```
        package londonsw;

        import londonsw.controller.StartUpController;

/**
 * This is the main class that first gets executed by the system
 * It starts the View components (and from there will branch to either simulatio
n or map-making)
 */
public class SystemApp {
        public static void main(String[] args) throws Exception {
                StartUpController sc = StartUpController.getInstance();
                sc.startSoftware(args);
        }
}
```

```
package londonsw.controller;


import londonsw.model.simulation.components.Intersection;
import londonsw.model.simulation.components.vehicles.Vehicle;
import londonsw.view.simulation.IntersectionDecorator;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;


public class IntersectionController {
    private static Map<Intersection,IntersectionDecorator> intersections = new H
ashMap<Intersection,IntersectionDecorator>();
    private static ArrayList<Intersection> allIntersections = new ArrayList<>();

    /**
     * Prevents instantiation of this class
     */
    protected IntersectionController() {  }

    /**
     * Register an instance of an Intersection to and IntersectionDecorator, and
 adds the Intersection to a list to keep
     * track of
     * @param i the instance of Intersection to register
     * @param gui the instance of IntersectionDecorator for that intersection
     */
    public static void addIntersectionAndDecoratorPair(Intersection i, Intersect
ionDecorator gui) {
        intersections.put(i,gui);
        allIntersections.add(i);
    }

    /**
     * Retrieve the decorator for that specific Intersection
     * @param i the Intersection to get the decorator for
     * @return the decorator associated with that Intersection
     */
    public static IntersectionDecorator getIntersectionDecoratorForIntersection(
Intersection i) {
        return intersections.get(i);
    }

    /**
     * Gets all Intersections in the system
     * @return ArrayList of all Intersections in the system
     */
    public ArrayList<Intersection> getAllIntersections() {
        return allIntersections;
    }


    /**
     * This method is used for determining which vehicle will turn first in an i
ntersection. By generating priorities,
     * this will prevent cars from driving into each other in intersections.
     * @param intersection the intersection where the vehicles are
```

```
     * @param vehicles array list of vehicles with priorities set
     * @return true if this method succeeds (always true)
     * @throws Exception
     */
    public static boolean vehicleTurnFirst(Intersection intersection, ArrayList<
Vehicle> vehicles) throws Exception {
        ArrayList<Integer>  randomPriority = (ArrayList < Integer>) intersection
.generateRandom().clone();
        intersection.vehicleTurnFirst(intersection.giveVehiclePriorities(randomP
riority));

        return true;
    }


}
```

```
package londonsw.controller;

import javafx.stage.Stage;
import londonsw.view.mapcreation.ComponentType;
import londonsw.view.mapcreation.MapMakerScreen;

/**
 * The controller for the Map making aspect of this software
 */
public class MapMakerController {

    private int width;
    private int height;
    private Stage primaryStage;

    private static ComponentType currentFocused;
    private static ComponentType previousFocused;

    /**
     * Creates an instance of a MapMakerController with a stage in which to draw
 the Map making screen
     * @param primaryStage the stage in which to draw the Map making screen
     */
    public MapMakerController(Stage primaryStage) {
        this.primaryStage = primaryStage;
    }

    /**
     * Tells this controller what width and height the user chose for their map
     * @param width the width of the user's new map
     * @param height the height of the user's new map
     */
    public void setWidthAndHeight(int width, int height) {
        this.width = width;
        this.height = height;
    }

    /**
     * Draws the screen using the stage given and displays it to the user
     * @throws Exception
     */
    public void drawScreen() throws Exception{
        MapMakerScreen mapMakerScreen = new MapMakerScreen(width, height);
        mapMakerScreen.drawScreen(primaryStage);
    }

    /**
     * Get the width the user chose for their new map
     * @return width of the map
     */
    public int getWidth() {
        return width;
    }

    /**
     * Get the height the user chose for their map
     * @return height of the map
     */
    public int getHeight() {
```

```
        return height;
    }

    /**
     * Gets what the current focused is in the screen. This is what the user cli
cked last in the screen. It is of type
     * ComponentType enum, which can be a RoadNS image, Map_Square, Nothing, etc
     * @return ComponentType enum of what the user clicked last
     */
    public static ComponentType getCurrentFocused() {
        return currentFocused;
    }

    /**
     * Sets the current focused in the map, this is set after the user clicks so
mething
     * @param focused what was last clicked on by the user, converted to type en
um ComponentType
     */
    public static void setCurrentFocused(ComponentType focused) {
        currentFocused = focused;
    }

    /**
     * Gets what the user previously clicked (the click before the current click
)
     * @return ComponentType of what was clicked on the click before the last
     */
    public static ComponentType getPreviousFocused() {
        return previousFocused;
    }

    /**
     * Sets the previously clicked before last item
     * @param prev sets the click before the last click to what was focused, con
verted to type ComponentType
     */
    public static void setPreviousFocused(ComponentType prev) {
        previousFocused = prev;
    }

}
```

```
package londonsw.controller;

import javafx.stage.Stage;
import londonsw.model.simulation.Log;
import londonsw.model.simulation.Map;
import londonsw.view.simulation.SimulationScreen;

import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * This is the controller for simulations in our View. It draws the screen and g
enerates the Log for the simulation.
 */
public class SimulationController {

    private Stage primaryStage;
    private String mapName;
    private String logFileName;

    /**
     * Creates a new SimulationController to control the simulation
     * @param primaryStage the stage in which to display the new screen
     */
    public SimulationController(Stage primaryStage) {
        this.primaryStage = primaryStage;
    }

    /**
     * Sets the name of the map that the user wants to open
     * @param mapName name of map file that user wants to simulate vehicles on
     */
    public void setMapName(String mapName) {
        this.mapName = mapName;
    }

    /**
     * Draws the screen where the simulation will be taking place, and creates t
he Log to log all activity
     * in the simulation
     * @throws Exception
     */
    public void drawScreen() throws Exception {
        Map map = Map.loadMap(mapName);
        SimulationScreen screen = new SimulationScreen(map);
        screen.drawScreen(primaryStage);
        generateLogFileName();
        Log log = new Log(getLogFileName());
    }

    /**
     * Get the name of the log file for that specific simulation
     * @return
     */
    public String getLogFileName() {
        return logFileName;
    }

    /**
```

```
     * Generates a file name for the log file for that simulation. It is in the
format
     * Log_DATE.log, where DATE is the YEAR-MONTH-DAY-HOUR-MINUTES-SECONDS
     */
    public void generateLogFileName() {
        String date = new SimpleDateFormat("yyyy-MM-dd-HH-mm-ss").format(new Dat
e());
        logFileName = "Log_" + date;
    }

}
```

```
package londonsw.controller;


import javafx.application.Application;
import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXMLLoader;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import java.io.File;

import javafx.stage.StageStyle;
import javafx.util.Pair;
import londonsw.model.simulation.Ticker;

//import java.awt.*;
import java.io.IOException;
import java.util.Optional;

/**
 *  This is the controller that gets called by the main SystemApp class. This co
ntroller initiates all GUI screens.
 */
@SuppressWarnings("Duplicates")
public class StartUpController extends Application {

    private static StartUpController instance;

    public StartUpController() { }

    public static StartUpController getInstance() {
        if(instance == null)
            instance = new StartUpController();
        return instance;
    }

    public void startSoftware(String[] args) {
        launch(args);
    }

    /**
     * This is the first method that gets called in the system. It loads the Sta
rtScreen fxml file, which contains
     * the START button.
     * @param primaryStage the stage that initially loads
     * @throws Exception
     */
```

```
    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("LondonSW Traffic Simulator");

        VBox vBox = new VBox();
        vBox.setPrefSize(600,400);
        vBox.setSpacing(10);
        vBox.setStyle("-fx-background-color:papayawhip");
        vBox.setAlignment(Pos.CENTER);

        Label londonSWLabel = new Label("London SW");
        londonSWLabel.setFont(Font.font("System Bold Italic", FontWeight.BOLD, 20));
        Label trafficSimLabel = new Label("Traffic Simulator");
        trafficSimLabel.setFont(Font.font("System Bold Italic", FontWeight.EXTRA_BOLD
, 22));
        trafficSimLabel.setPadding(new Insets(0,0,50,0));
        Button startButton = new Button("Start");
        startButton.setPrefSize(300,150);
        startButton.setStyle("-fx-base:Gold");
        startButton.setFont(Font.font("System Bold Italic", FontWeight.EXTRA_BOLD, 26
));

        vBox.getChildren().add(londonSWLabel);
        vBox.getChildren().add(trafficSimLabel);
        vBox.getChildren().add(startButton);

        Scene scene = new Scene(vBox);
        primaryStage.setScene(scene);
        primaryStage.setResizable(false);
        primaryStage.show();
        primaryStage.centerOnScreen();

        startButton.setOnMouseClicked(click -> {
            goToChooseModeScreen(primaryStage);
        });
    }

    /**
     * This is the method that gets called when the user hits the START button o
n the initial screen. It
     * loads the "Choose Mode" screen, which gives 2 options: Opening a pre-made
 map, or users building their own map.
     * @param primaryStage the stage to display the screen in
     */
    public void goToChooseModeScreen(Stage primaryStage) {
        VBox vBox = new VBox();
        vBox.setPrefSize(600,400);
        vBox.setSpacing(50);
        vBox.setStyle("-fx-background-color:papayawhip");
        vBox.setAlignment(Pos.CENTER);
        Platform.runLater(() -> vBox.requestFocus());

        Button openMap = new Button("Open a Pre-made Map");
        openMap.setPrefSize(300, 90);
        openMap.setStyle("-fx-base:Gold");
        openMap.setFont(Font.font("System Bold Italic", FontWeight.EXTRA_BOLD, 16));

        Button makeMap = new Button("Make a new Map");
        makeMap.setPrefSize(300, 90);
```

```
        makeMap.setStyle("-fx-base:Gold");
        makeMap.setFont(Font.font("System Bold Italic", FontWeight.BOLD, 16));

        vBox.getChildren().add(openMap);
        vBox.getChildren().add(makeMap);

        Scene scene = new Scene(vBox);
        primaryStage.setScene(scene);

        openMap.setOnMouseClicked(click -> {
            try {
                goToSimulationMode(primaryStage);
            } catch (Exception e) {
                e.printStackTrace();
            }
        });

        makeMap.setOnMouseClicked(click -> {
            goToMapMakerMode(primaryStage);
        });
    }

    /**
     * When the user click "Choose Pre-made map..." button, it will go to Simula
tionMode screen. It will first
     * prompt the user to open a file (only Map files are allowed to be opened),
 set a ticker interval speed,
     * and then it will go to draw the simulation mode screen.
     * @param primaryStage the click that caused this method invocation
     * @throws Exception
     */
    public void goToSimulationMode(Stage primaryStage) throws Exception {
        FileChooser chooser=new FileChooser();
        chooser.setTitle("Open Map");
        chooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("Map Fi
le (*.map)", "*.map"));
        File file = chooser.showOpenDialog(new Stage());


        if(file!=null)
        {
//            String mapName = file.getName();
            String mapName = file.getAbsolutePath();


            Dialog<Long> dialog = new Dialog<>();
            dialog.setTitle("Choose Ticker Interval Duration");
            dialog.setHeaderText("Choose a duration (in milliseconds) for\nthe ticker in the system.")
;
            dialog.setGraphic(null);
            dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK);
            dialog.initStyle(StageStyle.UNDECORATED);

            GridPane grid = new GridPane();
            grid.setHgap(10);
            grid.setVgap(10);
            grid.setPadding(new Insets(20, 80, 10, 10));
            grid.add(new Label("Duration: "), 0, 0);
            Spinner<Double> spinner = new Spinner<Double>(100, 2000, Ticker.getT
```

```
ickInterval(), 100);
            grid.add(spinner, 1, 0);
            dialog.getDialogPane().setContent(grid);
            Platform.runLater(() -> spinner.requestFocus());

            dialog.setResultConverter(dialogButton -> {
                if(dialogButton == ButtonType.OK) {
                    double value =  spinner.getValue();
                    return (long) value;
                }
                return null;
            });

            Optional<Long> result = dialog.showAndWait();
            result.ifPresent(aLong -> {
                Ticker.setTickInterval(aLong);
                Ticker.start();
            });

            //Decorate map to extend to GUI functionality
            SimulationController simulationController = new SimulationController
(primaryStage);
            simulationController.setMapName(mapName);
            simulationController.drawScreen();
        }
    }

    /**
     * This method gets called when the user chooses to go to Map Maker mode. It
 will prompt the user for
     * the width and height that they want for their new map, in the range from
5 to 30 for both width and
     * height. It will then bring the user to the screen where they can build th
e map.
     * @param primaryStage the click event that caused this method invocation
     */
    public void goToMapMakerMode(Stage primaryStage) {
        Dialog<Pair<String, String>> dialog = new Dialog<>();
        dialog.setTitle("Choose Map Size");
        dialog.setHeaderText("Choose new map's width and height");
        dialog.setGraphic(null);

        GridPane grid = new GridPane();
        grid.setHgap(10);
        grid.setVgap(10);
        grid.setPadding(new Insets(20, 150, 10, 10));

        dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK, ButtonType
.CANCEL);

        ObservableList<Integer> choices = FXCollections.observableArrayList();
        for(int i = 5; i <= 30; i++) {
            choices.add(i);
        }

        ChoiceBox<Integer> widthBox = new ChoiceBox<>();
        widthBox.setItems(choices);
        widthBox.setMinWidth(100);
        Platform.runLater(() -> widthBox.requestFocus());
```

```
        ChoiceBox<Integer> heightBox = new ChoiceBox<>();
        heightBox.setItems(choices);
        heightBox.setMinWidth(100);

        grid.add(new Label("Width:"), 0, 0);
        grid.add(widthBox, 1, 0);
        grid.add(new Label("Height:"), 0, 1);
        grid.add(heightBox, 1, 1);

        dialog.getDialogPane().setContent(grid);

        Button doneBtn = (Button) dialog.getDialogPane().lookupButton(ButtonType
.OK);
        doneBtn.setDisable(true);
        doneBtn.disableProperty().bind(
                widthBox.valueProperty().isNull()
                .or(heightBox.valueProperty().isNull())
        );

        dialog.setResultConverter(dialogButton -> {
            if(dialogButton == ButtonType.OK) {
                return new Pair<>(widthBox.getValue().toString(),heightBox.getVa
lue().toString());
            }
            return null;
        });

        Optional<Pair<String, String>> result = dialog.showAndWait();

        result.ifPresent(widthAndHeight -> {
            int width = Integer.parseInt(widthAndHeight.getKey());
            int height = Integer.parseInt(widthAndHeight.getValue());

            MapMakerController mapMakerController = new MapMakerController(prima
ryStage);
            mapMakerController.setWidthAndHeight(width,height);
            try {
                mapMakerController.drawScreen();
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
    }

}
```

```
package londonsw.controller;

import londonsw.model.simulation.components.LightColour;
import londonsw.model.simulation.components.TrafficLight;
import londonsw.view.simulation.TrafficLightDecorator;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

/**
 * Controls all the traffic lights. This gets notified when the traffic light ch
anges in the model, and it
 * does the necessary work to change the GUI to display the new model.
 */
public class TrafficLightController {

    private Map<TrafficLight, TrafficLightDecorator> trafficLights;
    private ArrayList<TrafficLight> allLights;
    private long DURATION;
    private boolean lightsEnabled;

    private static TrafficLightController instance;

    /**
     * Creates a new instance of a TrafficLightController. It is set to protecte
d so that the creation of this
     * controller is controlled. This class follows the singleton pattern, so th
ere can be at most one instance
     * of this class in the system.
     */
    protected TrafficLightController() {
        trafficLights = new HashMap<>();
        allLights = new ArrayList<>();
        DURATION = 3000;
        lightsEnabled = true;
    }

    /**
     * Gives an instance of this class. This is what needs to be called if an in
stance is needed. If there is
     * not an instance created, it creates a new one, otherwise it returns an ex
isting instance.
     * @return
     */
    public static TrafficLightController getInstance() {
        if(instance == null) {
            instance = new TrafficLightController();
        }
        return instance;
    }

    /**
     * Gets whether the traffic lights are enabled or not
     * @return true if all traffic lights are working and enabled, false otherwi
se
     */
    public boolean areLightsEnabled() {
        return lightsEnabled;
```

```
    }

    /**
     * Sets the lights boolean to true. Useful if re-loading a map and wanting t
o go back to default state.
     */
    public void setLightsToEnabled() {
        lightsEnabled = true;
    }

    /**
     * Disables or enables all traffic lights in the system. If traffic lights a
re disabled, cars will
     * move freely without stopping anywhere. If enabled, they listen to the tra
ffic lights.
     * @param disable true disables the lights, false enables the lights
     */
    public void disableLights(boolean disable) {
        if(disable) {
            lightsEnabled = false;
            for(TrafficLight t : allLights) {
                TrafficLightDecorator decorator = trafficLights.get(t);
                if(decorator != null)
                    decorator.hideCircle(true);
            }
        }
        else {
            lightsEnabled = true;
            for(TrafficLight t : allLights) {
                TrafficLightDecorator decorator = trafficLights.get(t);
                if(decorator != null)
                    decorator.hideCircle(false);
            }
        }
    }

    /**
     * Creates a new TrafficLightDecorator for the given TrafficLight
     * @param tl the TrafficLight for which to create the decorator
     * @return the newly created decorated for that TrafficLight
     */
    public TrafficLightDecorator createNewDecorator(TrafficLight tl) {
        return new TrafficLightDecorator(tl);
    }

    /**
     * Called by the TrafficLight (in the model) when the color changes. It tell
s the corresponding
     * TrafficLightDecorator to change its color in the GUI.
     * @param colour the new color to be
     * @param tl a TrafficLight that had its color changed
     */
    public void colourChanged(LightColour colour, TrafficLight tl) {
        if(trafficLights.get(tl) == null) {
            return;
        }
        trafficLights.get(tl).setGUIColour(colour);
    }
```

```
    /**
     * Register a TrafficLight (in the model) to a TrafficLightDecorator (in the
 GUI), and add to an
     * ArrayList to keep track of all lights
     * @param tl the TrafficLight instance from the model
     * @param gui the corresponding instance of the TrafficLightDecorator for th
at TrafficLight
     */
    public void addTrafficLightAndDecoratorPair(TrafficLight tl, TrafficLightDec
orator gui) {
        trafficLights.put(tl,gui);
        allLights.add(tl);
    }

    /**
     * If you need to get the decorator for that TrafficLight outside of this cl
ass, use this method
     * @param tl a TrafficLight that you are querying for to get its decorator
     * @return the corresponding TrafficLightDecorator for that TrafficLight
     */
    public TrafficLightDecorator getTrafficLightGUI(TrafficLight tl) {
        return trafficLights.get(tl);
    }

    /**
     * Get a list of all the TrafficLights in the system
     * @return ArrayList of all lights
     */
    public ArrayList<TrafficLight> getAllTrafficLights() {
        return allLights;
    }

    /**
     * Gets the duration length for traffic lights. Traffic lights call this met
hod to know how long to be a certain color.
     * @return length of duration in millis
     */
    public long getDurationLength() {
        return DURATION;
    }

    /**
     * Sets the duration length for traffic lights
     * @param duration the duration for the traffic lights in millis
     */
