mainFrame.simulationView();
                        // If there views set up for intersecton
                } else if (e.getSource().equals(fullSimulation)) { // Full simul
ation

                                // chosen by user.
                        wControl.setFullSimulation();
                        mainFrame.simulationView();
                }
        }
}
```

```
package control;

import java.util.ArrayList;

import model.CarPark;
import model.CarWorld;

public class ParkController {
        private CarWorld world;

        public ParkController(CarWorld world) {
                this.world = world;
        }

        public void setSpawnChance(int spawn) {
                ArrayList<CarPark> parks = world.getParks();
                for (int i = 0; i < parks.size(); i++) {
                        parks.get(i).setSpawn(spawn);
                }
        }
}
```

```
package model;

import java.awt.Rectangle;
import java.util.ArrayList;
import java.util.List;
//collision checking
//origially from
/*http://gamedevelopment.tutsplus.com/tutorials/
 * quick-tip-use-quadtrees-to-detect-likely-collisions-in-2d-space--gamedev-374*
/

public class QuadTree {
        private int MAX_OBJECTS = 8;
        private int MAX_LEVELS = 5;

        private int level;
        private ArrayList<Car> cars;
        private Rectangle bounds;
        private QuadTree[] nodes;

        /*
         * Constructor
         */
        public QuadTree(int pLevel, Rectangle pBounds) {
                level = pLevel;
                cars = new ArrayList<Car>();
                bounds = pBounds;
                nodes = new QuadTree[4];
        }

        public void clear() {
                cars.clear();

                for (int i = 0; i < nodes.length; i++) {
                        if (nodes[i] != null) {
                                nodes[i].clear();
                                nodes[i] = null;
                        }
                }
        }

        private void split() {
                int subWidth = (int) (bounds.getWidth() / 2);
                int subHeight = (int) (bounds.getHeight() / 2);
                int x = (int) bounds.getX();
                int y = (int) bounds.getY();

                nodes[0] = new QuadTree(level + 1, new Rectangle(x + subWidth, y
,
                                subWidth, subHeight));
                nodes[1] = new QuadTree(level + 1, new Rectangle(x, y, subWidth,
                                subHeight));
                nodes[2] = new QuadTree(level + 1, new Rectangle(x, y + subHeigh
t,
                                subWidth, subHeight));
                nodes[3] = new QuadTree(level + 1, new Rectangle(x + subWidth, y
                                + subHeight, subWidth, subHeight));
        }

        private int getIndex(Car car) {
                int index = -1;
                double verticalMidpoint = bounds.getX() + (bounds.getWidth() / 2
);
                double horizontalMidpoint = bounds.getY() + (bounds.getHeight()
/ 2);

                // Object can completely fit within the top quadrants
                boolean topQuadrant = (car.getCoordinate().getY() <= horizontalM
idpoint);
```

```
                // Object can completely fit within the bottom quadrants
                boolean bottomQuadrant = (car.getCoordinate().getY() > horizonta
lMidpoint);

                // Object can completely fit within the left quadrants
                if (car.getCoordinate().getX() <= verticalMidpoint) {
                        if (topQuadrant) {
                                index = 1;
                        } else if (bottomQuadrant) {
                                index = 2;
                        }
                }
                // Object can completely fit within the right quadrants
                else if (car.getCoordinate().getX() > verticalMidpoint) {
                        if (topQuadrant) {
                                index = 0;
                        } else if (bottomQuadrant) {
                                index = 3;
                        }
                }

                return index;
        }

        public void insert(Car car) {
                if (nodes[0] != null) {
                        int index = getIndex(car);

                        if (index != -1) {
                                nodes[index].insert(car);

                                return;
                        }
                }

                cars.add(car);

                if (cars.size() > MAX_OBJECTS && level < MAX_LEVELS) {
                        if (nodes[0] == null) {
                                split();
                        }

                        int i = 0;
                        while (i < cars.size()) {
                                int index = getIndex(cars.get(i));
                                if (index != -1) {
                                        nodes[index].insert(cars.remove(i));
                                } else {
                                        i++;
                                }
                        }
                }
        }

        public ArrayList<Car> retrieve(ArrayList<Car> returnObjects, Car car) {
                int index = getIndex(car);
                if (index != -1 && nodes[0] != null) {
                        nodes[index].retrieve(returnObjects, car);
                }

                returnObjects.addAll(cars);

                return returnObjects;
        }
}
```

```
package model;

import java.awt.Graphics;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.awt.geom.Point2D.Float;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

public abstract class Road {
        // knows which lane is connecting to which road, pair of (lane number, a
nd
        // the road it connects to).

        // some logical constrictions
        // base lane decides the type of the lanes that can be added

        private int roadId;
        private final int roadType;
        public final static int roadWidth = 20; // each lanes are 20 pixels wide
        private static int roadsCreated = 0;
        public static final int STRAIGHT_LANE = 0;
        public static final int ROUNDABOUT = 1;
        public static final int CURVE = 2;
        private HashMap<Integer, Lane> lanes = new HashMap<Integer, Lane>();
        private CarWorld world;

        // bilateral vs unilateral state
        // definition of connection: contiguous legal movements possible
        private boolean bilateral;

        // used for 90 degrees rotation
        private static final int[][] perpenMat = new int[][] { { 0, −1 }, { 1, 0
 } };

        public Road(int roadType, CarWorld cWorld) {
                this.roadId = roadsCreated;
                this.world = cWorld;
                roadsCreated++;
                this.roadType = roadType;
        }

        public void paint(Graphics g) {
                Iterator<Entry<Integer, Lane>> it = lanes.entrySet().iterator();
                while (it.hasNext()) {
                        Map.Entry pair = (Map.Entry) it.next();
                        Lane currentLane = (Lane) pair.getValue();
                        currentLane.paint(g);

                }

        }

        /**** road type dependent ****/

        // contaaining all lanes

        /***********************/

        /***** road type independent *******/
        public int getId() {
                return this.roadId;

        }
```

```

        public HashMap<Integer, Lane> getLanes() {
                return lanes;// should return an arraylist
        }

        public void update() {
                // update all lanes belonging to this road
        }

        public int getType() {
                return this.roadType;
        }

        public boolean getBilateral() {
                return this.bilateral;
        }

        protected void setBilateral(boolean bl) {
                this.bilateral = bl;
        }

        // currently only supports connections between straight lanes
        public static void connectLane(Road currentRoad, int currentLane,
                        Road targetRoad, int targetLane) throws UnknownConnectio
nError {

                if ((currentRoad instanceof StraightRoad && targetRoad instanceo
f StraightRoad)) {
                        lineLineConnect(currentRoad, currentLane, targetRoad, ta
rgetLane);

                } else if ((currentRoad instanceof StraightRoad && targetRoad in
stanceof RoundRoad)) {

                } else if (currentRoad instanceof RoundRoad
                                && targetRoad instanceof StraightRoad) {

                } else {
                        throw new UnknownConnectionError();
                }
                // sets connection state between the lane corresponding to the
                // currentLane in currentRoad and the corresponding targetLane,
in
                // targetRoad

        }

        // lane connection logic between straight lines
        private static boolean lineLineConnect(Road currentRoad, int currentLane
,
                        Road targetRoad, int targetLane) throws UnknownConnectio
nError {
                System.out.println("Start connecting");
                HashMap<Integer, Lane> sLanes = currentRoad.getLanes();
                HashMap<Integer, Lane> tLanes = targetRoad.getLanes();
                Point2D.Float intersectingStartPoint;
                Point2D.Float intersectingControlPoint;
                Point2D.Float intersectingEndPoint;
                boolean success = false;
                boolean overlapFound = false;
                // checks if the roads actually has the lanes that are needed to
 be
                // connected
                if (sLanes.containsKey(currentLane) && tLanes.containsKey(target
Lane)) {
                        System.out.println("Lanes in the roads found");
                        // get two most outer roads−> get most outer boundaries
of the two
                        // roads
                        // check if the closest outer boundary intersects with t
he
                        // boundaries of the other road.
                        // if it doesn’t check if it works other way.
                        // case where they are all even or all odd−>
                        // for one of the roads a1−a2, a3−a4
                        Point2D.Float a1;
                        Point2D.Float a2;
                        Point2D.Float a3;
                        Point2D.Float a4;

                        // for another road b1−b2, b3−b4
                        Point2D.Float b1;
                        Point2D.Float b2;
                        Point2D.Float b3;
                        Point2D.Float b4;

                        // for sLanes
                        ArrayList<Integer> sortedS = new ArrayList<Integer>(sLan
es.keySet());
                        Collections.sort(sortedS);
                        int highest = sortedS.get(sortedS.size() − 1);
                        int modulus = highest % 2;
                        int nextHighest = −1;

                        // ////////
                        // ////////
                        // start if
                        if (currentRoad.getBilateral()) {
                                // bilateral get one lane with highest even key
and one with
                                // highest odd key

                                // start checking for nextHighest
                                System.out.println("starting road is bilateral");
                                if (modulus == 0) {
                                        System.out.println("Starting highest==even");
                                        // highest is even−> next highest should
 be odd since
                                        // bilateral
                                        for (int i = sortedS.size() − 2; i >= 0;
 i−−) {
                                                if (sortedS.get(i) % 2 == 1) {
                                                        nextHighest = sortedS.ge
t(i);
                                                        break;
                                                }
                                        }
                                        // highest is even the outer line is ano
ther addition of the
                                        // half offset vector
                                        // nextHighest is odd the outer line is
another subtraction
                                        // of the half offset vector

                                        Point2D.Float base1 = sLanes.get(highest
).getStart();
                                        Point2D.Float base2 = sLanes.get(highest
).getEnd();
                                        Point2D.Float halfVector = getHalf(base1
, base2);

                                        Lane h = sLanes.get(highest);
                                        Lane nh = sLanes.get(nextHighest);
                                        a1 = new Point2D.Float(h.getStart().x +
halfVector.x,
                                                        h.getStart().y + halfVec
tor.y);
                                        a2 = new Point2D.Float(h.getEnd().x + ha
```

```
lfVector.x,
                                                        h.getEnd().y + halfVecto
r.y);
                                        a3 = new Point2D.Float(nh.getStart().x −
 halfVector.x,
                                                        nh.getStart().y − halfVe
ctor.y);
                                        a4 = new Point2D.Float(nh.getEnd().x − h
alfVector.x,
                                                        nh.getEnd().y − halfVect
or.y);

                                } else {
                                        // highest is odd−> next highest should
be even since
                                        // bilateral
                                        System.out.println("Starting highest== odd");
                                        for (int i = sortedS.size() − 2; i >= 0;
 i−−) {
                                                if (sortedS.get(i) % 2 == 0) {
                                                        nextHighest = sortedS.ge
t(i);
                                                        break;
                                                }
                                        }
                                        // highest is odd the outer line is anot
her subtraction of
                                        // the half offset vector
                                        // nextHighest is even the outer line is
 another addition
                                        // of the half offset vector
                                        Point2D.Float base1 = sLanes.get(nextHig
hest).getStart();
                                        Point2D.Float base2 = sLanes.get(nextHig
hest).getEnd();
                                        Point2D.Float halfVector = getHalf(base1
, base2);

                                        Lane h = sLanes.get(highest);
                                        Lane nh = sLanes.get(nextHighest);
                                        a1 = new Point2D.Float(h.getStart().x −
halfVector.x,
                                                        h.getStart().y − halfVec
tor.y);
                                        a2 = new Point2D.Float(h.getEnd().x − ha
lfVector.x,
                                                        h.getEnd().y − halfVecto
r.y);
                                        a3 = new Point2D.Float(nh.getStart().x +
 halfVector.x,
                                                        nh.getStart().y + halfVe
ctor.y);
                                        a4 = new Point2D.Float(nh.getEnd().x + h
alfVector.x,
                                                        nh.getEnd().y + halfVect
or.y);
                                }
                                // end of bilateral sLane.