    public void setDurationLength(long duration) {
        this.DURATION = duration;
    }

    /**
     * Set the duration of all the traffic lights in the map
     * @param duration the duration of the traffic light in milliseconds
     */
    public void setTrafficLightDuration(long duration) {
        for(TrafficLight t : allLights) {
            t.setDuration(duration);
        }
    }
```

```
    /**
     * Gets the HashMap of all TrafficLight,TrafficLightDecorator pairs
     * @return a Map of all TrafficLight, TrafficLightDecorator pairs
     */
    public Map getTrafficLightsMap() {
        return trafficLights;
    }

}
```

```
package londonsw.controller;


import londonsw.model.simulation.components.Lane;
import londonsw.model.simulation.components.vehicles.Vehicle;
import londonsw.view.simulation.VehicleGUIDecorator;


import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

/**
 * This is the controller for all vehicle movement. It keeps track of all vehicl
es and their decorators. It also
 * has the methods that facilitates vehicle movement in the model and in the map
.
 */
public class VehicleController {

    private static Map<Vehicle, VehicleGUIDecorator> vehiclesAndDecorators = new
 HashMap<>();
    private static ArrayList<Vehicle> allVehicles = new ArrayList<>();

    /**
     * Register a car to a specific CarGuiDecorator so we can retrieve it and dr
aw the decorator for
     * for that car. This replaces the need to pass the CarGuiDecorator as a par
ameter. This also adds
     * the vehicle to an ArrayList, only to keep track of all cars in the system
 and nothing more.
     *
     * @param vehicle   an instance of a Vehicle type
     * @param decorator the CarGuiDecorator for that specific vehicle
     */
    public static void addVehicleAndDecoratorPair(Vehicle vehicle, VehicleGUIDec
orator decorator) {
        vehiclesAndDecorators.put(vehicle, decorator);
        allVehicles.add(vehicle);
    }

    /**
     * Removes the vehicle instance from the list of all vehicles and the map of
 all vehicle,decorator pairs
     * @param v the vehicle to remove
     */
    public static void removeFromListAndMap(Vehicle v) {
        allVehicles.remove(v);
        vehiclesAndDecorators.remove(v);
    }

    /**
     * Gets all the vehicles in the system
     * @return ArrayList with all vehicles in the system
     */
    public static ArrayList<Vehicle> getVehicleList(){
        return allVehicles;
    }
```

```
    /**
     * Given an index in the array, this removes the vehicle from existence.
     * @param index the index in the list allVehicles that the vehicle occupies
     */
    public static void removeVehicle(int index) {
        try {
            Vehicle v = allVehicles.get(index);
            VehicleGUIDecorator decorator = vehiclesAndDecorators.get(v);
            decorator.getPane().getChildren().remove(decorator.getRectangle());
            v.setVehicleState(3);
            v.unsubscribe();
            Lane currLane = v.getCurrentLane();
            int currCell = v.getCurrentCell();
            currLane.setCell(null, currCell);
            allVehicles.remove(index);
            vehiclesAndDecorators.remove(v);
            v = null;
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Gets the total time spent standing for all vehicles in the system.
     * @return the total time spent standing by all vehicles in the system
     */
    public static int getTotalTimeSpentStanding() {
        int sum = 0;
        for(Vehicle v : allVehicles) {
            sum += v.getTimeSpentStanding();
        }
        return sum;
    }

    /**
     * Gets the total times ticked by all vehicles in the system. This is used i
n the calculation of
     * average time spent standing in the system.
     * @return
     */
    public static int getTotalTimesTicked() {
        int sum = 0;
        for(Vehicle v : allVehicles) {
            sum += v.getTimesTicked();
        }
        return sum;
    }

    /**
     * Retrieve the VehicleGGUIDecorator for the vehicle, for operations that ha
ppen outside this class
     *
     * @param vehicle the Vehicle to retrieve the decorator for
     * @return CarGuiDecorator instance associated with that specific Vehicle
     */
    public static VehicleGUIDecorator getDecoratorForVehicle(Vehicle vehicle) {
        return vehiclesAndDecorators.get(vehicle);
    }
```

```
    /**
     * This is the method that gets called by the Vehicle (in the model) when th
e ticker ticks. This controller
     * handles the rest of the moving.
     *
     * @param v    the Vehicle that notified that it should move
     * @param step how far the vehicle should move
     * @throws Exception
     */
    public static void moveOnTick(Vehicle v, int step) throws Exception {
        VehicleGUIDecorator decorator = vehiclesAndDecorators.get(v);
        moveVehicle(decorator, step);
    }

    /**
     * Moves the vehicle in the model and in the GUI
     * @param vehicleGUIDecorator the GUI decorator for this vehicle
     * @param step how many slots to move
     * @throws Exception
     */
    public static void moveVehicle(VehicleGUIDecorator vehicleGUIDecorator, int
step) throws Exception {

        int move = 0;

        vehicleGUIDecorator.setPreviousLane(vehicleGUIDecorator.getCurrentLane()
);
        vehicleGUIDecorator.setPreviousCoordinate(vehicleGUIDecorator.getCurrent
Coordinate());

        // Vehicle is at an intersection
        if (vehicleGUIDecorator.getCurrentCoordinate().equals(vehicleGUIDecorato
r.getCurrentLane().getExit())) {
            //only read when intersection is available
            vehicleGUIDecorator.readTrafficLight();

            if (vehicleGUIDecorator.getVehicleState() == 1) { // if vehicle was
moving
                // get next lane available to move to
                Lane l = vehicleGUIDecorator.chooseLane();

                vehicleGUIDecorator.setVehicleState(2); // set vehicle's state t
o "in intersection"
                move = vehicleGUIDecorator.vehicleTurn(l); // move the vehicle i
n the model and get a result int
            }
        } else {
            if (vehicleGUIDecorator.getVehicleState() != 0) { // if not at inter
section, and wasn't stopped, move forward
                move = vehicleGUIDecorator.moveVehicle(step);
            }
        }

        if(move == 0) {
            Vehicle thisVehicle = vehicleGUIDecorator.getVehicle();
            thisVehicle.incrementTimeSpentStanding();
        }

        if (vehicleGUIDecorator.getVehicleState() == 3) { //vehicle is deleted j
```

```
ust move to next space
            vehicleGUIDecorator.moveVehicleGUI(move, vehicleGUIDecorator.getVehi
cleState());
        }
        else if(move>0 && vehicleGUIDecorator.getVehicleState()!=0) { // move th
e vehicle in the GUI
            vehicleGUIDecorator.moveVehicleGUI(move, vehicleGUIDecorator.getVehi
cleState());
        }
    }
}
```

```
package londonsw.model.simulation;

import londonsw.model.simulation.components.TrafficLight;
import londonsw.model.simulation.components.vehicles.Vehicle;
import rx.Subscriber;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.logging.FileHandler;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;


 /**
  * Created by felix on 18/03/2016.
   * Logs what is happening in the system for every tick.
  */

public class Log extends Subscriber<Long> {

     private String fileName;
     private final String LOG_DIR = "./logs/";
     private String filePath;

     /**
      * Creates a log with the given file name to log what is happening in the s
ystem. It will be stored in the
      * directory LOG_DIR.
      * @param fileName the file name for the log
      */
     public Log(String fileName) {
         this.fileName = fileName;

         File directory = new File(LOG_DIR);
         if(!directory.exists()) {
             directory.mkdir();
         }

         filePath = LOG_DIR + fileName + ".log";
         File logFile = new File(filePath);
         if(!logFile.exists()) {
             try {
                 logFile.createNewFile();
             } catch (IOException e) {
                 e.printStackTrace();
             }
         }

         Ticker.subscribe(this);
     }

     /**
      * Generates log information for that Ticker tick
      * @param aLong current time in the system
      */
     private void generate(long aLong) {

         ArrayList al = Ticker.getSubscribers();
```

```
        Logger logger = Logger.getLogger("SIMULATION");

        FileHandler fh;

        try {

            StringBuilder sb = new StringBuilder();

            sb.append("TICK!:" + aLong);
            sb.append(System.lineSeparator());

            for (Object o : al) {
                if (o instanceof Vehicle) {

                    Vehicle vLog = (Vehicle) o;

                    sb.append(System.lineSeparator());
                    sb.append("---------VEHICLE---------");
                    sb.append(System.lineSeparator());
                    sb.append("ID:" + vLog.getId());
                    sb.append(System.lineSeparator());
                    sb.append("CURRENT LANE ID:" + vLog.getCurrentLane().getId(
));
                    sb.append(System.lineSeparator());
                    sb.append("CURRENT COORDINATES:" + vLog.getCurrentCoordina
te().getX() + "," + vLog.getCurrentCoordinate().getY());
                    sb.append(System.lineSeparator());

                    if (vLog.getPreviousLane() != null) {
                        sb.append("PREVIOUS LANE ID:" + vLog.getPreviousLane().
getId());
                        sb.append(System.lineSeparator());
                        sb.append("PREVIOUS LANE COORDINATES:" + vLog.getPrevi
ousCoordinate().getX() + "," + vLog.getCurrentCoordinate().getY());
                        sb.append(System.lineSeparator());
                    }

                    sb.append("CURRENT CELL:" + vLog.getCurrentCell());
                    sb.append(System.lineSeparator());
                    sb.append("BEHAVIOUR:" + vLog.getVehicleBehavior());
                    sb.append(System.lineSeparator());
                    sb.append("PRIORITY:" + vLog.getVehiclePriority());
                    sb.append(System.lineSeparator());
                    sb.append("STATE:" + vLog.getVehicleState());
                    sb.append(System.lineSeparator());
                    sb.append("SUBSCRIBED:" + (vLog.isUnsubscribed()==false?"Y
ES":"NO"));


                } else if (o instanceof TrafficLight) {
                    TrafficLight tLog = (TrafficLight) o;
                    sb.append(System.lineSeparator());
                    sb.append("------TRAFFIC LIGHT------");
                    sb.append(System.lineSeparator());
                    sb.append("ID:" + tLog.getId());
                    sb.append(System.lineSeparator());
                    sb.append("DURATION:" + tLog.getDuration());
                    sb.append(System.lineSeparator());
```

```
                    sb.append("STATE: " + tLog.getState());
                }
            }

            sb.append(System.lineSeparator());
            sb.append("============================================================
=");

            fh = new FileHandler(filePath,true);

            logger.addHandler(fh);
            SimpleFormatter formatter = new SimpleFormatter();
            fh.setFormatter(formatter);
            logger.setUseParentHandlers(false);
            logger.info(sb.toString());
            fh.close();

        } catch (SecurityException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onCompleted() {  }

    /**
     * If there's some error with the ticker and this subscriber, this method w
ould call.
     * Left not implemented on purpose
     * @param throwable
     */
    @Override
    public void onError(Throwable throwable) {    }

    /**
     * Called on Ticker tick, will generated a log entry for that tick
     * @param aLong current time in the sytem
     */
    @Override
    public void onNext(Long aLong) {
       generate(aLong);
    }
}
```

```
package londonsw.model.simulation;

import londonsw.controller.TrafficLightController;
import londonsw.model.simulation.components.*;
import londonsw.view.simulation.IntersectionDecorator;
import londonsw.view.simulation.TrafficLightDecorator;

import java.io.*;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Random;

/**
 * This is the graph structure that our map holds (Roads and Intersections)
 */
public class Map implements Serializable {
    private static final long serialVersionUID = -1932129809569954013L;
    private ArrayList<Road> roads;
    private ArrayList<Intersection> intersections;
    private MapGrid grid;
    private final static String MAP_DIR = "./maps/";


    /**
     * Creates an empty map with no roads or intersections
     *
     * @param width width of the map
     * @param height height of the map
     */
    public Map(int width, int height) {
        roads = new ArrayList<Road>();
        intersections = new ArrayList<Intersection>();
        grid = new MapGrid(width,height);
    }

    /**
     * Gets the width of the map
     *
     * @return integer representing width of the map
     */
    public int getWidth() {
        return grid.getWidth();
    }

    /**
     * Gets the height of the map
     *
     * @return integer representing the height of the map
     */
    public int getHeight() {
        return grid.getHeight();
    }

    /**
     * Gets all the roads currently in the map
     * @return ArrayList of all Roads in the map
     */
    public ArrayList<Road> getRoads() {
```

```
        return roads;
    }

    /**
     * Gets a random road from the Map
     * @return a random Road instance from the current map
     */
    public Road getRandomRoad() {
        ArrayList<Road> roads = getRoads();
        Road road = null;
        Random randomRoad = new Random();

        if(roads.size() > 0) {
            int roadSize = randomRoad.nextInt(this.getRoads().size());
            road = roads.get(roadSize);
        }

        return road;
    }

    /**
     * Gets a random lane from the Map
     * @return a random Lane instance from the current Map
     */
    public Lane getRandomLane() {
        Road road = getRandomRoad();
        Lane lane = null;

        if (road != null) {
            Random randomLane = new Random();
            int numberLanes = road.getNumberLanes();

            if (numberLanes > 0) {
                int laneSize = randomLane.nextInt(road.getNumberLanes());
                lane = road.getLanes().get(laneSize);
                while (lane.getState() != 1) { // if road is disabled, choose a
new one
                    road = getRandomRoad();
                    lane = road.getLanes().get(randomLane.nextInt(road.getNumber
Lanes()));
                }
            }
        }

        return lane;
    }

    /**
     * Gets a random cell from a random Lane from the system, which is an int. A
 vehicle can go in this cell.
     * @return index of random cell from a random Lane in the Map
     */
    public int getRandomCell(){
        Random randomCell = new Random();
        Lane randomLane = getRandomLane();
        return randomCell.nextInt(randomLane.getLength());
    }

    /**
```

```
     * If you have a list of roads already, set the roads to the map
     * @param roads an ArrayList of valid Road instances
     */
    public void setRoads(ArrayList<Road> roads) {
        this.roads = roads;
        for(Road r : roads)
            grid.addComponent(r);
    }

    /**
     * Gets all the intersections currently in the map
     * @return ArrayList of all intersections in the map
     */
    public ArrayList<Intersection> getIntersections() {
        return intersections;
    }

    /**
     * If you have a list of intersections already, set the intersections to the
map
     * @param intersections
     */
    public void setIntersections(ArrayList<Intersection> intersections) {
        this.intersections = intersections;
        for(Intersection i : intersections)
            grid.addComponent(i);
    }

    /**
     * Gets the underlying MapGrid of the Map and returns it
     * @return the actual MapGrid of this map
     */
    public MapGrid getGrid() {
        return grid;
    }

    /**
     * If you have a valid MapGrid grid, set it as the grid to this Map
     * @return valid instance of MapGrid
     */
    public void setGrid(MapGrid grid) {
        this.grid = grid;
    }

    /**
     * Adds a brand new road to this map.
     * @param r valid Road instance to put into the Map
     */
    public void addRoad(Road r) {
        roads.add(r);
        grid.addComponent(r);
    }

    public Component getAtLocation(Coordinate c) {
        int x = c.getX();
        int y = c.getY();
        return grid.get(x,y);
    }
```

```
    /**
     * Adds a brand new intersection to this map
     * @param i valid Intersection instance to put into the Map
     */
    public void addIntersection(Intersection i) {
        intersections.add(i);
        grid.addComponent(i);
    }

    /**
     * Removes the map Component at the given coordinate (only deletes it from t
he model, not the view)
     * @param c the coordinate where to delete the Map component
     */
    public void clearCell(Coordinate c) {
        int x = c.getX();
        int y = c.getY();
        grid.clearCell(x, y);
    }

    /**
     * Saves the map to the disk. Given a file name, it saves the map into MAP_D
IR with that file name
     * @param fileName the file name for saving the map
     */
    public void saveMap(String fileName)
    {
        try
        {
            File directory = new File(MAP_DIR);
            if(!directory.exists()) {
                directory.mkdir();
            }

            String path = MAP_DIR + fileName;

            File file = new File(path);
            if(!file.exists()) {
                file.createNewFile();
            }

            // save the map
            FileOutputStream fileOut = new FileOutputStream(path);
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(this);
            out.close();
            fileOut.close();
        }catch(IOException i)
        {
            i.printStackTrace();
        }
    }

    /**
     * Loads a map from the disk with the given file name. It unserializes it an
d returns an instance of a Map. This
     * method is static so it can be called without any instances of a Map.
     * @param fileName the name of the map file to open (extension included)
     * @return an instance of a loaded Map that was opened from the disk
```

```
     */
    public static Map loadMap(String fileName) {
        Map map = null;

        try {
            // open the Map
            FileInputStream fileIn = new FileInputStream(fileName);
            ObjectInputStream in = new ObjectInputStream(fileIn);
            map = (Map) in.readObject();
            in.close();
            fileIn.close();

            /*
             * Each intersection has 4 traffic lights, for each traffic light:
             *      - subscribe to the ticker
             *      - create a new Decorator
             * For each intersection:
             *      - create a new IntersectionDecorator
             *      - link each new TrafficLightDecorator to the corresponding f
ield in the IntersectionDecorator
             * */
            for(Intersection i : map.getIntersections()) {
                i.subscribeToTicker();
                IntersectionDecorator decorator = new IntersectionDecorator(i);
                TrafficLight north = i.getNorthTrafficLight();
                TrafficLight south = i.getSouthTrafficLight();
                TrafficLight east = i.getEastTrafficLight();
                TrafficLight west = i.getWestTrafficLight();
                if(north != null) {
                    north.subscribeToTicker();
                    TrafficLightDecorator dec = TrafficLightController.getInstan
ce().createNewDecorator(north);
                    decorator.setNorthTrafficLightDecorator(dec);
                }
                if(south != null) {
                    south.subscribeToTicker();
                    TrafficLightDecorator dec = TrafficLightController.getInstan
ce().createNewDecorator(south);
                    decorator.setSouthTrafficLightDecorator(dec);
                }
                if(east != null) {
                    east.subscribeToTicker();
                    TrafficLightDecorator dec = TrafficLightController.getInstan
ce().createNewDecorator(east);
                    decorator.setEastTrafficLightDecorator(dec);
                }
                if(west != null) {
                    west.subscribeToTicker();
                    TrafficLightDecorator dec = TrafficLightController.getInstan
ce().createNewDecorator(west);
                    decorator.setWestTrafficLightDecorator(dec);
                }
            }

            long duration = TrafficLightController.getInstance().getDurationLeng
th();
            TrafficLightController.getInstance().setTrafficLightDuration(duratio
n);
            TrafficLightController.getInstance().setLightsToEnabled();
```

```
        } catch(IOException i) {
            System.out.println("IO Exception");
            i.printStackTrace();
        } catch (ClassNotFoundException e) {
            System.out.println("Did not find object");
            e.printStackTrace();
        }

        return map;
    }

    /**
     * For debug only. Prints the map layout to the console.
     */
    public void printMapGrid() {
        Component[][] grid = this.getGrid().getGrid();
        int width = this.getGrid().getWidth();
        int height = this.getGrid().getHeight();
        for(int i = 0; i < height; i++) {
            for(int j = 0; j < width; j++) {
                Component current = grid[i][j];
                if(current instanceof Road)
                    System.out.print("R ");
                else if(current instanceof Intersection)
                    System.out.print("I ");
                else
                    System.out.print("- ");
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