                        } else {
                                // unilateral sLAne−>get a lane with highest key
 and one with
                                // the lowest key

                                nextHighest = sortedS.get(0);
                                if (nextHighest % 2 == 0) {
                                        System.out.println("Starting lane is even unilatera
l");
                                        // even lanes
                                        // one outer line is highest + offset
                                        // one outer line is nextHighest - offse
t
                                        Point2D.Float base1 = sLanes.get(nextHig
hest).getStart();
                                        Point2D.Float base2 = sLanes.get(nextHig
hest).getEnd();
                                        Point2D.Float halfVector = getHalf(base1
, base2);
                                        Lane h = sLanes.get(highest);
                                        Lane nh = sLanes.get(nextHighest);
                                        a1 = new Point2D.Float(h.getStart().x +
halfVector.x,
                                                        h.getStart().y + halfVec
tor.y);
                                        a2 = new Point2D.Float(h.getEnd().x + ha
lfVector.x,
                                                        h.getEnd().y + halfVecto
r.y);
                                        a3 = new Point2D.Float(nh.getStart().x -
 halfVector.x,
                                                        nh.getStart().y - halfVe
ctor.y);
                                        a4 = new Point2D.Float(nh.getEnd().x - h
alfVector.x,
                                                        nh.getEnd().y - halfVect
or.y);
                                } else {
                                        System.out.println("Starting lane is odd unilateral
");
                                        // odd lanes
                                        // one outer line is highest - offset
                                        // one outer line is nextHighest + offse
t
                                        Point2D.Float base1 = sLanes.get(nextHig
hest).getEnd();
                                        Point2D.Float base2 = sLanes.get(nextHig
hest).getStart();
                                        Point2D.Float halfVector = getHalf(base1
, base2);
                                        Lane h = sLanes.get(highest);
                                        Lane nh = sLanes.get(nextHighest);
                                        a1 = new Point2D.Float(h.getStart().x -
halfVector.x,
                                                        h.getStart().y - halfVec
tor.y);
                                        a2 = new Point2D.Float(h.getEnd().x - ha
lfVector.x,
                                                        h.getEnd().y - halfVecto
r.y);
                                        a3 = new Point2D.Float(nh.getStart().x +
 halfVector.x,
                                                        nh.getStart().y + halfVe
ctor.y);
                                        a4 = new Point2D.Float(nh.getEnd().x + h
alfVector.x,
                                                        nh.getEnd().y + halfVect
or.y);
                                }

                        }
                        if (nextHighest == -1)
                                throw new UnknownConnectionError();
                        // END for sLane
                        // //////////////
                        // //////////////

                        // for targetRoad
                        ArrayList<Integer> sortedT = new ArrayList<Integer>(tLan
```

```
es.keySet());
                        Collections.sort(sortedT);
                        int tHighest = sortedT.get(sortedT.size() - 1);

                        int tModulus = tHighest % 2;
                        int tNextHighest = -1;
                        if (targetRoad.getBilateral()) {
                                // bilateral get one lane with highest even key
and one with
                                // highest odd key

                                if (tModulus == 0) {
                                        // highest is even-> next highest should
 be odd since
                                        // bilateral

                                        for (int i = sortedT.size() - 2; i >= 0;
 i--) {

                                                if (sortedT.get(i) % 2 == 1) {
                                                        tNextHighest = sortedT.g
et(i);
                                                        break;
                                                }
                                        }

                                        // highest is even the outer line is ano
ther addition of the
                                        // half offset vector
                                        // nextHighest is odd the outer line is
another subtraction
                                        // of the half offset vector
                                        Point2D.Float base1 = tLanes.get(tHighes
t).getStart();
                                        Point2D.Float base2 = tLanes.get(tHighes
t).getEnd();
                                        Point2D.Float halfVector = getHalf(base1
, base2);

                                        Lane h = tLanes.get(tHighest);
                                        Lane nh = tLanes.get(tNextHighest);
                                        b1 = new Point2D.Float(h.getStart().x +
halfVector.x,
                                                        h.getStart().y + halfVec
tor.y);
                                        b2 = new Point2D.Float(h.getEnd().x + ha
lfVector.x,
                                                        h.getEnd().y + halfVecto
r.y);
                                        b3 = new Point2D.Float(nh.getStart().x -
 halfVector.x,
                                                        nh.getStart().y - halfVe
ctor.y);
                                        b4 = new Point2D.Float(nh.getEnd().x - h
alfVector.x,
                                                        nh.getEnd().y - halfVect
or.y);
                                } else {
                                        // highest is odd-> next highest should
be even since
                                        // bilateral
                                        for (int i = sortedT.size() - 2; i >= 0;
 i--) {
                                                if (sortedT.get(i) % 2 == 0) {
                                                        tNextHighest = sortedT.g
et(i);
                                                        break;
                                                }
                                        }
                                        // highest is odd the outer line is anot
her subtraction of
                                        // the half offset vector
                                        // nextHighest is even the outer line is
 another addition
                                        // of the half offset vector
                                        Point2D.Float base1 = tLanes.get(tNextHi
ghest).getStart();
                                        Point2D.Float base2 = tLanes.get(tNextHi
ghest).getEnd();
                                        Point2D.Float halfVector = getHalf(base1
, base2);

                                        Lane h = tLanes.get(tHighest);
                                        Lane nh = tLanes.get(tNextHighest);
                                        b1 = new Point2D.Float(h.getStart().x -
halfVector.x,
                                                        h.getStart().y - halfVec
tor.y);
                                        b2 = new Point2D.Float(h.getEnd().x - ha
lfVector.x,
                                                        h.getEnd().y - halfVecto
r.y);
                                        b3 = new Point2D.Float(nh.getStart().x +
 halfVector.x,
                                                        nh.getStart().y + halfVe
ctor.y);
                                        b4 = new Point2D.Float(nh.getEnd().x + h
alfVector.x,
                                                        nh.getEnd().y + halfVect
or.y);

                                }
                        } else {
                                // unilateral get a lane with highest key and on
e with the
                                // lowest key
                                tNextHighest = sortedT.get(0);
                                if (tNextHighest % 2 == 0) {
                                        // even lanes
                                        // one outer line is highest + offset
                                        // one outer line is nextHighest - offse
t
                                        Point2D.Float base1 = tLanes.get(tNextHi
ghest).getStart();
                                        Point2D.Float base2 = tLanes.get(tNextHi
ghest).getEnd();
                                        Point2D.Float halfVector = getHalf(base1
, base2);
                                        Lane h = tLanes.get(tHighest);
                                        Lane nh = tLanes.get(tNextHighest);
                                        b1 = new Point2D.Float(h.getStart().x +
halfVector.x,
                                                        h.getStart().y + halfVec
tor.y);
                                        b2 = new Point2D.Float(h.getEnd().x + ha
lfVector.x,
                                                        h.getEnd().y + halfVecto
r.y);
                                        b3 = new Point2D.Float(nh.getStart().x -
 halfVector.x,
                                                        nh.getStart().y - halfVe
ctor.y);
                                        b4 = new Point2D.Float(nh.getEnd().x - h
alfVector.x,
                                                        nh.getEnd().y - halfVect
or.y);
                                } else {
                                        // odd lanes
```

```
                                        // one outer line is highest - offset
                                        // one outer line is nextHighest + offse
t
                                        Lane h = tLanes.get(tHighest);
                                        Lane nh = tLanes.get(tNextHighest);
                                        Point2D.Float base1 = nh.getEnd();
                                        Point2D.Float base2 = nh.getStart();
                                        Point2D.Float halfVector = getHalf(base1
, base2);

                                        b1 = new Point2D.Float(h.getStart().x -
halfVector.x,
                                                        h.getStart().y - halfVec
tor.y);
                                        b2 = new Point2D.Float(h.getEnd().x - ha
lfVector.x,
                                                        h.getEnd().y - halfVecto
r.y);
                                        b3 = new Point2D.Float(nh.getStart().x +
 halfVector.x,
                                                        nh.getStart().y + halfVe
ctor.y);
                                        b4 = new Point2D.Float(nh.getEnd().x + h
alfVector.x,
                                                        nh.getEnd().y + halfVect
or.y);
                                }

                        }
                        // now see if a1-a2, a3-a4 intersects with b1-b2 or b3-b
4
                        // or if b1-b2 and b3-b4 overlaps with a1-a2 or a3-a4

                        Line2D.Float line1 = new Line2D.Float(a1, a2);
                        Line2D.Float line2 = new Line2D.Float(a3, a4);
                        Line2D.Float line3 = new Line2D.Float(b1, b2);
                        Line2D.Float line4 = new Line2D.Float(b3, b4);

                        // if the line overlaps check if the overlapped segment
is closest
                        // to the starting lane's starting point

                        if ((line1.intersectsLine(line3) && line2.intersectsLine
(line3))) {

                                overlapFound = true;

                                // check distance between the starting point of
the starting
                                // lane and line3, as well as the distance betwe
en the same
                                // point and line4. If line 3 is the closest int
ersection
                                // between this line is the starting point

                        } else if ((line1.intersectsLine(line4) && line2
                                        .intersectsLine(line4))) {

                                overlapFound = true;

                        } else if ((line3.intersectsLine(line1) && line4
                                        .intersectsLine(line1))) {

                                overlapFound = true;

                        } else if ((line3.intersectsLine(line2) && line4
                                        .intersectsLine(line2))) {

                                overlapFound = true;
                        } else {
                                System.out.println("overlapping not found");
                                return false;
                        }

                        if (overlapFound) {
                                System.out
                                                .println("Finding closest outerline of the t
arget road");
                                // get starting lane's outer lines
                                // find two intersection point with the closest
road outerline
                                // of the target road
                                // get the vector that is perpendicular to the o
uter line that
                                // previously found intersection point lines on
                                // get intersection of the previously found vect
or and the
                                // starting lane's vector-> starting point of th
e connection

                                // get intersection between the starting lane ve
ctor and target
                                // lane vector-> control point of the quadratic
bezier

                                // get a point that lies on target lane's vector
 with the
                                // distance of the displacement between the two
previously found
                                // points-> end point of the quadratic bezier
                                Lane startingL = sLanes.get(currentLane);
                                Lane endingL = tLanes.get(targetLane);
                                Point2D.Float slStarting = startingL.getStart();
                                Point2D.Float slEnding = startingL.getEnd();
                                Point2D.Float halfVector = getHalf(slStarting, s
lEnding);
                                Point2D.Float sla1;
                                Point2D.Float sla2;
                                Point2D.Float sla3;
                                Point2D.Float sla4;
                                sla1 = new Point2D.Float(slStarting.x + halfVect
or.x,
                                                slStarting.y + halfVector.y);
                                sla2 = new Point2D.Float(slEnding.x + halfVector
.x, slEnding.y
                                                + halfVector.y);
                                sla3 = new Point2D.Float(slStarting.x - halfVect
or.x,
                                                slStarting.y - halfVector.y);
                                sla4 = new Point2D.Float(slEnding.x - halfVector
.x, slEnding.y
                                                - halfVector.y);

                                Line2D.Float sLine1 = new Line2D.Float(sla1, sla
2);
                                Line2D.Float sLine2 = new Line2D.Float(sla3, sla
4);

                                Point2D.Float i1 = linesolver.checkIntersection(
slStarting,
                                                slEnding, (Point2D.Float) line3.
getP1(),
                                                (Point2D.Float) line3.getP2());
                                Point2D.Float i2 = linesolver.checkIntersection(
slStarting,
                                                slEnding, (Point2D.Float) line4.
getP1(),
```

```
                                                (Point2D.Float) line4.getP2());

                                float d1 = (float) Math.sqrt(Math.pow((slStartin
g.x - i1.x),
                                                2.0) + Math.pow(slStarting.y - i
1.y, 2.0)); // length

                                                // between

                                                // line 3
                                float d2 = (float) Math.sqrt(Math.pow((slStartin
g.x - i2.x),
                                                2.0) + Math.pow(slStarting.y - i
2.y, 2.0)); // length

                                                // between

                                Point2D.Float closestPoint;
                                Line2D.Float closestLine;
                                if (d1 > d2) {
                                        // line 4 is the closest outer line
                                        // get intersection between line 4 and l
ine 1,line 2

                                        Point2D.Float ii1 = linesolver.checkInte
rsection(
                                                        (Point2D.Float) line4.ge
tP1(),
                                                        (Point2D.Float) line4.ge
tP2(),
                                                        (Point2D.Float) sLine1.g
etP1(),
                                                        (Point2D.Float) sLine1.g
etP2());
                                        Point2D.Float ii2 = linesolver.checkInte
rsection(
                                                        (Point2D.Float) line4.ge
tP1(),
                                                        (Point2D.Float) line4.ge
tP2(),
                                                        (Point2D.Float) sLine2.g
etP1(),
                                                        (Point2D.Float) sLine2.g
etP2());

                                        float dd1 = (float) Math.sqrt(Math.pow(
                                                        (slStarting.x - ii1.x),
2.0)
                                                        + Math.pow(slStarting.y
- ii1.y, 2.0));
                                        float dd2 = (float) Math.sqrt(Math.pow(
                                                        (slStarting.x - ii2.x),
2.0)
                                                        + Math.pow(slStarting.y
- ii2.y, 2.0));

                                        if (dd1 > dd2) {
                                                // ii2 is the closest intersecti
on point
                                                // lies on line2

                                                closestPoint = ii2;
                                                closestLine = sLine2;

                                        } else if (dd2 > dd1) {
                                                // ii1 is the closest intersecti
on point
                                                // lies on line1
                                        closestPoint = ii1;
                                        closestLine = sLine1;
                                } else {
                                        // they are the same so chose wh
ich ever point
                                        // lies on line1

                                        closestPoint = ii1;
                                        closestLine = sLine1;
                                }
                        } else if (d2 > d1) {
                                // line 3 is the shorted outer line
                                // get intersection between line 3 and l
ine 1, line 2

                                Point2D.Float ii1 = linesolver.checkInte
rsection(
                                                (Point2D.Float) line3.ge
tP1(),
                                                (Point2D.Float) line3.ge
tP2(),
                                                (Point2D.Float) sLine1.g
etP1(),
                                                (Point2D.Float) sLine1.g
etP2());
                                Point2D.Float ii2 = linesolver.checkInte
rsection(
                                                (Point2D.Float) line3.ge
tP1(),
                                                (Point2D.Float) line3.ge
tP2(),
                                                (Point2D.Float) sLine2.g
etP1(),
                                                (Point2D.Float) sLine2.g
etP2());

                                float dd1 = (float) Math.sqrt(Math.pow(
                                                (slStarting.x - ii1.x),
2.0)
                                                + Math.pow(slStarting.y
- ii1.y, 2.0));
                                float dd2 = (float) Math.sqrt(Math.pow(
                                                (slStarting.x - ii2.x),
2.0)
                                                + Math.pow(slStarting.y
- ii2.y, 2.0));

                                if (dd1 > dd2) {
                                        // ii2 is the closest intersecti
on point
                                        // lies on line2

                                        closestPoint = ii2;
                                        closestLine = sLine2;

                                } else if (dd2 > dd1) {
                                        // ii1 is the closest intersecti
on point
                                        // lies on line1

                                        closestPoint = ii1;
                                        closestLine = sLine1;
                                } else {
                                        // they are the same so chose wh
ich ever point
                                        // lies on line1

                                        closestPoint = ii1;
                                        closestLine = sLine1;
```

```
                                        }
                                } else {
                                        throw new UnknownConnectionError();
                                }

                                Point2D.Float closestVec = new Point2D.Float(
                                                ((Point2D.Float) closestLine.get
P1()).x
                                                                - closestPoint.x
,
                                                ((Point2D.Float) closestLine.get
P1()).y
                                                                - closestPoint.y
);

                                intersectingStartPoint = new Point2D.Float(
                                                ((Point2D.Float) startingL.getSt
art()).x - closestVec.x,
                                                ((Point2D.Float) startingL.getSt
art()).y - closestVec.y);

                                intersectingControlPoint = linesolver.checkInter
section(
                                                slStarting, slEnding, endingL.ge
tStart(),
                                                endingL.getEnd());

                                Point2D.Float iVec = new Point2D.Float(intersect
ingStartPoint.x
                                                - intersectingControlPoint.x, in
tersectingStartPoint.y
                                                - intersectingControlPoint.y);
                                float scd = (float) Math.sqrt(Math.pow((iVec.x),
 2.0)
                                                + Math.pow(iVec.y, 2.0));
                                Point2D.Float tVec = new Point2D.Float(endingL.g
etStart().x
                                                - endingL.getEnd().x, endingL.ge
tStart().y
                                                - endingL.getEnd().y);
                                float tVecD = (float) Math.sqrt(Math.pow((tVec.x
), 2.0)
                                                + Math.pow(tVec.y, 2.0));
                                intersectingEndPoint = new Point2D.Float(
                                                intersectingControlPoint.x
                                                                + ((endingL.getE
nd().x - endingL.getStart().x)
/ tVecD * scd),

                                                intersectingControlPoint.y
                                                                + ((endingL.getE
nd().y - endingL.getStart().y)
/ tVecD * scd));

                                ConnectionPoint cp = new ConnectionPoint(current
Road,
                                                startingL, intersectingStartPoin
t);
                                Connection cn = new Connection(currentRoad, star
tingL,
                                                targetRoad, endingL, cp, interse
ctingStartPoint,
                                                intersectingEndPoint, intersecti
ngControlPoint);

                                success = startingL.addConnectionPoint(cp, cn);
                        }
                        return success;
                } else {

                        throw new UnknownConnectionError();
                }
                // end if
                // //////////
                // ////////////

        }

        private static Point2D.Float getHalf(Point2D.Float base1,
                        Point2D.Float base2) {
                Point2D.Float vector = new Point2D.Float(base2.x − base1.x, base
2.y
                                − base1.y);
                // System.out.println("Vector: " + vector);
                float vectorLength = (float) Math.sqrt(Math.pow(vector.x, 2.0)
                                + Math.pow(vector.y, 2.0));
                // System.out.println("Length: " + vectorLength);
                Point2D.Float normalVector = new Point2D.Float(vector.x / vector
Length,
                                vector.y / vectorLength);
                // System.out.println("x calculation: " + normalVector.x +
                // "*"
                // + perpenMat[0][0] + " + " + normalVector.y + "*"
                // + perpenMat[1][0]);
                Point2D.Float perpenVector = new Point2D.Float(normalVector.x
                                * perpenMat[0][0] + normalVector.y * perpenMat[1
][0],
                                normalVector.x * perpenMat[0][1] + normalVector.
y
                                                * perpenMat[1][1]);
                // System.out.println("Perpen vector: " + perpenVector);
                Point2D.Float halfScaled = new Point2D.Float(perpenVector.x
                                * Road.roadWidth / 2, perpenVector.y * Road.road
Width / 2);
                return halfScaled;

        }

        /****************************/
        public static ArrayList<Road> bfsParks(Road currentRoad,
                        CarPark destinationPark) {
                // TODO Auto−generated method stub
                return null;
        }

        public CarWorld getWorld() {
                return this.world;
        }

        public ArrayList<Connection> getConnections() {
                // TODO Auto−generated method stub
                ArrayList<Connection> cal = new ArrayList<Connection>();
                Iterator<Entry<Integer, Lane>> lit = this.lanes.entrySet().itera
tor();
                while (lit.hasNext()) {
                        Map.Entry<Integer, Lane> lp = lit.next();
                        Lane currentLane = lp.getValue();
                        Iterator<Entry<Point2D.Float, ConnectionPoint>> cit = cu
rrentLane
                                        .getConnectionPoints().entrySet().iterat
or();
                        while (cit.hasNext()) {
                                Map.Entry<Point2D.Float, ConnectionPoint> cp = c
it.next();
                                ConnectionPoint currentPoint = cp.getValue();
                                Iterator<Entry<Lane, Connection>> conIt = curren
```

```
tPoint
                                                .getConnections().entrySet().ite
rator();
                                while (conIt.hasNext()) {
                                        Map.Entry<Lane, Connection> conP = conIt
.next();
                                        Connection currentConnection = conP.getV
alue();
                                        cal.add(currentConnection);
                                }
                        }
                }
                return cal;
        }

        // car parks are only allowed to exist on straight roads
        public void setCarParks(int s) {
                if (this instanceof StraightRoad) {
                        Iterator<Entry<Integer, Lane>> lit = this.lanes.entrySet
()
                                        .iterator();
                        while (lit.hasNext()) {
                                Map.Entry<Integer, Lane> lp = lit.next();
                                Lane cl = lp.getValue();
                                int laneKey = cl.getLaneKey();
                                if (laneKey == s && !cl.getHasPark()) {

                                        CarPark newPark = new CarPark(cl, CarPar
k.START, this.world);
                                        this.world.getParks().add(newPark);
                                        System.out.println("Car park spawned");
                                        cl.setHasPark(true);

                                }
                        }

                } else {
                        System.out.println("road type doesn’t support car park");
                }

        }

        //currently exit point can only exist on straight road
        public void setEnding(int s, boolean b) {
                if (this instanceof StraightRoad) {
                        System.out.println("Straight road detected");
                        int remainder = s % 2;
                        Iterator<Entry<Integer, Lane>> lit = this.lanes.entrySet
()
                                        .iterator();
                        while (lit.hasNext()) {
                                Map.Entry<Integer, Lane> lp = lit.next();
                                Lane cl = lp.getValue();
                                int laneKey = cl.getLaneKey();
                                if (laneKey % 2 == remainder) {

                                        cl.setEnding(b);

                                }
                        }

                } else {
                        System.out.println("road type doesn’t support car park");
                }
        }

        public void paintBorders(Graphics g) {
                // TODO Auto−generated method stub
                Iterator<Entry<Integer, Lane>> lit = lanes.entrySet().iterator()
```

```
;

                while (lit.hasNext()) {
                        Lane currentLane = lit.next().getValue();
                        currentLane.paintBorders(g);
                }
        }