```
package londonsw.model.simulation;

import londonsw.model.simulation.components.*;

import java.io.Serializable;
import java.lang.reflect.Array;
import java.util.ArrayList;

/**
 * This class is the underlying structure of our Map. It is a 2D-array of map Co
mponents,
 * each Component being something that you would want to be displayed on the map
, such as
 * a Road or Intersection.
 */
public class MapGrid implements Serializable, IMapGrid {

    private static final long serialVersionUID = -8256761045077358688L;
    private int width;
    private int height;
    private Component[][] grid;
    private ArrayList<Component> allComponents;

    /**
     * Creates a brand new MapGrid instance to be part of a Map
     * @param width width of the Map that this will be part of
     * @param height height of the Map that this will be part of
     */
    public MapGrid(int width, int height) {
        this.width = width;
        this.height = height;
        grid = new Component[height][width];
        allComponents = new ArrayList<Component>();
    }

    /**
     * Returns the actual 2D-array of Components
     * @return 2D-array array of Components signifying the layout of the Map
     */
    public Component[][] getGrid() {
        return grid;
    }

    /**
     * Gets the width of the grid
     * @return width of the grid
     */
    public int getWidth() {
        return width;
    }

    /**
     * Gets the height of the grid
     * @return height of the grid
     */
    public int getHeight() {
        return height;
    }
```

```
    /**
     * Gets all Components from the MapGrid
     * @return an ArrayList of all components in the Map
     */
    public ArrayList<Component> getAllComponents() { return allComponents; }

    /**
     * Gets a Component from the MapGrid at the given (x, y) location
     * @param x the x coordinate of the requested Component
     * @param y the y coordinate of the requested Component
     * @return a Component at the location (x, y), if there is nothing there, nu
ll is returned
     */
    public Component get(int x, int y) {
        return grid[y][x];
    }

    /**
     * Empties the cell at the given (x, y) location
     * @param x the x-coordinate where to clear the
     * @param y
     */
    public void clearCell(int x, int y) {
        grid[y][x] = null;
    }

    /**
     * Adds a Component to the map grid structure
     * @param c the Component to be added
     * @return true if successfully added, false otherwise
     */
    @SuppressWarnings("Duplicates")
    public boolean addComponent(Component c) {
        if(c instanceof Intersection) {
            Intersection i = (Intersection) c;
            Coordinate coord = i.getLocation();
            grid[coord.getY()][coord.getX()] = i;
            allComponents.add(i);
            return true;
        }
        else if(c instanceof Road) {
            Road road = (Road) c;
            Coordinate start = road.getStartLocation();
            Coordinate end = road.getEndLocation();
            int startX = start.getX();
            int startY = start.getY();
            int endX = end.getX();
            int endY = end.getY();

            if(road.runsVertically()) { // road runs vertically
                if(startY <= endY) { // start coordinate is north of end coordin
ate
                    for(int i = startY; i <= endY; i++) {
                        grid[i][startX] = road;
                    }
                    allComponents.add(road);
                    return true;
                }
                else { // start coordinate is south of end coordinate
```

```
                    for(int i = endY; i <= startY; i++) {
                        grid[i][startX] = road;
                    }
                    allComponents.add(road);
                    return true;
                }
            }
            else { // road runs horizontally
                if(startX <= endX) { // start coordinate is west of end coordina
te
                    for(int i = startX; i <= endX; i++) {
                        grid[startY][i] = road;
                    }
                    allComponents.add(road);
                    return true;
                }
                else {
                    for(int i = endX; i <= startX; i++) { // start coordinate is
 east of end coordinate
                        grid[startY][i] = road;
                    }
                    allComponents.add(road);
                    return true;
                }
            }
        }
        else
            return false;
    }

}
```

```
package londonsw.model.simulation;

/**
 * This is the class that keeps time for our simulation
 *
 * This uses RxJava's Observer and Subscriber to pass events to the classes that
 want messages from the
 * ticker. This also uses RxJavaFx to make sure that the GUI events are running
on the JavaFX thread scheduler.
 */

import rx.Observable;
import rx.Subscriber;
import rx.schedulers.JavaFxScheduler;
import rx.subjects.PublishSubject;

import java.util.ArrayList;
import java.util.concurrent.TimeUnit;

public class Ticker {

    private static ArrayList<Subscriber<Long>> subscribers = new ArrayList<Subsc
riber<Long>>();;

    private static long TICK_INTERVAL = 1000;

    private static Ticker instance;

    private static Observable<Long> tickerObservable;

    private static PublishSubject stop = PublishSubject.create();


    protected Ticker() { }

    /**
     * Singleton of the Ticker class, prevents having more than 1 ticker in the
system. Creates a new
     * Ticker if one does not yet exist, otherwise gives the instance
     * @return
     */
    public static Ticker getInstance() {
        if(instance == null) {
            instance = new Ticker();
        }
        return instance;
    }

    /**
     * Starts the ticker with the tick interval
     */
    public static void start() {
        tickerObservable = Observable.interval(TICK_INTERVAL, TimeUnit.MILLISECO
NDS);
    }

    /**
     * Adds a subscriber to the ticker. Any class that extends Subscriber can su
bscribe to the ticker. This
```

```
     * also adds the subscriber to an arraylist of subscribers just to keep trac
k of them all.
     * @param sub the new subscriber to the ticker
     */
    public static void subscribe(Subscriber<Long> sub) {
        if(tickerObservable!=null) {
            tickerObservable.takeUntil(stop).observeOn(JavaFxScheduler.getInstan
ce()).subscribe(sub);
            subscribers.add(sub);
        }
    }

    /**
     * Get the list of all subscribers of this ticker, which can include Vehicle
s and TrafficLights
     * @return ArrayList of all subscribers of the ticker
     */
    public static ArrayList<Subscriber<Long>> getSubscribers() {
        return subscribers;
    }

    /**
     * Get the length of the current tick interval
     * @return length of the current tick interval
     */
    public static long getTickInterval() { return TICK_INTERVAL; }

    /**
     * Change the length of a tick interval.
     * @param interval length of new interval
     */
    public static void setTickInterval(long interval) { TICK_INTERVAL = interval
; }


    /**
     * Ends the ticker. All subscribers must unsubscribe and a "stop" call is ex
plicitly called, just in case.
     */
    public static void end() {
        for(Subscriber s : subscribers)
            s.unsubscribe();
        stop.onNext(null);
    }

}
```

```
package londonsw.model.simulation.components;

/**
 * This interface is to have a common denominator between things that can be add
ed to a Map
 * Currently, there are two types of Components, Intersections and Roads
 */
public interface Component {
 // intersection or road
}
```

```
package londonsw.model.simulation.components;

import java.io.Serializable;

/**
 * The Coordinate class represents an (x, y) pair of integers that signify a loc
ation in the map
 * Coordinates are also used for doing calculations and translations for movemen
t of vehicles
 * A coordinate at the top-left of a map is (0,0)
 */
public class Coordinate implements Serializable {

    private static final long serialVersionUID = 2522457951482787 39L;
    private int x, y;

    /**
     *
     * @param x takes the value of the x region/axis
     * @param y takes the value of the y region
     */
    public Coordinate(int x, int y) {
        this.x = x;
        this.y = y;
    }

    /**
     * Sets the y coordinate of this instance
     * @param y int of the y coordinate
     */
    public void setY(int y) {
        this.y = y;
    }

    /**
     * Sets the x coordinate of this instance
     * @param x int of the x coordinate
     */
    public void setX(int x) {
        this.x = x;
    }

    /**
     * Gets the x-coordinate of this instance
     * @return int of the x coordinate
     */
    public int getX() {
        return x;
    }

    /**
     * Gets the y-coordinate of this instance
     * @return int of the y coordinate
     */
    public int getY() {
        return y;
    }

    /**
```

```
     * Checks if two coordinates are equal
     * @param obj coordinate to be compared with
     * @return true if the coordinates are the same, false otherwise
     */
    public boolean equals(Object obj) {
        Coordinate other = (Coordinate)obj;
        return (x == other.getX()) && (y == other.getY());
    }

    /**
     * Adds 2 coordinates and returns their sum
     * @param a first coordinate to add
     * @param b second coordinate to add
     * @return the sum of the two coordinates' x's and y's as a Coordinate insta
nce
     */
    public static Coordinate add(Coordinate a, Coordinate b) {
        return new Coordinate(a.getX()+b.getX(), a.getY()+b.getY());
    }

    /**
     * Subtracts 2 coordinates and returns their difference
     * @param a coordinate to subtract from
     * @param b coordinate to subtract
     * @return the difference of the two coordinates' x's and y's as a Coordinat
e instance
     */
    public static Coordinate substract(Coordinate a, Coordinate b) {
        return new Coordinate(a.getX()-b.getX(), a.getY()-b.getY());
    }

    /**
     * Formats the coordinate for console output
     * @return the coordinate formatted as follows: (x, y)
     */
    public String toString() {
        return "(" + x + "," + y + ")";
    }

    /**
     * Adds a step to the coordinate based on map direction. i.e if the map dire
ction is eastwards,
     * and a specified coordinate is (2,1), calling addStep method returns a new
 coordinate with dimensions (2,1)
     * @param mapDirection i.e directions north,south, east or west
     * @return sum, returns a new valid coordinate with a step added to it
     */
    public Coordinate addStep(MapDirection mapDirection) {
        Coordinate sum = new Coordinate(-1,-1);

        switch (mapDirection) {
            case NORTH:
                sum.setY(this.getY() - 1);
                sum.setX(this.getX());
                break;
            case SOUTH:
                sum.setY(this.getY() + 1);
                sum.setX(this.getX());
                break;
```

```
            case EAST:
                sum.setX(this.getX() + 1);
                sum.setY(this.getY());
                break;
            case WEST:
                sum.setX(this.getX() - 1);
                sum.setY(this.getY());
                break;
        }

        return sum;
    }

}
```

```java
package londonsw.model.simulation.components;

/**
 * Created by felix on 25/02/2016.
 */
public interface IMapGrid {

    Component[][] getGrid();

    int getWidth();

    int getHeight();

    boolean addComponent(Component component);

}
```

```
package londonsw.model.simulation.components;

import java.util.ArrayList;

/**
 * Created by felix on 23/02/2016.
 */
public interface IRoad {
    ArrayList<Lane> getLanes();
    void addLane(Lane lane);
    Lane getLaneAtIndex(int index);
    Coordinate getEndLocation();
    int getNumberLanes();
    Intersection getIntersection();
    void setIntersection(Intersection intersection);
    int getLength();
    boolean runsVertically();
}
```

```
package londonsw.model.simulation.components;

/**
 * Interface ITrafficLight
 */
public interface ITrafficLight {

    /**
     * Methods....
     */
    public void nextState();
    public void change(int no);
    public void setDuration(long duration);
    public long getDuration();
}
```

```
package londonsw.model.simulation.components;

import londonsw.controller.IntersectionController;
import londonsw.model.simulation.components.vehicles.Vehicle;
import rx.Subscriber;
import londonsw.model.simulation.Ticker;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Random;


/**
 * This class is our "node" in our directed graph
 * It will hold anywhere between 1 and 4 traffic lights
 * It will connect anywhere between 2 and 4 roads
 * Each will have a location in the map
 */

/* the traffic light belongs to the road
 * in each intersection, a car can choose(maybe randomly) Which road he can ente
r based on the array IntersectionRoad
 */


public class Intersection extends Subscriber<Long> implements Component, Serializa
ble {

    private static final long serialVersionUID = -2621352799268337492L;
    private Road northRoad;
    private Road southRoad;
    private Road eastRoad;
    private Road westRoad;

    private TrafficLight northTrafficLight;
    private TrafficLight southTrafficLight;
    private TrafficLight eastTrafficLight;
    private TrafficLight westTrafficLight;
    private ArrayList<TrafficLight> allLights;
    private int id;
    private Coordinate location;
    private static  int counter=0;


    /**
     * Creats
     * @param location
     */
    public Intersection(Coordinate location){
        this.northRoad = null;
        this.southRoad = null;
        this.eastRoad = null;
        this.westRoad = null;
        this.location = location;
        this.northTrafficLight = null;
        this.southTrafficLight = null;
        this.eastTrafficLight = null;
        this.westTrafficLight = null;
        Ticker.subscribe(this);
```

```
        id=++counter;
    }

    /* getters */
    public TrafficLight getNorthTrafficLight() {
        return northTrafficLight;
    }
    public TrafficLight getSouthTrafficLight() {
        return southTrafficLight;
    }
    public TrafficLight getEastTrafficLight() {
        return eastTrafficLight;
    }
    public TrafficLight getWestTrafficLight() {
        return westTrafficLight;
    }
    public Road getNorthRoad() {
        return northRoad;
    }
    public Road getEastRoad() {
        return eastRoad;
    }
    public Road getSouthRoad() {
        return southRoad;
    }
    public Road getWestRoad() {
        return westRoad;
    }
    public Coordinate getLocation() {
        return location;
    }
    public int getId() {
        return id;
    }