}
```

```
package control;

import javax.swing.JFrame;

import model.CarWorld;
import model.Road;

public class RoadController {

	public RoadController(JFrame frame, CarWorld cWorld) {
		// TODO Auto-generated constructor stub
	}

	public Road createRoundAbout() {
		return null;

	}

	public Road createStraightRoad() {
		return null;
	}

}
```

```
package model;

import java.awt.Graphics;
import java.awt.geom.Point2D;
import java.awt.geom.Point2D.Float;

//currently not supported
//lane class of roundRoad object
public class RoundAbout extends Lane {
        private Point2D.Float center;
        private float radius;

        public RoundAbout(Point2D.Float center, float radius, Road cRoad,
                        CarWorld world, int lk) {
                super(center, center, cRoad, world, lk);
                this.center = center;
                this.radius = radius;

        }

        @Override
        public float calculateLaneSpan() {
                // TODO Auto-generated method stub

                return (float) (Math.PI * 2 * this.radius);
        }

        @Override
        public Point2D.Float getStart() {
                return getCenter();
        }

        @Override
        public Point2D.Float getEnd() {
                return getCenter();
        }

        public Point2D.Float getCenter() {
                return this.center;
        }

        @Override
        public Point2D.Float nextPosition(Car car, float targetDistance,
                        float distanceTravelled) {
                // TODO Auto-generated method stub

                double angle = 360 * targetDistance / calculateLaneSpan();

                double rad = Math.toRadians(angle);

                float newX = (float) (this.getStart().x + (Math.cos(rad)
                                * (car.getCoordinate().x - this.getStart().x) +
Math.sin(rad)
                                * (car.getCoordinate().y - this.getStart().y)));

                float newY = (float) (this.getStart().y
                                - (Math.sin(rad) * (car.getCoordinate().x - this
.getStart().x)) + Math
                                .cos(rad) * (car.getCoordinate().y - this.getSta
rt().y));

                return new Point2D.Float(newX, newY);
        }

        public float getRadius() {
                return this.radius;
        }

        @Override
```

```
        public void paint(Graphics g) {
                // TODO Auto-generated method stub

        }

        @Override
        public Road getRoad() {
                // TODO Auto-generated method stub
                return this.contained;
        }

        @Override
        public float findDistance(Car car) {
                // TODO Auto-generated method stub
                return 0;
        }

        @Override
        public Car getFrontCar(Car car) {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public TrafficLight getNextTrafficLight(Car car) {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        public void paintBorders(Graphics g) {
                // TODO Auto-generated method stub

        }

}
```

```
package tests;

import java.awt.geom.Point2D;
import java.awt.geom.Point2D.Float;

import model.Car;
import model.Road;
import model.RoundRoad;
import model.StraightRoad;

public class RoundAboutTest {
        public static void main(String[] args) throws InterruptedException {
                Road ra = new RoundRoad(new Point2D.Float(100, 100), 20, 1);
                Car c1 = new Car();
                c1.setCurrentSpeed(10);
                c1.enterLane(ra.getLanes().get(0), new Point2D.Float(65f, 100f))
;
                while (true) {
                        c1.move();
                        Thread.sleep(20);
                }
        }
}
```

```
package model;

import java.awt.geom.Point2D;
import java.awt.geom.Point2D.Float;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;

public class RoundRoad extends Road {

        // currently not supported
        // can only hold RoundAbout
        // centerPoint= center point of the roundabout
        // radius= radius of center circle
        // numLanes= number of lanes in the roundabout
        public RoundRoad(Point2D.Float centerPoint, int radius, int numLanes,
                        CarWorld world) {
                super(1, world);

                setUpLanes(centerPoint, radius, numLanes);
                Iterator it = this.getLanes().entrySet().iterator();
                while (it.hasNext()) {
                        Map.Entry pair = (Map.Entry) it.next();
                        Lane currentLane = (Lane) pair.getValue();
                        System.out.println("Lane found: " + currentLane.getStart()
+ ","
                                        + currentLane.getEnd());
                }
                this.setBilateral(false);
        }

        private void setUpLanes(Float centerPoint, int radius, int numLanes) {
                int firstRadius = radius + 15;
                for (int i = 0; i < numLanes; i++) {
                        Lane ra = new RoundAbout(centerPoint, firstRadius + (i *
 30), this,
                                        this.getWorld(), i);
                        this.getLanes().put(i, ra);
                }

        }

}
```

```java
package view;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.ArrayList;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import model.Car;
import model.CarPark;
import model.Lane;
import model.Road;
import model.TrafficLight;
import control.TrafficLightController;
import control.WorldController;

class SimulationPanel extends JPanel implements MouseListener {

        private WorldController control;
        private TrafficLightController tlc;
        private CarSimView mainFrame;
        private JButton stopButton, startButton, returnButton; // not used
        private BorderLayout borderLayout;
        private Integer selectedLane = null;

        public SimulationPanel(WorldController control, TrafficLightController t
lc,
                        CarSimView mainFrame) {

                this.control = control;
                this.tlc = tlc;
                this.mainFrame = mainFrame;
                this.setPreferredSize(new Dimension(980, 670));
                this.setBorder(BorderFactory.createMatteBorder(3, 3, 3, 3, Color
.BLACK));
                this.setBackground(Color.WHITE);
                borderLayout = new BorderLayout();
                this.setLayout(borderLayout);
                this.addMouseListener(this);

        }

        protected void paintComponent(Graphics g) {

                ArrayList<Road> roads = control.getRoads();

                for (int i = 0; i < roads.size(); i++) {
                        roads.get(i).paint(g);

                }
                for (int i = 0; i < roads.size(); i++) {
                        roads.get(i).paintBorders(g);
                }

                ArrayList<TrafficLight> lights = control.getLights();
                for (int i = 0; i < lights.size(); i++) {
                        lights.get(i).paint(g);
                }
                ArrayList<Car> cars = new ArrayList<Car>(control.getCars().value
s());
```

```java
                for (int i = 0; i < cars.size(); i++) {
                        cars.get(i).paint(g);

                }
                if (mainFrame.getAddingLight()) {
                        ArrayList<Lane> lanes = this.control.getLanes();
                        for (int i = 0; i < lanes.size(); i++) {
                                g.setColor(Color.BLACK);
                                g.fillOval((int) (lanes.get(i).getStart().x - Ma
th
                                                .sqrt(2 * (Math.pow(7.5 / 2, 2))
)), (int) (lanes.get(i)
                                                .getStart().y - Math.sqrt(2 * (M
ath.pow(7.5 / 2, 2)))),
                                                15, 15);

                        }
                }
        }

        // Can this be remove there not used?

        @Override
        public void mouseClicked(MouseEvent e) {
                // TODO Auto-generated method stub
                // check if traffic light exists where it is clicked

                if (!this.mainFrame.getAddingLight()) {
                        int selected = control.findLight(e.getPoint());
                        if (selected != 0)
                                mainFrame.TrafficPanel(selected);
                        // create new trafficlight panel
                        // change color
                        // add to main panel
                } else if (this.mainFrame.getAddingLight()) {
                        if (selectedLane == null) {
                                selectedLane = control.findLane(e.getPoint());
                                if (selectedLane != null) {
                                        JOptionPane.showMessageDialog(null, "lane
 id : "
                                                        + selectedLane + " has bee
n selected");
                                        this.mainFrame.repaint();
                                        this.mainFrame.revalidate();
                                }
                        }

                        else {
                                tlc.addNewLight(selectedLane, e.getPoint());
                                selectedLane = null;
                                this.mainFrame.setAddingLight(false);
                                this.repaint();
                        }
                }
        }

        @Override
        public void mousePressed(MouseEvent e) {
                // TODO Auto-generated method stub

        }

        @Override
        public void mouseReleased(MouseEvent e) {
                // TODO Auto-generated method stub

        }

        @Override
        public void mouseEntered(MouseEvent e) {
                // TODO Auto-generated method stub

        }

        @Override
        public void mouseExited(MouseEvent e) {
                // TODO Auto-generated method stub

        }

}
```

```
package model;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.geom.Point2D;
import java.awt.geom.Point2D.Float;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

public class StraightLane extends Lane {

        private static final int[][] perpenMat = new int[][] { { 0, −1 }, { 1, 0
 } };

        public StraightLane(Point2D.Float startingPoint, Point2D.Float endPoint,
                        Road cRoad, CarWorld cWorld, int lk) {
                super(startingPoint, endPoint, cRoad, cWorld, lk);
                this.setLaneSpan(calculateLaneSpan());
        }

        @Override
        public float calculateLaneSpan() {
                return (float) Math.sqrt(Math.pow(
                                (this.getEnd().x − this.getStart().x), 2.0)
                                + Math.pow(this.getEnd().y − this.getStart().y,
2.0));
        }

        @Override
        public Point2D.Float nextPosition(Car car, float targetDistance,
                        float distanceTravelled) {

                // update rate at 20ms
                float desiredDistance = targetDistance;

                Point2D.Float start = this.getStart();
                Point2D.Float end = this.getEnd();
                float laneSpan = calculateLaneSpan();
                Point2D.Float displacement = new Point2D.Float((end.x − start.x)
                                / laneSpan * desiredDistance, (end.y − start.y)
/ laneSpan
                                * desiredDistance);

                Point2D.Float newPoint = new Point2D.Float(car.getCoordinate().x
                                + displacement.x, car.getCoordinate().y + displa
cement.y);

                car.setTravelled(car.getTravelled() + targetDistance);
                return newPoint;
        }

        @Override
        public void paint(Graphics g) {
                Point2D.Float start = this.getStart();
                Point2D.Float end = this.getEnd();
                Point2D.Float halfVector = gethalf(start, end);
                Point2D.Float p1 = new Point2D.Float(end.x − halfVector.x, end.y
                                − halfVector.y);
                Point2D.Float p2 = new Point2D.Float(start.x − halfVector.x, sta
rt.y
                                − halfVector.y);

                Point2D.Float p3 = new Point2D.Float(start.x + halfVector.x, sta
rt.y
                                + halfVector.y);
                Point2D.Float p4 = new Point2D.Float(end.x + halfVector.x, end.y
```

```
                                + halfVector.y);
                int xpoints[] = { (int) p1.x, (int) p2.x, (int) p3.x, (int) p4.x
 };
                int ypoints[] = { (int) p1.y, (int) p2.y, (int) p3.y, (int) p4.y
 };
                Graphics2D g2D = (Graphics2D) g;
                RenderingHints qualityHints = new RenderingHints(
                                RenderingHints.KEY_ANTIALIASING,
                                RenderingHints.VALUE_ANTIALIAS_ON);
                qualityHints.put(RenderingHints.KEY_RENDERING,
                                RenderingHints.VALUE_RENDER_QUALITY);
                g2D.setRenderingHints(qualityHints);
                g.setColor(new Color(160, 160, 160));
                g.fillPolygon(xpoints, ypoints, 4);

        }

        private Float gethalf(Float start, Float end) {
                Point2D.Float vector = new Point2D.Float(end.x − start.x, end.y
                                − start.y);

                float vectorLength = (float) Math.sqrt(Math.pow(vector.x, 2.0)
                                + Math.pow(vector.y, 2.0));