    /* setters */
    public void setNorthTrafficLight(TrafficLight northTrafficLight) {
        this.northTrafficLight = northTrafficLight;
    }
    public void setSouthTrafficLight(TrafficLight southTrafficLight) {
        this.southTrafficLight = southTrafficLight;
    }
    public void setEastTrafficLight(TrafficLight eastTrafficLight) {
        this.eastTrafficLight = eastTrafficLight;
    }
    public void setWestTrafficLight(TrafficLight westTrafficLight) {
        this.westTrafficLight = westTrafficLight;
    }
    public void setLocation(Coordinate location) throws IntersectionSetupExcepti
on {
        this.location = location;
    }
    public void setIdIntersection(int id) {
        this.id = id;
    }

    /**
     * Subscribes this intersection to the Ticker. This is expecially useful whe
```

```
n loading maps from file!
     */
    public void subscribeToTicker() {
        Ticker.subscribe(this);
    }


    public void setNorthRoad(Road northRoad) throws Exception {
        if((this.location.getX() == northRoad.getEndLocation().getX()
                && (this.location.getY() - 1 == northRoad.getEndLocation().getY(
)
                ||  this.location.getY() - 1 == northRoad.getStartLocation().get
Y())))
        {
            this.northRoad = northRoad;
            for(int i=0; i<this.northRoad.getNumberLanes();i++){
                if(this.northRoad.getLaneAtIndex(i).getMovingDirection()==MapDir
ection.SOUTH)
                {
                    this.northRoad.getLaneAtIndex(i).setEndIntersection(this);
                }
            }
        }
        else
            throw new IntersectionSetupException("Road end location coordinates must match w
ith Intersection");
    }

    public void setSouthRoad(Road southRoad) throws Exception {
        if((this.location.getX()==southRoad.getEndLocation().getX()
                && (this.location.getY() + 1 == southRoad.getEndLocation().getY(
)
                || this.location.getY() + 1 == southRoad.getStartLocation().getY
())))
        {
            this.southRoad = southRoad;
            for(int i=0; i<this.southRoad.getNumberLanes();i++){
                if(this.southRoad.getLaneAtIndex(i).getMovingDirection()==MapDir
ection.NORTH)
                {
                    this.southRoad.getLaneAtIndex(i).setEndIntersection(this);
                }
            }

        }
        else
            throw new IntersectionSetupException("Road end location coordinates must match w
ith Intersection");
    }

    public void setEastRoad(Road eastRoad) throws Exception {
        if (this.location.getY() == eastRoad.getEndLocation().getY()
                && (this.location.getX() + 1  == eastRoad.getEndLocation().getX(
)
                || this.location.getX() + 1 == eastRoad.getStartLocation().getX(
))) {
            this.eastRoad = eastRoad;
            for(int i=0; i<this.eastRoad.getNumberLanes();i++){
                if(this.eastRoad.getLaneAtIndex(i).getMovingDirection()==MapDire
```

```
ction.WEST)
                {
                    this.eastRoad.getLaneAtIndex(i).setEndIntersection(this);
                }
            }

        } else
            throw new IntersectionSetupException("Road end location coordinates must match w
ith Intersection");
    }

    public void setWestRoad(Road westRoad) throws IntersectionSetupException {
        if ((this.location.getY()  == westRoad.getEndLocation().getY()
                && (this.location.getX() - 1 == westRoad.getEndLocation().getX()
                || this.location.getX() -1 == westRoad.getStartLocation().getX()
))) {
            this.westRoad = westRoad;
            for(int i=0; i<this.westRoad.getNumberLanes();i++){
                if(this.westRoad.getLaneAtIndex(i).getMovingDirection()==MapDire
ction.EAST)
                {
                    this.westRoad.getLaneAtIndex(i).setEndIntersection(this);
                }
            }

        } else
            throw new IntersectionSetupException("Road end location coordinates must match w
ith Intersection");
    }

    public void setDefaultTrafficLightsForRoads() {
        if(northRoad != null) {
            ArrayList<Lane> lanes = northRoad.getLanes();
            boolean hasSouthLane = false;
            for(Lane l : lanes) {
                if(l.getMovingDirection() == MapDirection.SOUTH) {
                    hasSouthLane = true;
                    break;
                }
            }
            if(hasSouthLane)
                northTrafficLight = new TrafficLight();
        }

        if(southRoad != null) {
            ArrayList<Lane> lanes = southRoad.getLanes();
            boolean hasNorthLane = false;
            for(Lane l : lanes) {
                if(l.getMovingDirection() == MapDirection.NORTH) {
                    hasNorthLane = true;
                    break;
                }
            }
            if(hasNorthLane)
                southTrafficLight = new TrafficLight();
        }
        if(eastRoad != null) {
            ArrayList<Lane> lanes = eastRoad.getLanes();
            boolean hasWestLane = false;
```

```
            for(Lane l : lanes) {
                if(l.getMovingDirection() == MapDirection.WEST) {
                    hasWestLane = true;
                    break;
                }
            }
            if(hasWestLane) {
                if (northRoad != null || southRoad != null) {
                    eastTrafficLight = new TrafficLight(LightColour.GREEN);
                } else
                    eastTrafficLight = new TrafficLight();
            }
        }
        if(westRoad != null) {
            ArrayList<Lane> lanes = westRoad.getLanes();
            boolean hasEastLane = false;
            for(Lane l : lanes) {
                if(l.getMovingDirection() == MapDirection.EAST) {
                    hasEastLane = true;
                    break;
                }
            }
            if(hasEastLane) {
                if (northRoad != null || southRoad != null)
                    westTrafficLight = new TrafficLight(LightColour.GREEN);
                else
                    westTrafficLight = new TrafficLight();
            }
        }
    }


    /**
     * generate a list of 4 items form 1 to 4
     * these items are placed in the array randomly
     * @return list of integer in type of ArrayList
     */
    public static ArrayList<Integer> generateRandom (){
        int size =4;
        ArrayList<Integer> list = new ArrayList<Integer>(size);
        ArrayList<Integer> l = new ArrayList<Integer>(size);

        for(int i = 1; i <= size; i++) {
            list.add(i);
        }Random rand = new Random();
        while(list.size() > 0) {
            int index = rand.nextInt(list.size());
            l.add(list.get(index));
           // System.out.println("Selected: "+list.remove(index));
            list.remove(index);
        }

        return l;

    }


    /**
     * intersection gives each vehicle on it a priority to turn first
```

```
     * first it checks :
     * 1. if there is a road connected to it.
     * 2. if in the road there is a lane which its direction to this intersectio
n
     * 3. if there is a vehicle at the last cell in this lane, which is this int
ersection
     * if these conditions are obtained , then the intersection gives this vehic
le a priority to turn
     * and put these vehicle in arrayList .
     * @param randomPriority a list 4 items form 1 to 4, arranged randomly in th
e array list
     * @return vehicleInIntersection an arrayList in type of integer , which con
tains all vehicles which
     * has priority to turn
     * @throws
     */
    public ArrayList<Vehicle> giveVehiclePriorities (ArrayList<Integer>  randomP
riority) throws Exception {

       // ArrayList<Integer>  randomPriority = (ArrayList<Integer>)this.generate
Random(4).clone();
        ArrayList<Vehicle> vehicleInIntersection= new ArrayList<>() ;

        if (this.getNorthRoad() != null) {
            for (int i = 0; i < this.getNorthRoad().getNumberLanes(); i++) {
                if ((this.getNorthRoad().getLaneAtIndex(i).getMovingDirection()
== MapDirection.SOUTH)) {
                    if ((this.getNorthRoad().getLaneAtIndex(i).getVehicleInInter
section() != null)) {
                        if(this.getNorthTrafficLight()!= null)
                            this.getNorthRoad().getLaneAtIndex(i).getVehicleInIn
tersection().setVehicleTrafficLight(this.getNorthTrafficLight());
                        this.getNorthRoad().getLaneAtIndex(i).getVehicleInInters
ection().setVehiclePriorityToTurn(randomPriority.get(0));
                        vehicleInIntersection.add( this.getNorthRoad().getLaneAt
Index(i).getVehicleInIntersection());
                    }
                }
            }
        }

        if (this.getSouthRoad() != null) {
            for (int i = 0; i < this.getSouthRoad().getNumberLanes(); i++) {
                if ((this.getSouthRoad().getLaneAtIndex(i).getMovingDirection()
== MapDirection.NORTH)) {
                    if ((this.getSouthRoad().getLaneAtIndex(i).getVehicleInInter
section() != null)) {
                        if(this.getSouthTrafficLight()!= null)
                            this.getSouthRoad().getLaneAtIndex(i).getVehicleInIn
tersection().setVehicleTrafficLight(this.getSouthTrafficLight());
                        this.getSouthRoad().getLaneAtIndex(i).getVehicleInInters
ection().setVehiclePriorityToTurn(randomPriority.get(1));
                        vehicleInIntersection.add( this.getSouthRoad().getLaneAt
Index(i).getVehicleInIntersection());
                    }
                }
            }
        }
```

```
        if (this.getEastRoad() != null) {
            for (int i = 0; i < this.getEastRoad().getNumberLanes(); i++) {
                if ((this.getEastRoad().getLaneAtIndex(i).getMovingDirection() =
= MapDirection.WEST)) {
                    if ((this.getEastRoad().getLaneAtIndex(i).getVehicleInInters
ection() != null)) {
                        if(this.getEastTrafficLight()!= null)
                            this.getEastRoad().getLaneAtIndex(i).getVehicleInInt
ersection().setVehicleTrafficLight(this.getEastTrafficLight());
                        this.getEastRoad().getLaneAtIndex(i).getVehicleInInterse
ction().setVehiclePriorityToTurn(randomPriority.get(2));
                        vehicleInIntersection.add( this.getEastRoad().getLaneAtI
ndex(i).getVehicleInIntersection());
                    }
                }
            }
        }

        if (this.getWestRoad() != null) {
            for (int i = 0; i < this.getWestRoad().getNumberLanes(); i++) {
                if ((this.getWestRoad().getLaneAtIndex(i).getMovingDirection() =
= MapDirection.EAST)) {
                    if ((this.getWestRoad().getLaneAtIndex(i).getVehicleInInters
ection() != null)) {
                        if(this.getWestTrafficLight()!= null)
                            this.getWestRoad().getLaneAtIndex(i).getVehicleInInt
ersection().setVehicleTrafficLight(this.getWestTrafficLight());
                        this.getWestRoad().getLaneAtIndex(i).getVehicleInInterse
ction().setVehiclePriorityToTurn(randomPriority.get(3));
                        vehicleInIntersection.add( this.getWestRoad().getLaneAtI
ndex(i).getVehicleInIntersection());
                    }
                }
            }
        }
        return vehicleInIntersection;

    }

    /**
     * checks the vehicle with higher priority to turn
     * if a vehicle has a higher priority
     * it sets its priority to turn to 1 which means move
     * otherwise sets it to 0 which means stop
     * @param vehicles an array list in type of integer which contains all vehic
les which has priority to turn
     * @return
     * @throws Exception
     */
    public boolean vehicleTurnFirst (ArrayList<Vehicle> vehicles)throws Exceptio
n{
        int max=0;

       // for (int i=0; i<vehicles.size();i++)
       // {System.out.println("ID is: " + vehicles.get(i).getId()+ "  priority i
s : "+ vehicles.get(i).getVehiclePriorityToTurn());}


        if (vehicles != null) {
```

```
    for (int i = 0; i < vehicles.size(); i++) {
            if (max <= vehicles.get(i).getVehiclePriorityToTurn() ){
                if(vehicles.get(i).getVehicleTrafficLight()!= null){
                if(vehicles.get(i).getVehicleTrafficLight().getState()==LightCol
our.GREEN)
                max = vehicles.get(i).getVehiclePriorityToTurn();}
        else max = vehicles.get(i).getVehiclePriorityToTurn();}

    }


    for (int i = 0; i < vehicles.size(); i++) {
            if (vehicles.get(i).getVehiclePriorityToTurn() == max || vehicles.ge
t(i).getVehiclePriority()==5)
            {vehicles.get(i).setVehiclePriorityToTurn(1);}
        else vehicles.get(i).setVehiclePriorityToTurn(0);
    }
}

        return true;
    }


    /**
     * This is the method that gets called when the ticker terminates
     *  Left not implemented on purpose
     */
    @Override
    public void onCompleted() {

    }

    /**
     * If there's some error with the ticker and this subscriber, this method wo
uld call.
     * Left not implemented on purpose
     * @param throwable
     */
    @Override
    public void onError(Throwable throwable) {

    }

    /**
     * This is like the onTick method. This is what intersection would do when t
he ticker ticks.
     * @param aLong this gives the current time in the system to the intersectio
n (although it is probably not required)
     */
    @Override
    public void onNext(Long aLong){
        try {

            IntersectionController.vehicleTurnFirst(this, this.giveVehiclePriori
ties(this.generateRandom()));
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }



}

class IntersectionSetupException extends Exception {
    public IntersectionSetupException() {
        super();
    }
    public IntersectionSetupException(String msg) {
        super();
    }
    public IntersectionSetupException(String msg, Throwable t) {
        super(msg, t);
    }
    public IntersectionSetupException(Throwable t) {
        super(t);
    }
}
```

```
package londonsw.model.simulation.components;
import londonsw.model.simulation.components.vehicles.Vehicle;

import java.io.Serializable;

/**
 * This class is where the vehicles actually move
 * This is based on the cell automaton model of simulation
 * Each lane is an "queue" and has a direction
 * Number slots in the lane will be based on the number of "cells" in the view t
he road/lane takes up
 */
public class Lane implements Serializable {

    private static final long serialVersionUID = 7899381124564682583L;
    private Vehicle[] lane;
    private int length;
    private Coordinate entry;
    private Coordinate exit;
    private MapDirection movingDirection;
    private Road road;
    private Intersection endIntersection;
    private int RoadIndex;
    private int state;
    private int id;
    private static  int counter=0;


    /**
     * Creates a lane and sets its first and last cell and it’s moving direction
     * and calculate the length of a lane and gives it a unique Id
     * stes the state to 1 which means a lane is enabled.
     * @param entry first cell in a lane in type of Coordinate
     * @param exit last cell in a lane un type of Coordinate
     * @param movingDirection the moving direction  of a lane in type of Map dir
ection
     * @throws NotALaneException
     */
    public Lane(Coordinate entry, Coordinate exit, MapDirection movingDirection)
 throws NotALaneException {
        this.entry = entry;
        this.exit = exit;
        this.movingDirection = movingDirection;
        length = this.getLaneLength();
        lane = new Vehicle[length];
        id=++counter;
        this.setState(1);
    }


    /**
     * Gets the vehicle in last cell in a lane
     * @return the vehicle in an intersection if there is any in type of vehicle
     * if there is no, it returns null
     * @throws Exception
     */
    public Vehicle getVehicleInIntersection() throws Exception {
        if (lane[length-1] != null)
            return lane[length-1];
```

```
        else
            return null;
    }

    /**
     * Gets a lane's state
     * 1 if a lane is enabled
     * 0 if it is disabled
     *
     * @return the state of a lane in type of integer
     */
    public int getState() {
        return state;
    }

    /**
     * Gets a unique ID for the lane
     * @return a lane's ID in type of integer
     */
    public int getId() {
        return id;
    }


    /**
     * Sets an ID to the lane
     * @param id is a unique Id for a lane in type of integer
     */
    public void setId(int id) {
        this.id = id;
    }

    /**
     * Sets a state of a lane
     * @param state is the state of a lane in type of integer
     *              where 1 is enabled  and 0 is disabled
     */
    public void setState(int state) {
        this.state = state;
    }


    /**
     * Gets the length of a lane,
     * and checks if it is a legal lane : which means the entry and exit coordin
ate of a lane must have
     * either same x's or y's
     *
     * @return the lane length in type od integer
     * @throws NotALaneException if the lane coordinates are not legal.
     */
    private int getLaneLength() throws NotALaneException {
        int aX = entry.getX();
        int aY = entry.getY();
        int bX = exit.getX();
        int bY = exit.getY();
        int length;

        if (aX == bX) {
```

```
            length = Math.abs(aY − bY) + 1;
            return length;
        } else if (aY == bY) {
            length = Math.abs(aX − bX) + 1;
            return length;
        } else
            throw new NotALaneException("Not a lane. Coordinate x or y must match for both");
    }

    /**
     * Gets the length of a lane
     * @return the lane length in type of integer
     */
    public int getLength() {
        return length;
    }

    /**
     * Checks if a given cell is empty
     * @param cell is a cell in a lane in type of integer
     * @return the true if a cell is empty and false if not
     */
    public boolean isCellEmpty(int cell) {
        if (cell < 0 || cell > this.length)
            return false;

        if (lane[cell] == null)
            return true;
        return false;
    }

    /**
     * Checks if a lane is full
     * @return true if a lane is full, false otherwise
     */
    public boolean isFull ()
    {
        for (int i=0; i<this.getLength();i++)
        {if (this.isCellEmpty(i))
            return false;
        }
        return true;
    }

    /**
     * Gets the entry coordinate of a lane
     * @return the lane entry in type of coordinate
     */
    public Coordinate getEntry() {
        return entry;
    }

    /**
     * Gets the exit coordinate of a lane
     * @return the lane exit in type of coordinate
     */
    public Coordinate getExit() {
        return exit;
    }
```

```
    /**
     * Gets the moving direction of a lane
     * @return the lane moving direction in type of MapDirection
     */
    public MapDirection getMovingDirection() {
        return movingDirection;
    }

    /**
     * Gets the road that a lane is belongs to
     * @return the Road that lane in it in type of road
     */
    public Road getRoad() {return road; }

    /**
     * Sets the road that a alane belongs to
     * @param road to set it to a lane in type of road
     */
    public void setRoad(Road road) {this.road = road;}


    /**
     * Gets the end Intersection of a lane,
     * @return the lane End intersection in type of Intersection
     */
    public Intersection getEndIntersection() {
        return endIntersection;
    }

    /**
     * Sets the end intersection for a lane
     *
     * then sets the intersection to the
     * and checks if the intersection is in the right coordinates
     * and if it matches the correct lane direction
     * @param endIntersection is the intersection to set to the lane
     */
    public void setEndIntersection(Intersection endIntersection) {
        int x =endIntersection.getLocation().getX();
        int y =endIntersection.getLocation().getY();

        if ((this.getMovingDirection()==MapDirection.NORTH)&&(this.getExit().get
X()==x)&&(this.getExit().getY()==y+1) )
        {this.endIntersection = endIntersection;}

        else if ((this.getMovingDirection()==MapDirection.SOUTH)&&(this.getExit(
).getX()==x)&&(this.getExit().getY()==y-1) )
        {this.endIntersection = endIntersection;}

        else if((this.getMovingDirection()==MapDirection.EAST)&&(this.getExit().
getX()==x-1)&&(this.getExit().getY()==y) )
        {this.endIntersection = endIntersection;}

        else if((this.getMovingDirection()==MapDirection.WEST)&&(this.getExit().
getX()==x+1)&&(this.getExit().getY()==y) )
        {this.endIntersection = endIntersection;}

    }
```

```
    /**
     * @param v is a vehicle to set it to a lane
     * @param cell is a cell in a lane to set vehicle on it
     * @return true if a vehicle is setted
     * false if the cell is out of bound
     */
    public boolean setCell(Vehicle v, int cell) {
        if (cell < 0 || cell >= length)
            return false;

        lane[cell] = v;
        return true;
    }




    public int getRoadIndex() {
        return RoadIndex;
    }
    public Vehicle get(int i) {
        return lane[i];
    }
    public void setRoadIndex(int roadIndex) {
        RoadIndex = roadIndex;
    }

    public static boolean Rotate(Lane lane1, Lane lane2){
        if (lane1.getMovingDirection()==lane2.getMovingDirection()){
            return false;
        }
        return true;
    }
}

class NotALaneException extends Exception {
    public NotALaneException() { super(); }
    public NotALaneException(String msg) { super(msg); }
    public NotALaneException(String msg, Throwable t) { super(msg,t); }
    public NotALaneException(Throwable t) { super(t); }
}
```

```
package londonsw.model.simulation.components;

/**
 * This enum represents the types of colors that a TrafficLight could have. The
system only uses RED and GREEN at the moment,
 * but in case YELLOW was to be added in the future, it is included here as well
.
 */
public enum LightColour {
    GREEN, YELLOW, RED
}
```

```
package londonsw.model.simulation.components;

/**
 * This enum is to represent different directions in the map, namely the 4 cardi
nal directions NORTH, SOUTH, EAST, AND WEST.
 */
public enum MapDirection {
    NORTH, SOUTH, EAST, WEST, ERROR
}
```

```
package londonsw.model.simulation.components;

import java.math.BigDecimal;

/**
 * This class represents how images will be resized to fit on the screen. Each i
mage in our system is 100x100, but they
 * need to be scaled down in order to be displayed properly. This is used throug
hout the system, for drawing the grid
 * squares, vehicles, and determining the location to where vehicles will move.
 */
public class ResizeFactor {

    private double resizeX;
    private double resizeY;

    /**
     * Creates a new ResizeFactor instance. There are two parameters, although o
nly one is used in most situations. This
     * is so that each square image maintains its aspect ratio. The typical rang
e for resize factors is from 0 < ResizeFactor < 1.
     * @param resizeX how much to resize the x direction by
     * @param resizeY how much to resize the y direction by
     */
    public ResizeFactor(double resizeX, double resizeY) {
        this.resizeX = resizeX;
        this.resizeY = resizeY;
    }

    /**
     * Gets the resize factor of the x-direction
     * @return resize factor of x
     */
    public double getResizeX() {
        return resizeX;
    }

    /**
     * Gets the resize factor of the y-direction
     * @return resize factor of y
     */
    public double getResizeY() {
        return resizeY;
    }

    /**
     * Dynamically determines the resize factor for a map. Given a map width and
 height, determine the best
     * resize factor so that the map best displays on the user's display. It use
s the larger of the two dimensions
     * to determine the best fitting resize factor.
     * @param mapWidth width of the map to get a resize factor for
     * @param mapHeight height of the map to get a resize factor for
     * @return a ResizeFactor that is good for the given map dimensions
     */
    public static ResizeFactor getSuggestedResizeFactor(int mapWidth, int mapHei
ght) {
        int larger = mapHeight > mapWidth ? mapHeight : mapWidth;
```

```
        double rf = 1.0;
        if(larger < 10 || larger < 10) {
            rf = 1.0 / (mapHeight * 0.4);
        }
        else {
            rf = 1.0 / (larger * 0.2);
        }

        BigDecimal resizeFactor = new BigDecimal(rf);
        BigDecimal rounded = resizeFactor.setScale(2,BigDecimal.ROUND_HALF_UP);
        double result = rounded.doubleValue();

        return new ResizeFactor(result, result);
    }

    public String toString() {
        return resizeX+","+resizeY;
    }
}
```

```
package londonsw.model.simulation.components;

import java.io.Serializable;
import java.util.ArrayList;

/**
 * Each road is connected to at most 2 intersections (one at each end)
 * Each road is composed of a number of lanes (currently 2, one for each directi
on)
 * These are like the edges in our directed graph
 * Each has a start-location and an end-location
 */
public class Road implements Component, Serializable, IRoad {

    private static final long serialVersionUID = 6679898165504556586L;
    private Coordinate start;
    private Coordinate end;
    private ArrayList<Lane> lanes;
    private Intersection intersection;
    private static int counter = 0;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    private int id;

    /**
     * Creates an instance of a new Road. It has no lanes yet. (Note: a road wit
hout lanes
     * should not exist, so the user must specify lanes right away). A road can
have anywhere
     * between 1 and n lanes.
     * A road is defined by where it starts in the grid and where it ends in the
 grid.
     * The coordinates can be in any order, as long as they form a straight line
.
     *
     * @param start the location of one end of the road
     * @param end   the location of the other end of the road
     */
    public Road(Coordinate start, Coordinate end) {
        lanes = new ArrayList<Lane>();
        this.start = start;
        this.end = end;
        this.id = ++counter;
    }

    /**
     * Gets the number of lanes a road has
     *
     * @return number of lanes in the road
     */
    public ArrayList<Lane> getLanes() {
        return lanes;
```

```
    }

    /**
     * Adds a lane to the road
     *
     * @param l the Lane to add to the road
     */
    public void addLane(Lane l) {
        l.setRoad(this);
        l.setRoadIndex(lanes.size());
        lanes.add(l);
    }

    /**
     * Gets the lane at the index specified
     *
     * @param i index of lane
     * @return the instance of Lane at that index i
     */
    public Lane getLaneAtIndex(int i) {
        return lanes.get(i);
    }

    /**
     * Gets the beginning coordinate of the road
     *
     * @return location of beginning of road of type Coordinate
     */
    public Coordinate getStartLocation() {
        return start;
    }

    /**
     * Gets the end coordinate of the road
     *
     * @return location of end of road of type Coordinate
     */
    public Coordinate getEndLocation() {
        return end;
    }

    /**
     * Gets the number of lanes currently part of the road
     *
     * @return number of lanes in the road of type int
     */
    public int getNumberLanes() {
        return lanes.size();
    }

    public Intersection getIntersection() {
        //TODO
        return intersection;
    }

    public void setIntersection(Intersection intersection) {
        //TODO
        this.intersection = intersection;
    }
```

```
    public void setStart(Coordinate start) {
        this.start = start;
    }

    public void setEnd(Coordinate end) {
        this.end = end;
    }

    /**
     * Uses the coordinates to determine how long the road is
     * A road has a minimum length of 1
     *
     * @return length of the road
     */
    public int getLength() {
        int aX = start.getX();
        int aY = start.getY();
        int bX = end.getX();
        int bY = end.getY();
        int length;

        if (aX == bX) {
            length = Math.abs(aY - bY) + 1;
            return length;
        } else if (aY == bY) {
            length = Math.abs(aX - bX) + 1;
            return length;
        } else
            return -1;
    }

    /**
     * Determines if a road runs NORTH to SOUTH
     *
     * @return true if the road runs NORTH to SOUTH or SOUTH to NORTH, false if
the road runs EAST to WEST or WEST to EAST
     */
    public boolean runsVertically() {
        int aX = start.getX();
        int bX = end.getX();
        int aY = start.getY();
        int bY = end.getY();

        // if aX==bX, then road runs vertically
        if (aX == bX) {
            if(aY == bY) {
                // the road is length of 1, do something else
                if(lanes.size() == 0) {
                    return true;
                } else {
                    Lane lane = lanes.get(0);
                    if(lane.getMovingDirection() == MapDirection.NORTH || lane.g
etMovingDirection() == MapDirection.SOUTH) {
                        return true;
                    } else {
                        return false;
                    }
                }
```

```
            }
            return true;
        }

        // if aY==bY, then road runs horizontally
        return false;
    }

    public boolean runsVertically(MapDirection mapDirection) {

        if (mapDirection.equals(mapDirection.NORTH) || mapDirection.equals(mapDi
rection.SOUTH))
            return true;
        else
            return false;
    }
}
```

```
package londonsw.model.simulation.components;

import londonsw.controller.TrafficLightController;
import londonsw.model.simulation.Ticker;
import rx.Subscriber;

import java.io.Serializable;

/**
 * The implementation of the traffic lights in our system. This extends the RxJa
va class Subscriber to receive
 * messages from the Ticker.
 */
public class TrafficLight extends Subscriber<Long> implements Serializable {

    private static final long serialVersionUID = 12997479486649264 47L;
    private LightColour state;
    private long duration;
    private long currentTime;
    private static int counter = 0;
    private int id;

    /**
     * Default constructor, initial light color is red
     */
    public TrafficLight() {
        this.currentTime = 1000;
        Ticker.subscribe(this);
        this.state = LightColour.RED;
        this.id = ++counter;
        this.duration = TrafficLightController.getInstance().getDurationLength()
;
    }

    /**
     * Creating a new traffic light with a new default color
     * @param colour the initial color of the traffic light
     */
    public TrafficLight(LightColour colour) {
        this.currentTime = 1000;
        Ticker.subscribe(this);
        this.state = colour;
        this.id = ++counter;
        this.duration = TrafficLightController.getInstance().getDurationLength()
;
    }

    /**
     * Subscribes this traffic light to the Ticker. Especially useful if using a
 loaded map!
     */
    public void subscribeToTicker() {
        Ticker.subscribe(this);
    }

    /**
     * Get the current color of the traffic light
     * @return LightColour (enum) of the current colour
     */
```

```
    public LightColour getState() {
        return state;
    }

    /**
     * Gets the ID of the traffic light, useful for logging
     * @return the id of this traffic light instance
     */
    public int getId() {
        return id;
    }

    /**
     * Sets the id of the traffic light
     * @param id the desired id of the traffic light
     */
    public void setId(int id) {
        this.id = id;
    }

    /**
     * Set the color of the traffic light from an external source
     * @param state LightColour (enum) of the color to be
     */
    public void setState(LightColour state) {
        this.state = state;
    }

    /**
     * Set the color of the traffic light based on the current state.
     * Current behaviour:
     * If currently red, go to green
     * If currently green, go to red
     */
    public void nextState() {
        switch (state) {
            case RED:
                state = LightColour.GREEN;
                break;
            case YELLOW:
                state = LightColour.RED;
                break;
            case GREEN:
                state = LightColour.RED; // changed to RED, no yellow behaviour
for now
                break;
            default:
                state = LightColour.RED;
                break;
        }

    }

    /**
     * Set how long the traffic light should be a specific color
     * @param duration time (in millis) of how long the traffic light should sta
y its color
     */
    public void setDuration(long duration) {
```

```
        this.duration = duration;

    }

    /**
     * Get the current duration of the traffic light (how long a color lasts)
     * @return the duration (in millis)
     */
    public long getDuration() {
        return duration;
    }


    /**
     * This is for what the traffic light would do if the ticker stops. Left uni
mplemented on purpose
     */
    @Override
    public void onCompleted() {    }

    /**
     * This is what the traffic light would do if there is an error thrown by th
e ticker’s observable. Left
     * unimplemented on purpose
     * @param throwable
     */
    @Override
    public void onError(Throwable throwable) {   }

    /**
     * This is like the onTick method. This is what happens when the ticker tick
s. The state changes after
     * a specified amount of ticks (the duration).
     * @param aLong the current time in the system
     */
    @Override
    public void onNext(Long aLong) {
        if(currentTime < (duration)) {
            currentTime += 1000;
        }
        else {
            currentTime = 1000;
            nextState();
            TrafficLightController.getInstance().colourChanged(state, this);
        }
    }
}
```

```
package londonsw.model.simulation.components;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.Random;

/**
 * This enum is to represent different types of behaviours vehicles can have. Fo
r instance, a Car might have average
 * behaviour, but an Ambulance has aggressive behaviour.
 */
public enum VehicleBehavior{
    AVERAGE, CAUTIOUS, AGGRESSIVE;

    private static final java.util.List<VehicleBehavior> VALUES= Collections.unm
odifiableList(Arrays.asList(values()));
    private static final int size = VALUES.size();
    private static  final Random RANDOM= new Random();
    public static VehicleBehavior randomLetter(){
        return VALUES.get(RANDOM.nextInt(size));
    }
}
```

```
package londonsw.model.simulation.components.vehicles;

import londonsw.model.simulation.components.Lane;


public class Ambulance extends Vehicle {


    /**
     * Create a vehicle and set its position by specify a cell in a lane
     *
     * @param currentCell the cell to set vehicle in, in the lane
     * @param currentLane the lane to set vehicle in
     */
    //TODO: Ambulance ignoring traffic light and blinking lights
    // TODO: figure out why ambulance gets stuck after moving forward
    public Ambulance(int currentCell, Lane currentLane) {
        super(currentCell, currentLane);
        this.vehicleLength=1;
        this.vehicleSpeed=1.0;
        this.vehiclePriority = 5;
        this.vehicleBehavior = vehicleBehavior.AGGRESSIVE;

    }
}
```

```
package londonsw.model.simulation.components.vehicles;
import londonsw.model.simulation.components.Lane;
import londonsw.model.simulation.components.VehicleBehavior;

import java.io.Serializable;

/**
 * An implementation of a vehicle
 * This moves in a lane
 * Only moves forwards when the slot in front of it is empty
 */


public class Car extends Vehicle implements Serializable, ICar {

    private static final long serialVersionUID = -3555254273903868035L;
    private static int idCounter = 0;
    private int id;

    public Car(int currentCell,Lane currentLane) {
        super(currentCell,currentLane);
        this.vehicleLength=1;
        this.vehicleSpeed=1.0;
        this.vehiclePriority = 1;
        this.vehicleBehavior = VehicleBehavior.AVERAGE; // default

        id = idCounter++;
    }

    public int getCarId() { return id; }

  }
```

```
package londonsw.model.simulation.components.vehicles;

/**
 * Created by felix on 26/02/2016.
 */
public interface ICar {

    int getCarId();

}
```

```
package londonsw.model.simulation.components.vehicles;

import londonsw.model.simulation.components.Coordinate;
import londonsw.model.simulation.components.Lane;
import londonsw.model.simulation.components.TrafficLight;
import londonsw.model.simulation.components.VehicleBehavior;

import java.util.ArrayList;