                Point2D.Float normalVector = new Point2D.Float(vector.x / vector
Length,
                                vector.y / vectorLength);
                Point2D.Float perpenVector = new Point2D.Float(normalVector.x
                                * perpenMat[0][0] + normalVector.y * perpenMat[1
][0],
                                normalVector.x * perpenMat[0][1] + normalVector.
y
                                                * perpenMat[1][1]);
                // System.out.println("Perpen vector: " + perpenVector);
                Point2D.Float halfScaled = new Point2D.Float(perpenVector.x
                                * Road.roadWidth / 2, perpenVector.y * Road.road
Width / 2);
                return halfScaled;

        }

        @Override
        public Road getRoad() {
                // TODO Auto−generated method stub
                return this.contained;
        }

        @Override
        public float findDistance(Car car) {
                // TODO Auto−generated method stub

                return Car.distance(this.getStart(), car.getCoordinate());
        }

        @Override
        public Car getFrontCar(Car car) {
                // also need to see cars infront of it in the nearest connection
s
                // starting from its lane
                float closest = 100f;
                Car closestCar = null;

                float cClosest = 100f;
                ConnectionPoint closestPoint = null;
                // find closest connection point to the car that is <100f distan
ce

                Iterator<Entry<Point2D.Float, ConnectionPoint>> cpIt = this
                                .getConnectionPoints().entrySet().iterator();
                while (cpIt.hasNext()) {
                        ConnectionPoint currentPoint = cpIt.next().getValue();
                        if (Car.distance(currentPoint.getPointCoordinate(),
                                        car.getCoordinate()) < cClosest) {
                                closestPoint = currentPoint;
                                cClosest = Car.distance(currentPoint.getPointCoo
rdinate(),
                                                car.getCoordinate());
                        }
                }
                float dtp = -1f;
                if (closestPoint != null) {
                        dtp = Car.distance(closestPoint.getPointCoordinate(),
                                        this.getStart());

                        Iterator<Entry<Lane, Connection>> connectionIt = closest
Point
                                        .getConnections().entrySet().iterator();
                        while (connectionIt.hasNext()) {
                                Iterator<Entry<Integer, Car>> connectionCarsIt =
 connectionIt
                                                .next().getValue().getCarsInLane
().entrySet()
                                                .iterator();
                                while (connectionCarsIt.hasNext()) {
                                        Car connectionCar = connectionCarsIt.nex
t().getValue();
                                        if ((connectionCar.getTravelled() + dtp)
 > car
                                                        .getTravelled()) {
                                                if (Car.distance(connectionCar.g
etCoordinate(),
                                                                car.getCoordinat
e()) < closest
                                                                && !car.equals(c
onnectionCar)) {
                                                        closest = Car.distance(
                                                                        connecti
onCar.getCoordinate(),
                                                                        car.getC
oordinate());
                                                        closestCar = connectionC
ar;
                                                }
                                        }
                                }
                        }
                }

                // retrieve cars in the connections that are starting from the c
losest
                // connection point

                // get cars in the connections that belongs to the closest conne
ction
                // point
                Iterator<Entry<Integer, Car>> cit = this.getCarsInLane().entrySe
t()
                                .iterator();

                while (cit.hasNext()) {
                        Map.Entry<Integer, Car> cPair = cit.next();
                        Car cCar = cPair.getValue();

                        if (cCar.getTravelled() > car.getTravelled()) {
                                // cCar is somewhere ahead of the car in the sam
e lane
                                if (((cCar.getTravelled() - car.getTravelled())
< closest)
```

```
                                                && !car.equals(cCar)) {
                                        closest = cCar.getTravelled() - car.getT
ravelled();
                                        closestCar = cCar;
                                }
                        }
                }
                // if closest car is null it means the parameter car is the lead
ing
                // car
                return closestCar;
        }

        @Override
        public TrafficLight getNextTrafficLight(Car car) {
                // TODO Auto-generated method stub
                float closest = 100;
                TrafficLight ctl = null;

                for (int i = 0; i < this.trafficLights.size(); i++) {
                        TrafficLight currentLight = trafficLights.get(i);
                        float td = Car.distance(currentLight.getCoordinate(),
                                        car.getCoordinate());

                        if (td < closest
                                        && Car.distance(car.getCurrentLane().get
Start(),
                                                        currentLight.getCoordina
te()) > car.getTravelled()) {

                                closest = td;
                                ctl = currentLight;

                        }
                }

                return ctl;
        }

        @Override
        public void paintBorders(Graphics g) {
                // TODO Auto-generated method stub
                Point2D.Float start = this.getStart();
                Point2D.Float end = this.getEnd();
                Point2D.Float halfVector = gethalf(start, end);
                Point2D.Float p1 = new Point2D.Float(end.x - halfVector.x, end.y
                                - halfVector.y);
                Point2D.Float p2 = new Point2D.Float(start.x - halfVector.x, sta
rt.y
                                - halfVector.y);

                Point2D.Float p3 = new Point2D.Float(start.x + halfVector.x, sta
rt.y
                                + halfVector.y);
                Point2D.Float p4 = new Point2D.Float(end.x + halfVector.x, end.y
                                + halfVector.y);
                int xpoints[] = { (int) p1.x, (int) p2.x, (int) p3.x, (int) p4.x
 };
                int ypoints[] = { (int) p1.y, (int) p2.y, (int) p3.y, (int) p4.y
 };
                Graphics2D g2D = (Graphics2D) g;
                RenderingHints qualityHints = new RenderingHints(
                                RenderingHints.KEY_ANTIALIASING,
                                RenderingHints.VALUE_ANTIALIAS_ON);
                qualityHints.put(RenderingHints.KEY_RENDERING,
                                RenderingHints.VALUE_RENDER_QUALITY);
                g2D.setRenderingHints(qualityHints);
                g.setColor(Color.white);
                g.drawPolygon(xpoints, ypoints, 4);
        }
}
```

```
package model;

import java.awt.geom.Point2D;
import java.awt.geom.Point2D.Float;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;

public class StraightRoad extends Road {
        private Point2D.Float startingPoint, endingPoint;
        private final int[][] perpenMat = new int[][] { { 0, −1 }, { 1, 0 } };

        // can only hold straight lanes
        // even and odd lane
        // numAddLane= numbers of lanes that equals to the direction of the cent
er
        // line
        // num subLane = numbers of lanes that is opposite of the direction of t
he
        // center line

        public StraightRoad(Point2D.Float startingPoint, Point2D.Float endingPoi
nt,
                        int numAddLane, int numSubLane, CarWorld world) {
                super(Road.STRAIGHT_LANE, world);
                this.startingPoint = startingPoint;
                this.endingPoint = endingPoint;
                setUpLanes(startingPoint, endingPoint, numAddLane, numSubLane);
                Iterator it = this.getLanes().entrySet().iterator();
                if ((numAddLane == 0 && numSubLane != 0)
                                || (numAddLane != 0 && numSubLane == 0))
                        this.setBilateral(false);
                else
                        this.setBilateral(true);

                while (it.hasNext()) {
                        Map.Entry pair = (Map.Entry) it.next();
                        Lane currentLane = (Lane) pair.getValue();
                        System.out.println("Lane found: " + currentLane.getStart()
+ ","
                                        + currentLane.getEnd());
                }

                // TODO Auto−generated constructor stub
        }

        private void setUpLanes(Point2D.Float startingPoint,
                        Point2D.Float endingPoint, int numAddLane, int numSubLan
e) {
                // TODO Auto−generated method stub
                Point2D.Float vector = new Point2D.Float(endingPoint.x
                                − startingPoint.x, endingPoint.y − startingPoint
.y);
                // System.out.println("Vector: " + vector);
                float vectorLength = (float) Math.sqrt(Math.pow(vector.x, 2.0)
                                + Math.pow(vector.y, 2.0));
                // System.out.println("Length: " + vectorLength);
                Point2D.Float normalVector = new Point2D.Float(vector.x / vector
Length,
                                vector.y / vectorLength);
                // System.out.println("x calculation: " + normalVector.x + "*"
                // + perpenMat[0][0] + " + " + normalVector.y + "*"
                // + perpenMat[1][0]);
                Point2D.Float perpenVector = new Point2D.Float(normalVector.x
                                * perpenMat[0][0] + normalVector.y * perpenMat[1
][0],
                                normalVector.x * perpenMat[0][1] + normalVector.
y
                                                * perpenMat[1][1]);
```

```
                // System.out.println("Perpen vector: " + perpenVector);
                Point2D.Float scaledPerpen = new Point2D.Float(perpenVector.x
                                * Road.roadWidth, perpenVector.y * Road.roadWidt
h);
                Point2D.Float halfScaled = new Point2D.Float(perpenVector.x
                                * Road.roadWidth / 2, perpenVector.y * Road.road
Width / 2);

                // setting up add lanes
                for (int i = 0; i < numAddLane; i++) {
                        int laneNumber = i * 2;
                        Point2D.Float newStart = new Point2D.Float(startingPoint
.x
                                        + halfScaled.x + i * scaledPerpen.x, sta
rtingPoint.y
                                        + halfScaled.y + i * scaledPerpen.y);
                        Point2D.Float newEnd = new Point2D.Float(endingPoint.x
                                        + halfScaled.x + scaledPerpen.x * i, end
ingPoint.y
                                        + halfScaled.y + i * scaledPerpen.y);
                        Lane newStraight = new StraightLane(newStart, newEnd, th
is,
                                        this.getWorld(), laneNumber);
                        this.getLanes().put(laneNumber, newStraight);
                }

                // setting up sub lanes
                for (int i = 0; i < numSubLane; i++) {
                        int laneNumber = i * 2 + 1;
                        Point2D.Float newStart = new Point2D.Float(endingPoint.x
                                        − halfScaled.x − i * scaledPerpen.x, end
ingPoint.y
                                        − halfScaled.y − i * scaledPerpen.y);

                        Point2D.Float newEnd = new Point2D.Float(startingPoint.x
                                        − halfScaled.x − i * scaledPerpen.x, sta
rtingPoint.y
                                        − halfScaled.y − i * scaledPerpen.y);
                        Lane newStraight = new StraightLane(newStart, newEnd, th
is,
                                        this.getWorld(), laneNumber);
                        this.getLanes().put(laneNumber, newStraight);
                }

        }
}
```

```
package tests;

import java.awt.geom.Point2D;
import java.awt.geom.Point2D.Float;

import model.Car;
import model.Road;
import model.StraightRoad;

public class StraightRoadTest {
        public static void main(String[] args) throws InterruptedException {
                Road sr = new StraightRoad(new Point2D.Float(100, 100),
                                new Point2D.Float(180, 160), 2, 2);
                Car c1 = new Car();
                c1.setCurrentSpeed(10);
                c1.enterLane(sr.getLanes().get(3), sr.getLanes().get(3).getStart
());
                while (true) {
                        c1.move();
                        Thread.sleep(20);
                }
        }
}
```

```
package model;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.geom.Point2D;
import java.util.Date;

public class TrafficLight {
        private Lane lane; // lane object it belongs to
        private String status;
        private boolean initial;
        private float greenInterval;
        private float redInterval;
        private float initInterval;
        private Date lastChanged;
        private Point2D.Float coordination;
        private int id;
        private static int totalLights = 0;
        private double tempInterval = 0.0;