/**
 * Created by felix on 04/03/2016.
 */
public interface IVehicle {

    Lane getPreviousLane();
    void setPreviousLane(Lane previousLane);
    void setCurrentCoordinate(Coordinate currentCoordinate);
    int getVehicleLength();
    double getVehicleSpeed();
    int getVehiclePriority();
    Lane getCurrentLane();
    int getCurrentCell();
    int getVehicleState();
    VehicleBehavior getVehicleBehavior();
    Coordinate getCurrentCoordinate();
    void setVehicleLength(int vehicleLength);
    void setVehicleSpeed(double vehicleSpeed);
    void setVehiclePriority(int vehiclePriority);
    void setCurrentLane(Lane currentLane) throws Exception;
    void setCurrentCell(int curCell,Lane currentLane) throws Exception;
    void setVehicleState(int vehicleState);
    void setVehicleBehavior(VehicleBehavior vehicleBehavior);
    Lane chooseLane () throws Exception;
    int moveVehicle(int step) throws Exception;//changed return type to int
    void readTrafficLight() throws Exception;
    ArrayList<Lane> getLaneOptions() throws Exception;
    int vehicleTurn (Lane l) throws Exception; //changed return type to int
    Coordinate getStoredCurrentCoordinate();
    void setPreviousCoordinate(Coordinate coord);

     int getVehiclePriorityToTurn();
     void setVehiclePriorityToTurn(int vehiclePriorityToTurn);
     TrafficLight getVehicleTrafficLight();
     void setVehicleTrafficLight(TrafficLight vehicleTrafficLight);
}
```

```
package londonsw.model.simulation.components.vehicles;
import londonsw.controller.TrafficLightController;
import londonsw.controller.VehicleController;
import londonsw.model.simulation.Ticker;
import londonsw.model.simulation.components.*;
import rx.Subscriber;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Random;


/**
 * This is the abstract class that all vehicles will implement
 * This allows for scalability because we can add more types of cars (eg. ambula
nce, bus)
 *
 * This uses RxJava's Subscriber class to subscribe to the Ticker (that has as O
bservable). On each tick,
 * the onNext(..) method runs.
 */
public abstract class Vehicle extends Subscriber<Long> implements Serializable {

    private static final long serialVersionUID = -4552832373570448039L;
    protected int vehicleLength;
    protected double vehicleSpeed;
    protected int currentCell;
    protected int vehiclePriority;// 1 is the lowest
    protected int vehicleState;
    protected VehicleBehavior vehicleBehavior;
    protected  Lane currentLane;
    protected  ArrayList<Lane> laneOptions = new ArrayList<Lane>();
    protected  Random randomDirection;
    protected Lane l;
    protected Coordinate currentCoordinate;
    protected  Coordinate previousCoordinate;
    protected  Lane previousLane;
    protected int vehiclePriorityToTurn;
    protected TrafficLight vehicleTrafficLight;
    protected int timesTicked;
    private static int counter = 0;
    protected int id;
    protected int timeSpentStanding;


    /**
     * Create a vehicle and set its position by specify a cell in a lane
     *
     * @param currentCell the cell to set vehicle in, in the lane
     * @param currentLane the lane to set vehicle in
     */
    public Vehicle(int currentCell, Lane currentLane) {
        this.currentCell = currentCell;
        this.currentLane = currentLane;
        this.currentLane.setCell(this, currentCell);
        Ticker.subscribe(this);
        timesTicked = 0;
        id = ++counter;
        timeSpentStanding = 0;
```

```
    }

    /**
     * gets the traffic light that vehicle must read
     * @return traffic light in front of a vehicle in type of traffic light
     */
    public TrafficLight getVehicleTrafficLight() {
        return vehicleTrafficLight;
    }

    /**
     * sets the traffic light for vehicle
     * @param vehicleTrafficLight traffic light in front of vehicle
     */
    public void setVehicleTrafficLight(TrafficLight vehicleTrafficLight) {
        this.vehicleTrafficLight = vehicleTrafficLight;
    }


    /**
     * sets the vehicle priority to turn
     * @param vehiclePriorityToTurn the Priority Of vehicle To turn first int ty
pe of integer
     * in each intersection if there are more than vehicle, vehicles are given p
riorities to decide which turn first
     * so they do not crash
     */
    public void setVehiclePriorityToTurn(int vehiclePriorityToTurn) {
        this.vehiclePriorityToTurn = vehiclePriorityToTurn;
    }


    /**
     * gets vehicle priority to turn ,
     * @return the Priority Of vehicle To which turn first ib type of integer
     *depends on its priority its turns or stops
     */
    public int getVehiclePriorityToTurn() {
        return vehiclePriorityToTurn;
    }

    /**
     * gets a unique vehicle ID
     * @return  ID of vehicle in type of integer
     */
    public int getId() {
        return id;
    }

    /**
     * sets a unique ID for vehicle
     * @param id  unique ID for vehicle in type of integer
     */
    public void setId(int id) {
        this.id = id;
    }

    /**
     * Gets the length of a vehicle
```

```
     *
     * @return the length of a vehicle in type of integer
     * each type of vehicle has its own length
     */
    public int getVehicleLength() {
        return vehicleLength;
    }

    /**
     * Gets the speed of a vehicle
     *
     * @return the speed of a vehicle in type of double
     * each vehicle's speed depends on its behavior
     */
    public double getVehicleSpeed() {
        return vehicleSpeed;
    }


    public int getVehiclePriority() {
        return vehiclePriority;
    }

    /**
     * Gets the lane which is vehicle in
     *
     * @return the lane which is vehicle in on the current time in type of lane
     */
    public Lane getCurrentLane() {
        return currentLane;
    }

    /**
     * Gets the current cell in lane which is vehicle in
     *
     * @return cell from lane which is vehicle in on the current time , in type
of integer
     */
    public int getCurrentCell() {
        return currentCell;
    }

    /**
     * Gets the state of vehicle which are 0 refers to still or 1 refers to movi
ng
     *
     * @return the state of vehicle in type if integer
     */
    public int getVehicleState() {
        return vehicleState;
    }

    /**
     * Gets the behavior of a vehicle from three behaviors AVERAGE, AGGRESSIVE,
and CAUTIOUS
     *
     * @return the behavior of a vehicle in type of enum VehicleBehavior
     */
    public VehicleBehavior getVehicleBehavior() {
```

```
        return this.vehicleBehavior;
    }

    /**
     * Gets the previous lane which vehicle was in
     * @return the previous lane in type of Lane
     */
    public Lane getPreviousLane() {
        return previousLane;
    }

    public void setPreviousCoordinate(Coordinate prev) {
        this.previousCoordinate = prev;
    }

    public void setPreviousLane(Lane previousLane) {
        this.previousLane = previousLane;
    }

    public Coordinate getPreviousCoordinate() {
        return previousCoordinate;
    }

    /**
     * Increments the time spent standing by this vehicle. This exists because t
he VehicleController needs
     * to call this for all stationary vehicles.
     */
    public void incrementTimeSpentStanding() {
        timeSpentStanding++;
    }

    /**
     * Gets the time spent standing (not moving) in the system
     * @return the total time spent standing by this vehicle in the system
     */
    public int getTimeSpentStanding() {
        return timeSpentStanding;
    }

    /**
     * Gets the times ticked by this vehicle in the system. This is used in the
calculation for the average vehicle time
     * standing in the system.
     * @return the number of ticks this vehicle heard
     */
    public int getTimesTicked() {
        return timesTicked;
    }


    /**
     * Gets the current coordinate of this vehicle in the Map
     * @return the current coordinate of this vehicle in the Map
     */
    public Coordinate getCurrentCoordinate() {
        int currentCell = this.getCurrentCell();
        Lane currentLane = this.getCurrentLane();
        Coordinate coordinate = new Coordinate(0, 0);
```

```
        MapDirection mapDirection = currentLane.getMovingDirection();


        switch (mapDirection) {
            case NORTH:
                coordinate.setX(currentLane.getEntry().getX());
                coordinate.setY(currentLane.getEntry().getY() - currentCell);
                break;

            case SOUTH:
                coordinate.setX(currentLane.getEntry().getX());
                coordinate.setY(currentLane.getEntry().getY() + currentCell);
                break;

            case EAST:
                coordinate.setX(currentLane.getEntry().getX() + currentCell);
                coordinate.setY(currentLane.getEntry().getY());
                break;

            case WEST:
                coordinate.setX(currentLane.getEntry().getX() - currentCell);
                coordinate.setY(currentLane.getEntry().getY());
                break;
        }

        this.currentCoordinate = coordinate;

        return coordinate;
    }

    public Coordinate getStoredCurrentCoordinate() {
        return currentCoordinate;
    }
    //Setter

    /**
     * @param vehicleLength the length of a vehicle in type of integer
     */
    public void setVehicleLength(int vehicleLength) {
        this.vehicleLength = vehicleLength;
    }

    /**
     * @param vehicleSpeed the speed of vehicle in type of double
     */
    public void setVehicleSpeed(double vehicleSpeed) {
        this.vehicleSpeed = vehicleSpeed;
    }

    public void setVehiclePriority(int vehiclePriority) {
        this.vehiclePriority = vehiclePriority;
    }


    /**
     * @param currentLane a lane to set car in, in type of lane
     * @throws Exception if lane is not exist
     *                   set a vehicle into a new lane
```

```
     */
    public void setCurrentLane(Lane currentLane) throws Exception {
        if (currentLane != null) {
            this.currentLane = currentLane;
        } else
            throw new Exception("Lane is not Exist !");
    }

    /**
     * @param curCell     new cell to assign vehicle to
     * @param currentLane lane that cell takes place in
     * @throws Exception if the cell is out of bounds or is not empty
     *                   <p>
     *                   set a vehicle into a new cell
     *                   check if a new cell is empty, and not out of bounds
     *                   not less than zero or equal or more than lane length
     */
    public void setCurrentCell(int curCell, Lane currentLane) throws Exception {
        if ((curCell >= 0) && (curCell < currentLane.getLength()) && (currentLan
e.isCellEmpty(curCell))) {
            this.currentCell = curCell;
        } else if (this.getVehiclePriority() != 5) {
            throw new Exception("new cell is not available !");
        }
    }

    /**
     * @param vehicleState in type of integer
     *                     set the state of vehicle
     *                     0 for still and 1 for movement
     */
    public void setVehicleState(int vehicleState) {
        this.vehicleState = vehicleState;
    }

    /**
     * @param vehicleBehavior in type of VehicleBehavior enum
     *                        sets the behavior of a vehicle
     *                        there are three behaviors AVERAGE, AGGRESSIVE, and
 CAUTIOUS
     */
    public void setVehicleBehavior(VehicleBehavior vehicleBehavior) {
        this.vehicleBehavior = vehicleBehavior;
    }

    public void setCurrentCoordinate(Coordinate currentCoordinate) {
        this.currentCoordinate = currentCoordinate;
    }

    /**
     * @param step number of steps to move, depends on the behavior
     * @return returns the number of steps the vehicle is able to achieve
     * @throws Exception vehicle can move one step or more, depends on its behav
ior
     *                   if it moves two steps, should check if this movement is
 available
     *                   if not it checks if one step is available
     *                   if not vehicle stops
     */
```

```
    public int moveVehicle(int step) throws Exception {

        int curCell = this.getCurrentCell();

        while (step > 0) {
            if (curCell + step >= this.currentLane.getLength() || !this.currentL
ane.isCellEmpty(curCell + step)) {
                step--;
            } else
                break;
        }

            currentLane.setCell(null, curCell);
            curCell += step;
            this.setCurrentCell(curCell, this.getCurrentLane());
            currentLane.setCell(this, curCell);

            return step;
        }



    /**
     * to make a vehicle reads a traffic light
     * use intersection to read the traffic light, each intersection has up to f
our traffic light
     * depends on the direction of a lane which vehicle is in, a vehicle can rea
ds the corresponding traffic light
     * if the traffic light is green the vehicle state set to moving
     * if the traffic light is red the vehicle state set to still
     * @throws Exception
     */
    public void readTrafficLight()throws Exception {
        if (this.getCurrentCell() == this.currentLane.getLength() - 1) {
            TrafficLight light;

            // traffic lights are disabled, allow cars to move through lights
            if(!TrafficLightController.getInstance().areLightsEnabled()) {
                this.setVehicleState(1);
                return;
            }

            if (this.getCurrentLane().getEndIntersection() != null) {
                switch (this.getCurrentLane().getMovingDirection()) {
                    case NORTH:
                        light = this.getCurrentLane().getEndIntersection().getSo
uthTrafficLight();
                        break;
                    case SOUTH:
                        light = this.getCurrentLane().getEndIntersection().getNo
rthTrafficLight();
                        break;
                    case EAST:
                        light = this.getCurrentLane().getEndIntersection().getWe
stTrafficLight();
                        break;
                    case WEST:
                        light = this.getCurrentLane().getEndIntersection().getEa
stTrafficLight();
```

```
                    break;
                default: // ERROR case
                    light = null;
                    throw new Exception("Error Direction!");
                    // break;
            }

            if (this.getVehiclePriority() > 1 && light != null) {
                if (light.getState() == LightColour.RED) {
                    this.vehicleState = 1;

                }
            } else if (light != null) {
                if (light.getState() == LightColour.RED)
                    this.vehicleState = 0;
                else
                    this.vehicleState = 1;
            }
            else
            {
                //move because there isn't any traffic light
                this.setVehicleState(1);

            }
        } else {
            int curCell = this.getCurrentCell();
            currentLane.setCell(null, curCell);
            this.setVehicleState(3);
            this.unsubscribe();
            VehicleController.removeFromListAndMap(this);
        }


    } else
        throw new Exception("Reading traffic light when not at end of lane");
}

/**
 * in each intersection this method gives vehicles options of available lane
s that vehicles can move to
 * this method checks three conditions
 * 1. if the lane is exist
 * 2. if there is a space for a new vehicle
 * 3. if the direction of a lane is legal for the vehicle
 * if  a lane obtains these conditions, then it is added to laneOptions Arra
y List
 * @return  the options of lanes that vehicle can move to in type of Array L
ist of lanes
 * @throws Exception
 */
public ArrayList<Lane> getLaneOptions() throws Exception {
    laneOptions.clear();

    if(this.currentLane.getEndIntersection()!=null) {
        if ((this.currentLane.getEndIntersection().getEastRoad() != null) &&
                (this.currentLane.getMovingDirection() != MapDirection.WEST)
) {
            for (int i = 0; i < this.currentLane.getEndIntersection().getEas
tRoad().getNumberLanes(); i++) {
```

```
                    if ((this.currentLane.getEndIntersection().getEastRoad().get
LaneAtIndex(i).getMovingDirection() == MapDirection.EAST)
                            && (this.currentLane.getEndIntersection().getEastRoa
d().getLaneAtIndex(i).getState() == 1)) {
                        laneOptions.add(this.currentLane.getEndIntersection().ge
tEastRoad().getLaneAtIndex(i));
                    }
                }
            }

            if ((this.currentLane.getEndIntersection().getSouthRoad() != null) &
&
                    (this.currentLane.getMovingDirection() != MapDirection.NORTH
)) {
                for (int i = 0; i < this.currentLane.getEndIntersection().getSou
thRoad().getNumberLanes(); i++) {
                    if ((this.currentLane.getEndIntersection().getSouthRoad().ge
tLaneAtIndex(i).getMovingDirection() == MapDirection.SOUTH)
                            && (this.currentLane.getEndIntersection().getSouthRo
ad().getLaneAtIndex(i).getState() == 1)) {
                        laneOptions.add(this.currentLane.getEndIntersection().ge
tSouthRoad().getLaneAtIndex(i));
                    }
                }
            }

            if ((this.currentLane.getEndIntersection().getNorthRoad() != null) &
&
                    (this.currentLane.getMovingDirection() != MapDirection.SOUTH
)) {
                for (int i = 0; i < this.currentLane.getEndIntersection().getNor
thRoad().getNumberLanes(); i++) {
                    if ((this.currentLane.getEndIntersection().getNorthRoad().ge
tLaneAtIndex(i).getMovingDirection() == MapDirection.NORTH)
                            && (this.currentLane.getEndIntersection().getNorthRo
ad().getLaneAtIndex(i).getState() == 1)) {
                        laneOptions.add(this.currentLane.getEndIntersection().ge
tNorthRoad().getLaneAtIndex(i));
                    }
                }
            }

            if ((this.currentLane.getEndIntersection().getWestRoad() != null) &&
                    (this.currentLane.getMovingDirection() != MapDirection.EAST)
) {
                for (int i = 0; i < this.currentLane.getEndIntersection().getWes
tRoad().getNumberLanes(); i++) {
                    if ((this.currentLane.getEndIntersection().getWestRoad().get
LaneAtIndex(i).getMovingDirection() == MapDirection.WEST)
                            && (this.currentLane.getEndIntersection().getWestRoa
d().getLaneAtIndex(i).getState() == 1)) {
                        laneOptions.add(this.currentLane.getEndIntersection().ge
tWestRoad().getLaneAtIndex(i));
                    }
                }
            }

            return laneOptions;
        }
```

```
        return null;
    }




    /**
     * Chooses lane randomly from lanes options
     * @return a random lane in type of Lane
     * @throws Exception
     */
    public Lane chooseLane() throws Exception {
        int num = 0;


        if(this.getLaneOptions()!=null) {
            num = this.getLaneOptions().size();
        }

        if (num > 0) {
            randomDirection = new Random();
            int size = randomDirection.nextInt(this.getLaneOptions().size());
            l= this.getLaneOptions().get(size);
            return l;
        }

        return null;
    }


    /**
     * there are four condition for vehicle to turn:
     * 1. if the lane is exist and
     * 2. the vehicle is at the last cell of the lane
     * 3. the first cell is empty
     * 4. the vehicle priority to turn is 1
     * if these conditions are obtained vehicle turn
     * otherwise vehicle stops
     * @param l a random lane to move to in type of Lane
     * @return integer representation of booleans
     * @throws Exception
     */
    public int vehicleTurn(Lane l) throws Exception {
        Lane oldLane = this.currentLane;