        // testing stubs
        public TrafficLight() {
                this.greenInterval = 5;

                this.redInterval = 5;
                this.lastChanged = new Date();
                System.out.println(this.lastChanged);
                this.status = "Green";
                this.id = totalLights;
                totalLights++;
        }

        public TrafficLight(Lane lane) {
                this.lane = lane;

                this.lastChanged = new Date();
                this.status = "Green";
                this.id = totalLights;
                totalLights++;
        }

        // testing stubs end

        public TrafficLight(Lane lane, String status, float greenInterval,
                        float redInterval, float initInterval, Point2D.Float coo
rdination) {
                this.lane = lane;
                System.out.println("traffic coord" + coordination);
                initial = true;
                if (initInterval == 0) {
                        initial = false;
                }
                this.initInterval = initInterval;
                this.redInterval = redInterval;
                this.greenInterval = greenInterval;
                this.lastChanged = new Date();
                this.status = status;
                this.coordination = coordination;
                this.id = totalLights + 1;
                totalLights++;
        }

        public void setCoordinate(Point2D.Float coordination) {
                this.coordination = coordination;
        }

        public Point2D.Float getCoordinate() {
                return this.coordination;
        }
```

```

        public String getStatus() {
                return status;
        }

        public void setStatus(String status) {
                this.status = status;
        }

        public void setRedInterval(float interval) {
                this.redInterval = interval;
        }

        public Lane getLane() {
                return this.lane;
        }

        public void paint(Graphics g) {

                if (status.equalsIgnoreCase("green")) {

                        g.setColor(Color.GREEN);

                } else if (status.equalsIgnoreCase("red")) {
                        g.setColor(Color.RED);
                }
                g.fillOval(
                                (int) (coordination.x - Math.sqrt(2 * (Math.pow(
7.5 / 2, 2)))),
                                (int) (coordination.y - Math.sqrt(2 * (Math.pow(
7.5 / 2, 2)))),
                                15, 15);
                g.setColor(Color.BLACK);

        }

        public void update() {
                /*
                 * Date currentDate = new Date(); if (currentDate.getTime() -
                 * lastChanged.getTime() > interval) { System.out.println("chang
ing");
                 * System.out.println(currentDate.getTime() - lastChanged.getTim
e());
                 * lastChanged = currentDate; if (status.equals("Green")) { this
.status
                 * = "Red"; } else if (status.equals("Red")) { this.status = "Gr
een"; }
                 * System.out.println(this.status); }
                 */

                if (initial) {
                        // stay red till initial interval
                        // past initial interval make initial false
                        this.status = "red";
                        tempInterval += 0.02;
                        if (tempInterval >= initInterval) {
                                this.initial = false;
                                this.status = "green";
                        }
                } else {
                        if (this.status.equalsIgnoreCase("green")) {

                                if (tempInterval >= greenInterval + initInterval
) {
                                        if (redInterval != 0)
                                                this.status = "red";
                                        tempInterval = initInterval;
                                } else {
                                        tempInterval += 0.02;
                                }
                        } else {
                                if (tempInterval >= redInterval + initInterval)
{
                                        if (greenInterval != 0)
                                                this.status = "green";
                                        tempInterval = initInterval;
                                } else {
                                        tempInterval += 0.02;
                                }
                        }

                }

                // System.out.println(getStatus());
        }

        public int getId() {
                return this.id;
        }

        public float getGreen() {
                return this.greenInterval;
        }

        public float getRed() {
                return this.redInterval;
        }

        public float getInit() {
                return this.initInterval;
        }

        public void setGreen(float f) {
                this.greenInterval = f;
        }

        public void setInit(float f) {
                this.initInterval = f;
        }

        public void reset() {
                this.tempInterval = 0d;
                this.status = "green";
                if (initInterval != 0)
                        this.initial = true;
                else if (initInterval == 0)
                        this.initial = false;
        }

        public static void setTotalLights(int i) {
                // TODO Auto-generated method stub
                totalLights = i;
        }

}
```

```java
package control;

import java.awt.Point;
import java.util.ArrayList;

import javax.swing.JFrame;

import model.CarWorld;
import model.TrafficLight;

public class TrafficLightController {
        CarWorld world;
        JFrame mainFrame;

        public TrafficLightController(CarWorld cw) {
                this.world = cw;
        }

        public void setView(JFrame frame) {
                // TODO Auto-generated method stub
                mainFrame = frame;
        }

        public ArrayList<TrafficLight> getLights() {
                // TODO Auto-generated method stub
                return this.world.getLights();
        }

        public void setInterval(String color, float interval, int id) {
                ArrayList<TrafficLight> lights = getLights();

                for (int i = 0; i < lights.size(); i++) {
                        if (lights.get(i).getId() == id) {
                                TrafficLight selected = lights.get(i);
                                if (color.equals("red")) {
                                        selected.setRedInterval(interval);
                                        System.out.println("traffic light id : " + id
                                                        + " red interval changed to : " +
 interval);
                                } else if (color.equals("green")) {
                                        selected.setGreen(interval);
                                        System.out.println("traffic light id : " + id
                                                        + " green interval changed to : "
+ interval);
                                } else if (color.equals("initial")) {
                                        selected.setInit(interval);
                                        System.out.println("traffic light id : " + id
                                                        + " initial interval changed to : "
+ interval);
                                }
                        }
                }
        }

        public void removeLight(int id) {
                ArrayList<TrafficLight> lights = getLights();
                for (int i = 0; i < lights.size(); i++) {
                        if (lights.get(i).getId() == id) {
                                lights.get(i).getLane().removeTrafficLight(id);
                        }
                }

        }

        public void addNewLight(Integer selectedLane, Point point) {
                // TODO Auto-generated method stub
                this.world.addNewLight(selectedLane, point);
                mainFrame.repaint();
                mainFrame.revalidate();
```

```java
        }
}
```

```java
package view;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;

import model.TrafficLight;
import control.TrafficLightController;
import control.WorldController;

public class TrafficLightPanel extends JPanel implements ActionListener {
        private JPanel panel, btnHolderPanel;
        private JScrollPane jsp;
        private BorderLayout bl = new BorderLayout();
        private JButton rtrnButton;
        private static final int INTERVAL_MIN = 0;
        private static final int INTERVAL_MAX = 20;
        private ArrayList<TrafficLight> lights;

        private WorldController wController;
        private CarSimView mainFrame;

        private String TrifficLightID;
        private ImageIcon traffic_img;
        private HashMap<Integer, JTextField> greenList, redList, initList;

        public TrafficLightPanel(WorldController wc,
                        final TrafficLightController tlc, final CarSimView mainF
rame, int id) {
                this.wController = wc;
                this.greenList = new HashMap<Integer, JTextField>();
                this.redList = new HashMap<Integer, JTextField>();
                this.initList = new HashMap<Integer, JTextField>();
                this.mainFrame = mainFrame;

                panel = new JPanel();
                btnHolderPanel = new JPanel();
                rtrnButton = new CustomJButton("Return");
                rtrnButton.addActionListener(this);
                lights = tlc.getLights();
                System.out.println(lights.size());

                JLabel trafficLightTitle = new JLabel();
                traffic_img = new ImageIcon(getClass().getResource(File.separato
r + "gfx"
                                + File.separator + "adjustTrafficLight_gfx.png"));
                trafficLightTitle.setIcon(traffic_img);
                //JLabel space = new JLabel("  ");
                JLabel iInterval = new JLabel("Initial Interval ");
                JLabel gInterval = new JLabel(" Green Interval ");
                JLabel rInterval = new JLabel(" Red Interval");

                Font font = new Font("Tahoma", Font.BOLD, 15);
                iInterval.setFont(font);
                gInterval.setFont(font);
```

```java
                rInterval.setFont(font);
                //space.setFont(font);

                JPanel a = new JPanel();
                a.setPreferredSize(new Dimension(920, 30));
        //      a.add(space);
                a.add(iInterval);
                a.add(gInterval);
                a.add(rInterval);
                a.add(new JLabel("          "));

                panel.add(a);
                int trafficLightSize = lights.size();
                panel.setPreferredSize(new Dimension(920, 64 * trafficLightSize)
);

                System.out.println("size" + trafficLightSize);
                if (id == 0) {
                        for (int i = 0; i < trafficLightSize; i++) {

                                JPanel container = new JPanel();
                                container.setBackground(Color.WHITE);
                                container.setPreferredSize(new Dimension(920, 58
));
                                TrifficLightID = "" + lights.get(i).getId(); //
THIS IS NEW

                                JLabel lbl = new JLabel("traffic light "+TrifficLightI
D+" ");
                                Font font2 = new Font("Tahoma", Font.BOLD, 16);
                                lbl.setFont(font2);
                                lbl.setForeground(Color.black);
                                if (lights.get(i).getId() == id) {
                                        lbl.setOpaque(true);
                                        container.setBackground(Color.red);
                                        lbl.setForeground(Color.blue);
                                }
                                container.add(lbl);
                                JTextField initialTField = new JTextField(""
                                                + lights.get(i).getInit(), 7);
                                initList.put(lights.get(i).getId(), initialTFiel
d);
                                JTextField greenTField = new JTextField(""
                                                + lights.get(i).getGreen(), 7);
                                greenList.put(lights.get(i).getId(), greenTField
);
                                JTextField redTField = new JTextField(""
                                                + lights.get(i).getRed(), 7);
                                redList.put(lights.get(i).getId(), redTField);

                                JButton submitBtn = new LightButton("submit",
                                                Integer.parseInt(TrifficLightID)
);
                                submitBtn.setPreferredSize(new Dimension(100, 25
));
                                submitBtn.addActionListener(new ActionListener()
 {

                                        @Override
                                        public void actionPerformed(ActionEvent
e) {
                                                // TODO Auto-generated method st
ub
                                                int id = ((LightButton) e.getSou
rce()).getId();

                                                JTextField init = initList.get(i
d);
```

```
                                                JTextField red = redList.get(id)
;

                                                JTextField green = greenList.get
(id);
                                                try {
                                                        float redInt = Float.par
seFloat(red.getText());
                                                        float initInt = Float.pa
rseFloat(init.getText());
                                                        float greenInt = Float.p
arseFloat(green.getText());
                                                        if (redInt < 0 || initIn
t < 0 || greenInt < 0)
                                                                throw new Except
ion();

                                                        tlc.setInterval("red", r
edInt, id);
                                                        tlc.setInterval("green",
greenInt, id);
                                                        tlc.setInterval("initial",
initInt, id);
                                                } catch (Exception exc) {
                                                        JOptionPane
                                                                        .showMes
sageDialog(null,
"check if the input is valid. Input needs to be a positive float number");
                                                }

                                        }

                                });
                                JButton removeBtn = new LightButton("remove",
                                                Integer.parseInt(TrifficLightID)
);
                                removeBtn.setPreferredSize(new Dimension(100, 25
));
                                removeBtn.addActionListener(new ActionListener()
 {

                                        @Override
                                        public void actionPerformed(ActionEvent
e) {
                                                // TODO Auto-generated method st
ub
                                                tlc.removeLight(((LightButton) e
.getSource()).getId());
                                                mainFrame.TrafficPanel(0);

                                        }

                                });
                                container.add(new JLabel("   "));
                                container.add(initialTField);
                                container.add(new JLabel("      "));
                                container.add(greenTField);
                                container.add(new JLabel("      "));
                                container.add(redTField);
                                container.add(new JLabel("   "));
                                container.add(submitBtn);
                                container.add(removeBtn);
                                panel.add(container);