            //validate if its end of lane
            if (((l != null) && (this.getCurrentCell() == this.currentLane.getLe
ngth() -1) && (l.isCellEmpty(0)) && this.getVehiclePriorityToTurn()==1) ||
                    (l != null && this.getCurrentCell() == this.currentLane.getL
ength()-1 && l.isCellEmpty(0) && !TrafficLightController.getInstance().areLights
Enabled()))
                {
                    oldLane.setCell(null, oldLane.getLength() - 1);
                    this.setCurrentLane(l);
                    this.setCurrentCell(0, l);
                    this.getCurrentLane().setCell(this,0);
```

```
                    return 1;
                }

            else {
                this.setVehicleState(0);
                return 0;}

    }




    /**
     * This is the method that gets called when the ticker terminates (i.e. the
stop() method was called
     * on the ticker). Left not implemented on purpose
     */
    @Override
    public void onCompleted() {    }

    /**
     * If there's some error with the ticker and this subscriber, this method wo
uld call.
     * Left not implemented on purpose
     * @param throwable
     */
    @Override
    public void onError(Throwable throwable) {    }

    /**
     * This is like the onTick method. This is what cars would do when the ticke
r ticks.
     * @param aLong this gives the current time in the system to the car (althou
gh it is probably not required)
     */
    @Override
    public void onNext(Long aLong) {
//        System.out.print("Tick! " + aLong + "     ");
//        System.out.println("Car: "+ this.getId()+"  Location: " + this.getCurr
entCoordinate().getX() + "," + this.getCurrentCoordinate().getY());
        timesTicked++;
        try {
            if (vehicleBehavior == VehicleBehavior.AVERAGE) {
                VehicleController.moveOnTick(this,1);
            } else if (vehicleBehavior == VehicleBehavior.AGGRESSIVE) {
                VehicleController.moveOnTick(this,2);
            } else if (vehicleBehavior == VehicleBehavior.CAUTIOUS) {
                VehicleController.moveOnTick(this,1);
            } else
            {
                VehicleController.moveOnTick(this,1); // default behaviour
            }
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
package londonsw.view.mapcreation;

/**
 * Created by violet on 17/03/2016.
 */
public enum ComponentType {
    INTERSECTION,
    ROADNS,
    ROADEW,
    MAP_SQUARE,
    GRASS,
    NOTHING
}
```

```
package londonsw.view.mapcreation;

import javafx.application.Platform;
import javafx.beans.value.ObservableValue;
import javafx.collections.ObservableList;
import javafx.fxml.FXMLLoader;
import javafx.geometry.Insets;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.stage.Stage;
import londonsw.controller.MapMakerController;
import londonsw.controller.StartUpController;
import londonsw.model.simulation.Map;
import londonsw.model.simulation.Ticker;
import londonsw.model.simulation.components.*;
import londonsw.view.simulation.MapGridGUIDecorator;

import java.util.ArrayList;
import java.util.Optional;


@SuppressWarnings("Duplicates")
public class MapMakerScreen {

    private int width;
    private int height;

    private ImageView intersectionImgView;
    private ImageView roadNSImgView;
    private ImageView roadEWImgView;
    private ImageView grassImgView;

    /**
     * Creates a new MapMaker screen
     * @param width the width the user chose for their map
     * @param height the height the user chose for their map
     */
    public MapMakerScreen(int width, int height) {
        this.width = width;
        this.height = height;
    }

    /**
     * Draws the MapMaker screen and displays it to the user
     * @param primaryStage the stage to show it in
     * @throws Exception
     */
    public void drawScreen(Stage primaryStage) throws Exception {
        // Create the base BorderPane for the whole window
```

```
        BorderPane borderPane = new BorderPane();
        borderPane.setStyle("-fx-background-color: papayawhip");

        // Add some instructions to the user
        String text = "Instructions:\n" +
                "1. Click on the map component that you would like to place in the map\n" +
                "2. Click on the place in the map where you want to place the component\n" +
                "3. Repeat until you built the map you want!\n" +
                "4. Hit the 'Save' button when you are done";
        Label instructions = new Label(text);
        instructions.setFont(Font.font("Arial", FontWeight.BOLD,12));
        instructions.setPadding(new Insets(5, 5, 5, 5));
        borderPane.setTop(instructions);

        // Create the blank Map
        Pane mapPane = new Pane();
        Map map = new Map(width, height);
        MapGridGUIDecorator mapGridGUIDecorator = new MapGridGUIDecorator(map.ge
tGrid());
        ResizeFactor rf = ResizeFactor.getSuggestedResizeFactor(width, height);
        mapGridGUIDecorator.setResizeFactor(rf);
        GridPane mapGridPane = mapGridGUIDecorator.drawComponents();
        mapGridPane.setPadding(new Insets(0,0,5,5));
        mapPane.getChildren().add(mapGridPane);
        borderPane.setCenter(mapPane);
        MapMakerController.setCurrentFocused(ComponentType.NOTHING);

        VBox sideComponents = new VBox();

        /* Add "Components" label */
        Label componentsLabel = new Label("Components");
        componentsLabel.setFont(Font.font("Arial",FontWeight.EXTRA_BOLD,14));
        componentsLabel.setPadding(new Insets(15,5,0,20));
        sideComponents.getChildren().add(componentsLabel);

        /* Add Intersection square image */
        VBox intersectionPane = new VBox();
        Label intersectionLabel = new Label("Intersection");
        intersectionLabel.setPadding(new Insets(5,5,0,30));
        intersectionLabel.setFont(Font.font("Arial",FontWeight.SEMI_BOLD,12));
        Image intersectionImg = new Image("IntersectionX.png",60,60,true,false);
        intersectionImgView = new ImageView(intersectionImg);
        StackPane intersectionStackPane = new StackPane(intersectionImgView);
        intersectionStackPane.setPadding(new Insets(0,10,10,10));
        intersectionPane.getChildren().add(intersectionLabel);
        intersectionPane.getChildren().add(intersectionStackPane);
        sideComponents.getChildren().add(intersectionPane);

        /* Add RoadNS square image */
        VBox roadNSPane = new VBox();
        Label roadNSLabel = new Label("Road (North-South)");
        roadNSLabel.setPadding(new Insets(5,5,0,15));
        roadNSLabel.setFont(Font.font("Arial",FontWeight.SEMI_BOLD,12));
        Image roadNSImg = new Image("RoadBackgroundNS.png",60,60,true,false);
        roadNSImgView = new ImageView(roadNSImg);
        StackPane roadNSStackPane = new StackPane(roadNSImgView);
        roadNSStackPane.setPadding(new Insets(0,10,10,10));
        roadNSPane.getChildren().add(roadNSLabel);
        roadNSPane.getChildren().add(roadNSStackPane);
```

```
        sideComponents.getChildren().add(roadNSPane);

        /* Add RoadEW square image */
        VBox roadEWPane = new VBox();
        Label roadEWLabel = new Label("Road (East-West)");
        roadEWLabel.setPadding(new Insets(5,5,0,15));
        roadEWLabel.setFont(Font.font("Arial",FontWeight.SEMI_BOLD,12));
        Image roadEWImg = new Image("RoadBackgroundEW.png",60,60,true,false);
        roadEWImgView = new ImageView(roadEWImg);
        StackPane roadEWStackPane = new StackPane(roadEWImgView);
        roadEWStackPane.setPadding(new Insets(0,10,10,10));
        roadEWPane.getChildren().add(roadEWLabel);
        roadEWPane.getChildren().add(roadEWStackPane);
        sideComponents.getChildren().add(roadEWPane);

        /* Add Grass square image to empty out cells */
        VBox grassPane = new VBox();
        Label grassLabel = new Label("Grass (clear square)");
        grassLabel.setPadding(new Insets(5,5,0,15));
        grassLabel.setFont(Font.font("Arial",FontWeight.SEMI_BOLD,12));
        Image grassImg = new Image("Grass.png",60,60,true,false);
        grassImgView = new ImageView(grassImg);
        StackPane grassStackPane = new StackPane(grassImgView);
        grassStackPane.setPadding(new Insets(0,10,10,10));
        grassPane.getChildren().add(grassLabel);
        grassPane.getChildren().add(grassStackPane);
        sideComponents.getChildren().add(grassPane);

        /* Add Save, Reset buttons */
        VBox buttonsPane = new VBox();
        buttonsPane.setPadding(new Insets(0,0,0,10));
        Label toolsLabel = new Label("Tools");
        toolsLabel.setFont(Font.font("Arial",FontWeight.EXTRA_BOLD,14));
        toolsLabel.setPadding(new Insets(15,5,5,35));
        buttonsPane.getChildren().add(toolsLabel);
        Insets padding = new Insets(0,0,5,0);
        Button saveButton = new Button("Save Map");
        StackPane saveButtonPane = new StackPane(saveButton);
        saveButtonPane.setPadding(padding);
        saveButton.setStyle("-fx-base:Gold");
        saveButton.setFont(Font.font("System Bold Italic", FontWeight.BOLD, 13));
        buttonsPane.getChildren().add(saveButtonPane);
        Button resetButton = new Button("Reset Map");
        resetButton.setStyle("-fx-base:Gold");
        resetButton.setFont(Font.font("System Bold Italic", FontWeight.BOLD, 13));
        StackPane resetButtonPane = new StackPane(resetButton);
        resetButtonPane.setPadding(padding);
        buttonsPane.getChildren().add(resetButtonPane);
        Button backButton = new Button("Go Back");
        backButton.setStyle("-fx-base:Gold");
        backButton.setFont(Font.font("System Bold Italic", FontWeight.BOLD, 13));
        StackPane backButtonPane = new StackPane(backButton);
        backButtonPane.setPadding(padding);
        buttonsPane.getChildren().add(backButtonPane);

        sideComponents.getChildren().add(buttonsPane);

        Ticker.start();
```

```
        /* Add click processing for Map grid squares */
        for(int i = 0; i < height; i++) {
            for(int j = 0; j < width; j++) {
                Node current = getNodeFromIndex(i, j, mapGridPane);
                final int x = j;
                final int y = i;
                current.setOnMouseClicked((MouseEvent click) -> {
                    MapMakerController.setPreviousFocused(MapMakerController.get
CurrentFocused());
                    MapMakerController.setCurrentFocused(ComponentType.MAP_SQUAR
E);
                    current.requestFocus();
                });
                current.focusedProperty().addListener((ObservableValue<? extends
 Boolean> observable, Boolean oldValue, Boolean newValue) -> {
                    ComponentType previous = MapMakerController.getPreviousFocus
ed();
                    if(previous == ComponentType.INTERSECTION) {
                        addIntersection(x,y,map,mapGridGUIDecorator,mapGridPane,
intersectionImgView);
                    }
                    else if(previous == ComponentType.ROADNS) {
                        addRoadNS(x,y,map,mapGridGUIDecorator,mapGridPane,roadNS
ImgView);
                    }
                    else if(previous == ComponentType.ROADEW) {
                        addRoadEW(x,y,map,mapGridGUIDecorator,mapGridPane,roadEW
ImgView);
                    }
                    else if(previous == ComponentType.GRASS) {
                        addGrass(x,y,map,mapGridGUIDecorator,mapGridPane,grassIm
gView);
                    }
                });
            }
        }

        /* Add intersection icon click processing */
        DropShadow ds = new DropShadow(15, Color.BLUE);
        intersectionImgView.setOnMouseClicked(click -> {
            MapMakerController.setPreviousFocused(MapMakerController.getCurrentF
ocused());
            MapMakerController.setCurrentFocused(ComponentType.INTERSECTION);
            intersectionImgView.requestFocus();
        });
        intersectionImgView.focusedProperty().addListener((ObservableValue<? ext
ends Boolean> observable, Boolean oldValue, Boolean newValue) -> {
            if(newValue)
                intersectionImgView.setEffect(ds);
            else
                intersectionImgView.setEffect(null);
        });

        /* Add roadNS icon click processing */
        roadNSImgView.setOnMouseClicked(click -> {
            MapMakerController.setPreviousFocused(MapMakerController.getCurrentF
ocused());
            MapMakerController.setCurrentFocused(ComponentType.ROADNS);
            roadNSImgView.requestFocus();
```

```
        });
        roadNSImgView.focusedProperty().addListener((ObservableValue<? extends B
oolean> observable, Boolean oldValue, Boolean newValue) -> {
            if(newValue)
                roadNSImgView.setEffect(ds);
            else
                roadNSImgView.setEffect(null);
        });

        /* Add roadEW icon click processing */
        roadEWImgView.setOnMouseClicked(click -> {
            MapMakerController.setPreviousFocused(MapMakerController.getCurrentF
ocused());
            MapMakerController.setCurrentFocused(ComponentType.ROADEW);
            roadEWImgView.requestFocus();
        });
        roadEWImgView.focusedProperty().addListener((ObservableValue<? extends B
oolean> observable, Boolean oldValue, Boolean newValue) -> {
            if(newValue)
                roadEWImgView.setEffect(ds);
            else
                roadEWImgView.setEffect(null);
        });

        /* Add grass icon click processing */
        grassImgView.setOnMouseClicked(click -> {
            MapMakerController.setPreviousFocused(MapMakerController.getCurrentF
ocused());
            MapMakerController.setCurrentFocused(ComponentType.GRASS);
            grassImgView.requestFocus();
        });
        grassImgView.focusedProperty().addListener((ObservableValue<? extends Bo
olean> observable, Boolean oldValue, Boolean newValue) -> {
            if(newValue)
                grassImgView.setEffect(ds);
            else
                grassImgView.setEffect(null);
        });

        /* Add save button functionality */
        saveButton.setOnMouseClicked(click -> {
            TextInputDialog nameDialog = new TextInputDialog();
            nameDialog.setTitle("Save Map");
            nameDialog.setHeaderText("Please provide a name for your map (no spaces or special char
acters).\nSaved maps go into the /maps directory of your working directory.");
            nameDialog.setContentText("File name");
            Button btOk = (Button) nameDialog.getDialogPane().lookupButton(Butto
nType.OK);
            TextField textfield = nameDialog.getEditor();
            Platform.runLater(() -> textfield.requestFocus());
            btOk.setDisable(true);
            textfield.textProperty().addListener((observable, oldValue, newValu
e) -> {
                btOk.setDisable(newValue.trim().isEmpty());
            }));

            Optional<String> result = nameDialog.showAndWait();
            result.ifPresent(name -> {
                name = name.concat(".map");
```

```
                try {
                    Map finalMap = buildAndSaveMap(map);
                    finalMap.saveMap(name);
                    goBack(primaryStage);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            });
        });

        resetButton.setOnMouseClicked(click -> {
            for(int y = 0; y < height; y++) {
                for(int x = 0; x < width; x++) {
                    Component component = map.getAtLocation(new Coordinate(x, y)
);
                    if(component instanceof Road || component instanceof Interse
ction) {
                        addGrass(x,y,map,mapGridGUIDecorator,mapGridPane,grassIm
gView);
                    }
                }
            }
        });

        backButton.setOnMouseClicked(click -> {
            try {
                goBack(primaryStage);
            } catch (Exception e) {
                e.printStackTrace();
            }
        });

        borderPane.setRight(sideComponents);
        Scene scene = new Scene(borderPane);
        primaryStage.setScene(scene);
        primaryStage.centerOnScreen();
        primaryStage.setResizable(false);
    }

    /**
     * Build the map that the user drew into a complete and connected map
     * @param map the map that the user built
     * @return a fixed map that has all roads and intersections connected
     * @throws Exception
     */
    private Map buildAndSaveMap(Map map) throws Exception {
        System.out.println("Building and saving map...");
        int width = map.getWidth();
        int height = map.getHeight();
        Map fixed = new Map(width,height);

        for(int y = 0; y < height; y++) {
            for(int x = 0; x < width; x++) {
                Component current = map.getGrid().get(x, y);
                if(current instanceof Intersection) {
                    Coordinate location = new Coordinate(x, y);
                    Intersection i = new Intersection(location);
                    fixed.addIntersection(i);
                    deleteFromOldMap(map, location, location);
```