                        }
                } else {
                        JPanel container = new JPanel();
                        container.setBackground(Color.WHITE);
```

```
                        container.setPreferredSize(new Dimension(920, 58));
                        TrifficLightID = ""; // THIS IS NEW
                        TrafficLight cl = null;
                        for (int i = 0; i < lights.size(); i++) {

                                if (lights.get(i).getId() == id) {
                                        cl = lights.get(i);
                                        TrifficLightID += cl.getId();
                                        break;
                                }

                        }
                        JLabel lbl = new JLabel("traffic light "+TrifficLightID+" ");
                        Font font2 = new Font("Tahoma", Font.BOLD, 16);
                        lbl.setFont(font2);
                        lbl.setForeground(Color.black);

                        container.add(lbl);
                        JTextField initialTField = new JTextField("" + cl.getIni
t(), 7);
                        initList.put(cl.getId(), initialTField);
                        JTextField greenTField = new JTextField("" + cl.getGreen
(), 7);
                        greenList.put(cl.getId(), greenTField);
                        JTextField redTField = new JTextField("" + cl.getRed(),
7);
                        redList.put(cl.getId(), redTField);

                        JButton submitBtn = new LightButton("submit",
                                        Integer.parseInt(TrifficLightID));
                        submitBtn.setPreferredSize(new Dimension(100, 25));
                        submitBtn.addActionListener(new ActionListener() {

                                @Override
                                public void actionPerformed(ActionEvent e) {
                                        // TODO Auto-generated method stub
                                        int id = ((LightButton) e.getSource()).g
etId();

                                        JTextField init = initList.get(id);

                                        JTextField red = redList.get(id);

                                        JTextField green = greenList.get(id);
                                        try {
                                                float redInt = Float.parseFloat(
red.getText());
                                                float initInt = Float.parseFloat
(init.getText());
                                                float greenInt = Float.parseFloa
t(green.getText());
                                                if (redInt < 0 || initInt < 0 ||
 greenInt < 0)
                                                        throw new Exception();
                                                tlc.setInterval("red", redInt, i
d);
                                                tlc.setInterval("green", greenInt
, id);
                                                tlc.setInterval("initial", initInt,
 id);
                                        } catch (Exception exc) {
                                                JOptionPane
                                                                .showMessageDial
og(null,
"check if the input is valid. Input needs to be a positive float number");
                                        }

                                }
                        });
                        JButton removeBtn = new LightButton("remove",
                                        Integer.parseInt(TrifficLightID));
                        removeBtn.setPreferredSize(new Dimension(100, 25));
                        removeBtn.addActionListener(new ActionListener() {

                                @Override
                                public void actionPerformed(ActionEvent e) {
                                        // TODO Auto-generated method stub
                                        tlc.removeLight(((LightButton) e.getSour
ce()).getId());

                                        panel.removeAll();
                                        repaint();

                                }

                        });
                        container.add(new JLabel("   "));
                        container.add(initialTField);
                        container.add(new JLabel("      "));
                        container.add(greenTField);
                        container.add(new JLabel("      "));
                        container.add(redTField);
                        container.add(new JLabel("   "));
                        container.add(submitBtn);
                        container.add(removeBtn);
                        panel.add(container);
                }
                /*
                 * JSlider slider1 = new JSlider(); JSlider slider2 = new JSlide
r();
                 * JSlider slider3 = new JSlider(); JSlider slider4 = new JSlide
r();
                 * slider1.setPaintTicks(true); slider1.setPaintLabels(true);
                 * slider1.setMinorTickSpacing(2);
                 */

                jsp = new JScrollPane(panel, JScrollPane.VERTICAL_SCROLLBAR_ALWA
YS,
                                JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
                panel.setBackground(Color.WHITE);
                btnHolderPanel.setPreferredSize(new Dimension(1000, 200));
                jsp.setPreferredSize(new Dimension(920, 500));
                jsp.setBackground(Color.WHITE);
                this.add(trafficLightTitle);
                this.add(jsp);
                this.add(btnHolderPanel);
                btnHolderPanel.add(rtrnButton);

        }

        @Override
        public void actionPerformed(ActionEvent e) {
                // TODO Auto-generated method stub
                if (e.getSource() == rtrnButton) { // T Junction choosen by user
                        this.wController.pause();
                        mainFrame.simulationView(); // Need to confirm where use
r goes back

// too
                }

        }

}
```

```
package tests;

import model.TrafficLight;

public class TrafficLightTest {
	public static void main(String[] args) throws InterruptedException {
		TrafficLight light = new TrafficLight();
		System.out.println("Traffic light created");
		while (light.getStatus().equals("Green")) {
			light.update();
			System.out.println(light.getStatus());
			Thread.sleep(1000);
		}
	}
}
```

```java
package model;

public class UnknownConnectionError extends Exception {

}
```

```
package view;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

import control.WorldController;

public class UserHelpPanel extends JPanel implements ActionListener {
        WorldController wController;
        CarSimView mainFrame;
        private JScrollPane jscrollPane;
        private JPanel usrHelpPanel, btnHolderPanel;
        private JButton rtrnButton;
        private JLabel helpTitle, aimTitle, trafficTitle, collsionTitle,
                        addTrafficLightLabel, collisionLabel;
        private ImageIcon helpTitle_img, aim_img, addTrafficLight_img,
                        collision_img;
        private JTextArea introTextArea, trafficTextArea, collsionTextArea;
        private Font titleFont = new Font("Tahoma", Font.BOLD, 20);

        public UserHelpPanel(WorldController wController, CarSimView mainFrame)
{
                this.wController = wController;
                this.mainFrame = mainFrame;

                this.setBorder(BorderFactory.createMatteBorder(3, 3, 3, 3, Color
.BLACK));

                usrHelpPanel = new JPanel();
                btnHolderPanel = new JPanel();

                introTextArea = new JTextArea(2, 50);
                introTextArea.setLineWrap(true);
                introTextArea.setEditable(false);
                trafficTextArea = new JTextArea(2, 50);
                trafficTextArea.setLineWrap(true);
                trafficTextArea.setEditable(false);
                collsionTextArea = new JTextArea(2, 50);
                collsionTextArea.setLineWrap(true);
                collsionTextArea.setEditable(false);

                rtrnButton = new CustomJButton("Return");
                rtrnButton.addActionListener(this);

                helpTitle = new JLabel();
                helpTitle_img = new ImageIcon(getClass().getResource( File.separ
ator + "gfx"
                                + File.separator + "helpTitle_gfx.gif"));
                helpTitle.setIcon(helpTitle_img);
                aimTitle = new JLabel();
                aim_img = new ImageIcon(getClass().getResource( File.separator +
 "gfx" + File.separator
                                + "aim_gfx.gif"));
                aimTitle.setIcon(aim_img);

                trafficTitle = new JLabel("How to add a traffic light:");
                collsionTitle = new JLabel("What is a collsion?");
```

```
                trafficTitle.setFont(titleFont);
                collsionTitle.setFont(titleFont);
                addTrafficLightLabel = new JLabel();
                addTrafficLight_img = new ImageIcon(getClass().getResource(
                                File.separator + "gfx" + File.separator
                                                + "addTrafficLightHelp.png"));
                addTrafficLightLabel.setIcon(addTrafficLight_img);

                collisionLabel = new JLabel();
                collision_img = new ImageIcon(getClass().getResource(
                                File.separator + "gfx" + File.separator + "collison
Help.png"));
                collisionLabel.setIcon(collision_img);

                this.add(helpTitle);

                jscrollPane = new JScrollPane(usrHelpPanel,
                                JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                                JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
                jscrollPane.setPreferredSize(new Dimension(700, 400));
                jscrollPane.setBackground(Color.WHITE);
                jscrollPane.setBorder(BorderFactory.createMatteBorder(2, 2, 2, 2
,
                                Color.BLACK));

                // introTextArea
                // .setText("The aim of this software is to allow you to find th
e perfect coordination f");
                trafficTextArea
                                .setText("The above image illustrates how you can add new traffic
 light to the system. (1) Press the 'Add Traffic Light' button and then the screen will pause and you will then see the bla
ck circles at each lane entry point (2) Select the lane you would like to add the traffic light to (3) Simply click on the pa
rt of the lane you would like to add the traffic light to. If you make a mistake or do not place the traffic light in the posit
ion you wanted simply, select the new tarffic light created and once 'Adjust Traffic Light' screen opens up simply delet
e that traffic light and start again.");
                collsionTextArea
                                .setText("The image above illustrates an example of a collision, th
is is an indication that they setup of the traffic light intervals is not good and has resulted in a collsion between two cars
. To solve this issue select 'Adjust Traffic Lights' or select each traffic light and make adjustments and then press the 's
ubmit' button to confirm these changes. When making adjustments to the traffic light intervals the [Initial] Interval is th
e interval that the traffic light will stay as red before it enters recurring [Green] and [Red] light loop. The [Green] and [
Red] interval values are the time (seconds) you would like the interval to last");
                // usrHelpPanel.add(aimTitle);
                // usrHelpPanel.add(introTextArea);
                usrHelpPanel.add(trafficTitle);
                usrHelpPanel.add(addTrafficLightLabel);
                usrHelpPanel.add(trafficTextArea);
                usrHelpPanel.add(collsionTitle);
                usrHelpPanel.add(collisionLabel);
                usrHelpPanel.add(collsionTextArea);
                usrHelpPanel.setBackground(Color.WHITE);
                usrHelpPanel.setPreferredSize(new Dimension(700, 1000));
                btnHolderPanel.setPreferredSize(new Dimension(1000, 150));

                this.add(jscrollPane);
                this.add(btnHolderPanel);
                btnHolderPanel.add(rtrnButton);

        }

        @Override
        public void actionPerformed(ActionEvent e) {
                // TODO Auto-generated method stub
                if (e.getSource() == rtrnButton) { // T Junction choosen by user
                        this.wController.pause();
                        mainFrame.simulationView(); // Need to confirm where use
r goes back

// too
```

```
                }

        }
}
```

```java
package control;

import java.awt.Point;
import java.awt.geom.Point2D;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map.Entry;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

import view.CarSimView;
import model.Car;
import model.CarPark;
import model.CarWorld;
import model.Lane;
import model.Road;
import model.StraightRoad;
import model.TrafficLight;

public class WorldController {
        private CarWorld cWorld; // core model
        private JFrame carView; // JFrame

        public WorldController() {

                this.cWorld = this.createWorld();

        }

        public CarWorld getcWorld() {
                return cWorld;
        }

        public void setcWorld(CarWorld cWorld) {
                this.cWorld = cWorld;
        }

        public String toString() {
                return cWorld.toString();

        }

        public void setView(JFrame frame) {
                this.carView = frame;
        }

        public void pause() {
                this.cWorld.setStatus("paused");
        }

        public void start() {
                this.cWorld.setStatus("running");
        }

        public void exit() {
                this.cWorld.setStatus("exit");
        }

        // program loop
        public void simulate() throws InterruptedException {

                while (true) {

                        if (cWorld.getStatus().equals("running")) {
                                update();

                                Thread.sleep(20); // the timing mechanism
```

```java
                                                                   // needs
 improvement
                        } else if (cWorld.getStatus().equals("paused")) {
                                Thread.sleep(20);
                        } else if (cWorld.getStatus().equals("exit")) {
                                return;
                        }
                }
        }

        public HashMap<Integer, Car> getCars() {
                // TODO Auto-generated method stub
                return this.cWorld.getCars();
        }

        public ArrayList<Road> getRoads() {
                // TODO Auto-generated method stub
                return this.cWorld.getRoads();
        }

        private void update() {

                for (int i = 0; i < cWorld.getParks().size(); i++) {
                        cWorld.getParks().get(i).update();
                }

                ArrayList<Car> cars = new ArrayList<Car>(cWorld.getCars().values
());
                for (int i = 0; i < cars.size(); i++) {
                        cars.get(i).move();

                }
                ArrayList<TrafficLight> lights = cWorld.getLights();
                for (int i = 0; i < lights.size(); i++) {
                        lights.get(i).update();
                }
                ((CarSimView) carView).getDynamicChart().updateData();
                ArrayList<Car> collided = this.cWorld.checkCollision();
                if (collided.size() == 2) {

                        cWorld.reset();
                        for (int i = 0; i < collided.size(); i++) {
                                this.cWorld.addCar(collided.get(i));
                        }
                        this.carView.repaint();
                        JOptionPane.showMessageDialog(null, "Collision detected!");
                        cWorld.reset();
                }

                this.carView.repaint();

        }

        public void setFullSimulation() {
                this.cWorld.setStatus("paused");
                this.cWorld.flush();
                Road road1 = new StraightRoad(new Point2D.Float(100, 70),
                                new Point2D.Float(1000, 70), 1, 1, this.getcWorl
d());
                Road road2 = new StraightRoad(new Point2D.Float(800, 50),
                                new Point2D.Float(800, 590), 1, 1, this.getcWorl
d());
                Road road3 = new StraightRoad(new Point2D.Float(100, 570),
                                new Point2D.Float(1000, 570), 1, 1, this.getcWor
ld());
                Road road4 = new StraightRoad(new Point2D.Float(100, 200),
                                new Point2D.Float(1000, 400), 2, 2, this.getcWor
ld());
                Road road5 = new StraightRoad(new Point2D.Float(550, 70),
                                new Point2D.Float(180, 570), 1, 1, this.getcWorl
d());
                try {
                        Road.connectLane(road1, 0, road2, 0);
                        Road.connectLane(road1, 0, road5, 0);
                        Road.connectLane(road1, 1, road2, 0);
                        Road.connectLane(road1, 1, road5, 0);

                        Road.connectLane(road2, 1, road1, 0);
                        Road.connectLane(road2, 1, road1, 1);
                        Road.connectLane(road2, 0, road3, 0);
                        Road.connectLane(road2, 0, road3, 1);
                        Road.connectLane(road2, 0, road4, 2);
                        Road.connectLane(road2, 0, road4, 0);
                        Road.connectLane(road2, 0, road4, 1);
                        Road.connectLane(road2, 0, road4, 3);
                        Road.connectLane(road2, 1, road4, 2);
                        Road.connectLane(road2, 1, road4, 0);
                        Road.connectLane(road2, 1, road4, 1);
                        Road.connectLane(road2, 1, road4, 3);

                        Road.connectLane(road3, 0, road2, 1);
                        Road.connectLane(road3, 1, road2, 1);
                        Road.connectLane(road3, 0, road5, 1);
                        Road.connectLane(road3, 1, road5, 1);

                        Road.connectLane(road5, 1, road1, 0);
                        Road.connectLane(road5, 1, road1, 1);
                        Road.connectLane(road5, 0, road3, 0);
                        Road.connectLane(road5, 0, road3, 1);
                        Road.connectLane(road5, 0, road4, 2);
                        Road.connectLane(road5, 0, road4, 0);
                        Road.connectLane(road5, 0, road4, 1);
                        Road.connectLane(road5, 0, road4, 3);
                        Road.connectLane(road5, 1, road4, 2);
                        Road.connectLane(road5, 1, road4, 0);
                        Road.connectLane(road5, 1, road4, 1);
                        Road.connectLane(road5, 1, road4, 3);

                        Road.connectLane(road4, 2, road5, 1);
                        Road.connectLane(road4, 0, road5, 0);
                        Road.connectLane(road4, 2, road2, 1);
                        Road.connectLane(road4, 0, road2, 0);
                        Road.connectLane(road4, 1, road5, 1);
                        Road.connectLane(road4, 3, road5, 0);
                        Road.connectLane(road4, 1, road2, 1);
                        Road.connectLane(road4, 3, road2, 0);

                } catch (Exception e) {

                }
                road1.setCarParks(0);
                road1.setCarParks(1);
                road1.setEnding(0, true);
                road1.setEnding(1, true);

                road1.setCarParks(0);
                road1.setCarParks(1);
                road1.setEnding(0, true);
                road1.setEnding(1, true);

                road4.setCarParks(0);
                road4.setCarParks(1);
                road4.setCarParks(2);
                road4.setCarParks(3);
                road4.setEnding(0, true);
                road4.setEnding(1, true);
                road4.setEnding(2, true);
                road4.setEnding(3, true);
```

```
                road3.setCarParks(0);
                road3.setCarParks(1);
                road3.setEnding(0, true);
                road3.setEnding(1, true);

                this.cWorld.addRoad(road1);
                this.cWorld.addRoad(road2);
                this.cWorld.addRoad(road3);
                this.cWorld.addRoad(road4);
                this.cWorld.addRoad(road5);
        }

        // example case setup
        public void setTJunction() {
                this.cWorld.setStatus("paused");
                this.cWorld.flush();
                // add new roads and such
                Road sr = new StraightRoad(new Point2D.Float(100, 100),
                                new Point2D.Float(1000, 100), 1, 1, this.getcWor
ld());
                Road ar = new StraightRoad(new Point2D.Float(550, 80),
                                new Point2D.Float(550, 600), 1, 1, this.getcWorl
d());
                try {
                        Road.connectLane(sr, 0, ar, 0);
                        Road.connectLane(sr, 1, ar, 0);
                        Road.connectLane(ar, 1, sr, 0);
                        Road.connectLane(ar, 1, sr, 1);
                } catch (Exception e) {

                }
                sr.setCarParks(0);
                sr.setCarParks(1);
                ar.setCarParks(1);

                sr.setEnding(0, true);
                sr.setEnding(1, true);
                ar.setEnding(0, true);
                this.cWorld.addRoad(sr);
                this.cWorld.addRoad(ar);
        }

        public void setIntersection() {
                // construct the world with road network
                this.cWorld.setStatus("paused");
                this.cWorld.flush();
                // add new roads and such
                Road sr = new StraightRoad(new Point2D.Float(100, 200),
                                new Point2D.Float(1000, 400), 2, 2, this.getcWor
ld());

                Road cr = new StraightRoad(new Point2D.Float(700, 100),
                                new Point2D.Float(700, 600), 2, 2, this.getcWorl
d());

                try {

                        Road.connectLane(sr, 0, cr, 0);
                        Road.connectLane(sr, 2, cr, 3);
                        Road.connectLane(sr, 3, cr, 2);
                        Road.connectLane(sr, 1, cr, 1);

                        Road.connectLane(cr, 0, sr, 1);
                        Road.connectLane(cr, 2, sr, 2);
                        Road.connectLane(cr, 3, sr, 3);
                        Road.connectLane(cr, 1, sr, 0);

                } catch (Exception e) {
                        System.out.println("error while connecting");
                        e.printStackTrace();

                }
                sr.setCarParks(0);

                sr.setCarParks(1);

                sr.setCarParks(2);
                sr.setCarParks(3);

                cr.setCarParks(1);
                cr.setCarParks(0);
                cr.setCarParks(2);
                cr.setCarParks(3);

                sr.setEnding(0, true);
                sr.setEnding(1, true);
                sr.setEnding(2, true);
                sr.setEnding(3, true);

                cr.setEnding(0, true);
                cr.setEnding(1, true);
                cr.setEnding(2, true);
                cr.setEnding(3, true);

                this.cWorld.addRoad(sr);

                this.cWorld.addRoad(cr);

        }

        public CarWorld createWorld() {
                if (this.cWorld == null) {
                        CarWorld world = new CarWorld();
                        this.cWorld = world;
                        return cWorld;
                } else {
                        return this.cWorld;
                }
        }

        public double getAverageSpeed() {
                // TODO Auto-generated method stub
                Iterator<Entry<Integer, Car>> cit = this.cWorld.getCars().entryS
et()
                                .iterator();
                float sum = 0;
                float count = 0;
                while (cit.hasNext()) {
                        Car currentCar = cit.next().getValue();
                        sum += currentCar.getCurrentSpeed();
                        count++;

                }
                if (count == 0)
                        return 0;
                else
                        return sum / count;
        }

        public int findLight(Point p) {
                ArrayList<TrafficLight> lights = this.cWorld.getLights();
                TrafficLight selected = null;
                for (int i = 0; i < lights.size(); i++) {
                        TrafficLight cl = lights.get(i);
                        if ((p.x > cl.getCoordinate().x - 7.5 && p.x < cl.getCoo
rdinate().x + 7.5)
                                        && (p.y > cl.getCoordinate().y - 7.5 &&
```

```
p.y < cl
                                                        .getCoordinate().y + 7.5
)) {
                                selected = cl;
                                break;
                        }
                }
                if (selected == null)
                        return 0;

                return selected.getId();

        }

        public ArrayList<TrafficLight> getLights() {
                // TODO Auto-generated method stub
                return this.cWorld.getLights();
        }

        public ArrayList<CarPark> getParks() {
                // TODO Auto-generated method stub
                return this.cWorld.getParks();
        }

        public void reset() {
                this.cWorld.reset();
        }

        public Integer findLane(Point p) {
                // TODO Auto-generated method stub
                ArrayList<Lane> lanes = this.cWorld.getLanes();

                for (int i = 0; i < lanes.size(); i++) {
                        Lane cl = lanes.get(i);
                        if ((p.x > cl.getStart().x - 7.5 && p.x < cl.getStart().
x + 7.5)
                                        && (p.y > cl.getStart().y - 7.5 && p.y <
 cl.getStart().y + 7.5)) {
                                return cl.getLaneId();
                        }
                }
                return null;
        }

        public ArrayList<Lane> getLanes() {
                return this.cWorld.getLanes();
        }
}
```

```java
package model;

import java.awt.geom.Point2D;

//Util clas
public class linesolver {

        /*
         * we assume that there are two points p1,p2 belonging to line1 two poin
ts
         * p3,p4 belonging to line2 and we need to calculate whether there is an
         * intersection of these two lines
         */

        public static Point2D.Float checkIntersection(Point2D.Float p1,
                        Point2D.Float p2, Point2D.Float p3, Point2D.Float p4)
                        throws UnknownConnectionError {
                Float f = (p1.x - p2.x) * (p3.y - p4.y) - (p1.y - p2.y) * (p3.x
 - p4.x);
                if (f == 0)
                        throw new UnknownConnectionError();
                Float xi = ((p3.x - p4.x) * (p1.x * p2.y - p1.y * p2.x) - (p1.x
 - p2.x)
                                * (p3.x * p4.y - p3.y * p4.x))
                                / f;
                Float yi = ((p3.y - p4.y) * (p1.x * p2.y - p1.y * p2.x) - (p1.y
 - p2.y)
                                * (p3.x * p4.y - p3.y * p4.x))
                                / f;

                Point2D.Float intersection = new Point2D.Float(xi, yi);

                return intersection;
        }

}
```