```
                }
                else if(current instanceof Road) {
                    Road road = (Road) current;
                    Coordinate lastKnownCoord = road.getEndLocation();

                    if(road.runsVertically()) {
                        if(lastKnownCoord.getY() != height-1) {
                            Component next = map.getGrid().get(x,y++);
                            while(next != null && next instanceof Road) {
                                lastKnownCoord = ((Road) next).getEndLocation();
                                if(y == height) break;
                                next = map.getGrid().get(x, y++);
                            }
                            y = road.getStartLocation().getY(); // go back to th
e row we started at
                        }
                        Coordinate start = road.getStartLocation();
                        Coordinate end = lastKnownCoord;
                        Road newRoad = new Road(start, end);
                        newRoad.addLane(new Lane(end, start, MapDirection.NORTH)
);
                        newRoad.addLane(new Lane(start, end, MapDirection.SOUTH)
);
                        fixed.addRoad(newRoad);
                        deleteFromOldMap(map, start, end);
                    }
                    else {
                        if(lastKnownCoord.getX() != width-1) {
                            Component next = map.getGrid().get(x++, y);
                            while (next != null && next instanceof Road) {
                                lastKnownCoord = ((Road) next).getEndLocation();
                                if(x == width) break;
                                next = map.getGrid().get(x++, y);
                            }
                            x = x - 2; // we overshot by 1, so go back, and loop
 will increment, so go back another
                        }
                        Coordinate start = road.getStartLocation();
                        Coordinate end = lastKnownCoord;
                        Road newRoad = new Road(start, end);
                        newRoad.addLane(new Lane(start, end, MapDirection.EAST))
;
                        newRoad.addLane(new Lane(end, start, MapDirection.WEST))
;
                        fixed.addRoad(newRoad);
                        deleteFromOldMap(map, start, end);
                    }
                }
            }
        }

        assignIntersectionsToRoads(fixed);
        return fixed;
    }

    /**
     * Gets the Node at a given location in the GridPane
     * @param row the row (y-coordinate) where to get the Node
     * @param column the column (x-coordinate) where to get the Node
```

```
     * @param gridPane the GridPane to get a Node from
     * @return the Node at that given location from the GridPane
     */
    private Node getNodeFromIndex(int row, int column, GridPane gridPane) {
        Node result = null;
        ObservableList<Node> childrens = gridPane.getChildren();
        for(Node node : childrens) {
            if(gridPane.getRowIndex(node) == row && gridPane.getColumnIndex(node
) == column) {
                result = node;
                break;
            }
        }
        return result;
    }

    /**
     * Adds an intersection to the map where the user clicks
     *
     * @param x the x coordinate where to add the intersection
     * @param y the y coordinate where to add the intersection
     * @param map the map to add the intersection to
     * @param mapGridGUIDecorator the GUI decorator associated with this map
     * @param mapGridPane the gridPane that would need to be updated with the ne
w view
     * @param imgView the associated image to place in the x,y cell
     */
    private void addIntersection(int x, int y, Map map, MapGridGUIDecorator mapG
ridGUIDecorator, GridPane mapGridPane, ImageView imgView) {
        Coordinate coord = new Coordinate(x,y);
        Intersection intersection = new Intersection(coord);
        map.addIntersection(intersection);
        StackPane sp = mapGridGUIDecorator.redrawCell(x,y,mapGridPane);

        sp.setOnMouseClicked(click -> {
            ComponentType currentFocused = MapMakerController.getCurrentFocused(
);
            if(currentFocused == ComponentType.ROADNS) {
                addRoadNS(x,y,map,mapGridGUIDecorator,mapGridPane,roadNSImgView)
;
            } else if(currentFocused == ComponentType.ROADEW) {
                addRoadEW(x,y,map,mapGridGUIDecorator,mapGridPane,roadEWImgView)
;
            } else if(currentFocused == ComponentType.GRASS) {
                addGrass(x,y,map,mapGridGUIDecorator,mapGridPane,grassImgView);
            }
        });

        // put focus back on Intersection
        MapMakerController.setPreviousFocused(MapMakerController.getCurrentFocus
ed());
        MapMakerController.setCurrentFocused(ComponentType.INTERSECTION);
        imgView.requestFocus();
    }

    /**
     * Adds a section of road with 2 lanes that travels in the directions north
and south
     *
```

```
     * @param x the x coordinate where to add the road
     * @param y the y coordinate where to add the road
     * @param map the map to add the road to
     * @param mapGridGUIDecorator the GUI decorator associated with this map
     * @param mapGridPane the gridPane that would need to be updated with the ne
w view
     * @param imgView the associated image to place in the x,y cell
     */
    private void addRoadNS(int x, int y, Map map, MapGridGUIDecorator mapGridGUI
Decorator, GridPane mapGridPane, ImageView imgView) {
        Coordinate coord = new Coordinate(x,y);
        Road road = new Road(coord,coord);
        try {
            road.addLane(new Lane(coord,coord,MapDirection.NORTH));
            road.addLane(new Lane(coord,coord,MapDirection.SOUTH));
            map.addRoad(road);
            StackPane sp = mapGridGUIDecorator.redrawCell(x,y,mapGridPane);

            sp.setOnMouseClicked(click -> {
                ComponentType currentFocused = MapMakerController.getCurrentFocu
sed();
                if(currentFocused == ComponentType.INTERSECTION) {
                    addIntersection(x,y,map,mapGridGUIDecorator,mapGridPane,inte
rsectionImgView);
                } else if(currentFocused == ComponentType.GRASS) {
                    addGrass(x,y,map,mapGridGUIDecorator,mapGridPane,grassImgVie
w);
                } else if(currentFocused == ComponentType.ROADEW) {
                    addRoadEW(x,y,map,mapGridGUIDecorator,mapGridPane,roadEWImgV
iew);
                }
            });

            // put focus back on RoadNS
            MapMakerController.setPreviousFocused(MapMakerController.getCurrentF
ocused());
            MapMakerController.setCurrentFocused(ComponentType.ROADNS);
            imgView.requestFocus();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Adds a section of road with 2 lanes that travels in the directions east a
nd west
     *
     * @param x the x coordinate where to add the road
     * @param y the y coordinate where to add the road
     * @param map the map to add the road to
     * @param mapGridGUIDecorator the GUI decorator associated with this map
     * @param mapGridPane the gridPane that would need to be updated with the ne
w view
     * @param imgView the associated image to place in the x,y cell
     */
    private void addRoadEW(int x, int y, Map map, MapGridGUIDecorator mapGridGUI
Decorator, GridPane mapGridPane, ImageView imgView) {
        Coordinate coord = new Coordinate(x,y);
        Road road = new Road(coord,coord);
```

```
        try {
            road.addLane(new Lane(coord,coord,MapDirection.EAST));
            road.addLane(new Lane(coord,coord,MapDirection.WEST));
            map.addRoad(road);
            StackPane sp = mapGridGUIDecorator.redrawCell(x,y,mapGridPane);

            sp.setOnMouseClicked(click -> {
                ComponentType currentFocused = MapMakerController.getCurrentFocu
sed();
                if(currentFocused == ComponentType.INTERSECTION) {
                    addIntersection(x,y,map,mapGridGUIDecorator,mapGridPane,inte
rsectionImgView);
                } else if(currentFocused == ComponentType.ROADNS) {
                    addRoadNS(x,y,map,mapGridGUIDecorator,mapGridPane,roadNSImgV
iew);
                } else if(currentFocused == ComponentType.GRASS) {
                    addGrass(x,y,map,mapGridGUIDecorator,mapGridPane,grassImgVie
w);
                }
            });

            // put focus back on RoadEW
            MapMakerController.setPreviousFocused(MapMakerController.getCurrentF
ocused());
            MapMakerController.setCurrentFocused(ComponentType.ROADEW);
            imgView.requestFocus();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Adds a "grass" component to the map, aka a null component
     * Useful if the user wants to delete a map component they placed in the map
     *
     * @param x the x coordinate where to add the grass
     * @param y the y coordinate where to add the grass
     * @param map the map to add the grass to
     * @param mapGridGUIDecorator the GUI decorator associated with this map
     * @param mapGridPane the gridPane that would need to be updated with the ne
w view
     * @param imgView the associated image to place in the x,y cell
     */
    private void addGrass(int x, int y, Map map, MapGridGUIDecorator mapGridGUID
ecorator, GridPane mapGridPane, ImageView imgView) {
        Coordinate coord = new Coordinate(x,y);
        map.clearCell(coord);

        StackPane sp = mapGridGUIDecorator.redrawCell(x,y,mapGridPane);

        sp.setOnMouseClicked(click -> {
            ComponentType currentFocused = MapMakerController.getCurrentFocused(
);
            if(currentFocused == ComponentType.INTERSECTION) {
                addIntersection(x,y,map,mapGridGUIDecorator,mapGridPane,intersec
tionImgView);
            } else if(currentFocused == ComponentType.ROADNS) {
                addRoadNS(x,y,map,mapGridGUIDecorator,mapGridPane,roadNSImgView)
;
```

```
            } else if(currentFocused == ComponentType.ROADEW) {
                addRoadEW(x,y,map,mapGridGUIDecorator,mapGridPane,roadEWImgView)
;
            }
        });

        // put focus back on Grass
        MapMakerController.setPreviousFocused(MapMakerController.getCurrentFocus
ed());
        MapMakerController.setCurrentFocused(ComponentType.GRASS);
        imgView.requestFocus();

    }

    /**
     * When saving a map, we are looping through the map that was built and addi
ng the fixed roads to a new map.
     * To prevent adding the same bits of roads again, delete all the bits from
the already added road
     *
     * @param oldMap the map from which to delete some number of components
     * @param start the start coordinate from where to begin deleting components
     * @param end the end coordinate to which we must delete all components
     */
    private void deleteFromOldMap(Map oldMap, Coordinate start, Coordinate end)
{
        int startX = start.getX();
        int startY = start.getY();
        int endX = end.getX();
        int endY = end.getY();

        if(startY == endY) { // horizontal
            for(int i = startX; i <= endX; i++) {
                oldMap.clearCell(new Coordinate(i, startY));
            }
        } else { // vertical
            for(int i = startY; i <= endY; i++) {
                oldMap.clearCell(new Coordinate(startX, i));
            }
        }
    }

    /**
     * Takes a map with disconnected Roads and Intersections and connects them.
This is like connecting nodes (intersections) to
     * edges (roads) in a directed graph.
     * @param fixed the map where components need to be connected
     */
    private void assignIntersectionsToRoads(Map fixed) throws Exception {
        ArrayList<Intersection> intersections = fixed.getIntersections();
        for(int i = 0; i < intersections.size(); i++) {
            Intersection current = intersections.get(i);
            Coordinate coord = current.getLocation();
            int x = coord.getX();
            int y = coord.getY();
            Coordinate north = (y-1 >= 0) ? new Coordinate(x, y-1) : null;
            Coordinate south = (y+1 < height) ? new Coordinate(x, y + 1) : null;
            Coordinate east = (x + 1 < width) ? new Coordinate(x + 1, y) : null;
            Coordinate west = (x - 1 >= 0) ? new Coordinate(x - 1, y) : null;
```

```
            if(north != null) {
                Component component = fixed.getAtLocation(north);
                if(component instanceof Road) {
                    current.setNorthRoad((Road) component);
                }
            }

            if(south != null) {
                Component component = fixed.getAtLocation(south);
                if(component instanceof Road) {
                    current.setSouthRoad((Road) component);
                }
            }

            if(east != null) {
                Component component = fixed.getAtLocation(east);
                if(component instanceof Road) {
                    current.setEastRoad((Road) component);
                }
            }

            if(west != null) {
                Component component = fixed.getAtLocation(west);
                if(component instanceof Road) {
                    current.setWestRoad((Road) component);
                }
            }

            current.setDefaultTrafficLightsForRoads();
        }
    }

    /**
     * Goes back to the previous screen (Choose mode screen)
     * @param stage the stage which will display the screen
     * @throws Exception
     */
    private void goBack(Stage stage) throws Exception {
        StartUpController.getInstance().goToChooseModeScreen(stage);
        stage.centerOnScreen();
        stage.setResizable(false);
    }

}
```

```
package londonsw.view.simulation;

import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import londonsw.controller.IntersectionController;
import londonsw.model.simulation.components.Intersection;
import londonsw.model.simulation.components.ResizeFactor;

import java.io.Serializable;

/**
 * Associates an Intersection GUI with an instance of an Intersection. Each has
up to 4
 * TrafficLightDecorators that live inside this GUI. The Intersection does not h
ave any
 * explicit circles itself, it relies on the circles of each TrafficLightDecorat
or that
 * is associated with each TrafficLight in the Intersection instance.
 *
 * Exactly one IntersectionDecorator is created for exactly one Intersection.
 */
public class IntersectionDecorator implements Serializable {

    private static final long serialVersionUID = -197785071328536445L;
    private Intersection intersection;
    private int width = 100;
    private int height = 100;

    private TrafficLightDecorator northLight;
    private TrafficLightDecorator eastLight;
    private TrafficLightDecorator southLight;
    private TrafficLightDecorator westLight;

    private ResizeFactor resizeFactor;

    private Group root;
    private Scene scene;

    /**
     * Creates a GUI for that specific intersection. It creates TrafficLightDeco
rators for all
     * TrafficLights that are in this intersection. It draws and displays them i
n a Scene specified
     * by the global variables of width and height.
     * @param intersection an instance of an Intersection to associate this deco
rator
     */
    public IntersectionDecorator(Intersection intersection) {
        this.intersection = intersection;
        IntersectionController.addIntersectionAndDecoratorPair(intersection,this
);
    }
```

```
    /**
     * Sets the resize factor for this intersection GUI decorator
     * @param rf the resize factor to set for this decorator
     */
    public void setResizeFactor(ResizeFactor rf) {
        resizeFactor = rf;
    }

    /**
     * Sets the north traffic light decorator for this intersection to represent
 the north traffic light
     * @param t the decorator representing the north traffic light
     */
    public void setNorthTrafficLightDecorator(TrafficLightDecorator t) {
        northLight = t;
    }

    /**
     * Sets the west traffic light decorator for this intersection to represent
the west traffic light
     * @param t the decorator representing the west traffic light
     */
    public void setWestTrafficLightDecorator(TrafficLightDecorator t) {
        westLight = t;
    }

    /**
     * Sets the south traffic light decorator for this intersection to represent
 the south traffic light
     * @param t the decorator representing the south traffic light
     */
    public void setSouthTrafficLightDecorator(TrafficLightDecorator t) {
        southLight = t;
    }

    /**
     * Sets the east traffic light decorator for this intersection to represent
the east traffic light
     * @param t the decorator representing the east traffic light
     */
    public void setEastTrafficLightDecorator(TrafficLightDecorator t) {
        eastLight = t;
    }

    /**
     * Draws the intersection to display it in the GUI. It initializes all Traff
icLightDecorators as well, to make
     * them display properly.
     * @return the StackPane to represent this intersection
     */
    public StackPane drawIntersection() {
        StackPane stackPane = new StackPane();

        String roadBackgroundPath = "IntersectionX.png";
        Image image = new Image(roadBackgroundPath);
        Image im = new Image(roadBackgroundPath, image.getHeight() * resizeFacto
r.getResizeX(), image.getWidth() * resizeFactor.getResizeY(), false, true);
        ImageView iv = new ImageView(im);
```

```
        Pane lights = new Pane();
        double middleCoordX = 50*resizeFactor.getResizeX();
        double middleCoordY = 50*resizeFactor.getResizeY();
        double edgeCloseX = 18*resizeFactor.getResizeX();
        double edgeCloseY = 18*resizeFactor.getResizeY();
        double edgeFarX = 82*resizeFactor.getResizeX();
        double edgeFarY = 82*resizeFactor.getResizeY();
        double radius = 15*resizeFactor.getResizeX();
        if(intersection.getNorthTrafficLight() != null) {
            northLight = new TrafficLightDecorator(intersection.getNorthTrafficL
ight());
            northLight.drawLight(middleCoordX,edgeCloseY,radius);
            lights.getChildren().add(northLight.getCircle());
        }
        if(intersection.getEastTrafficLight() != null) {
            eastLight = new TrafficLightDecorator(intersection.getEastTrafficLig
ht());
            eastLight.drawLight(edgeFarX,middleCoordY,radius);
            lights.getChildren().add(eastLight.getCircle());
        }
        if(intersection.getSouthTrafficLight() != null) {
            southLight = new TrafficLightDecorator(intersection.getSouthTrafficL
ight());
            southLight.drawLight(middleCoordX,edgeFarY,radius);
            lights.getChildren().add(southLight.getCircle());
        }
        if(intersection.getWestTrafficLight() != null) {
            westLight = new TrafficLightDecorator(intersection.getWestTrafficLig
ht());
            westLight.drawLight(edgeCloseX,middleCoordY,radius);
            lights.getChildren().add(westLight.getCircle());
        }

        stackPane.getChildren().add(iv);
        stackPane.getChildren().add(lights);
        return stackPane;
    }

    /**
     * This method is only for DEBUG and TESTING! Displays the intersection in t
he given stack pane
     * @param stage the Stage to display this Intersection in
     */
    public void showIntersection(Stage stage, StackPane s) {
        root = new Group();
        scene = new Scene(root, width, height, Color.BEIGE);
        root.getChildren().add(s);
        stage.setScene(scene);
        stage.show();
    }

}
```

```
package londonsw.view.simulation;

import javafx.scene.Group;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.shape.Polygon;
import londonsw.model.simulation.components.Lane;
import londonsw.model.simulation.components.ResizeFactor;

/**
 * Created by felix on 16/03/2016.
 */
public class LaneArrow extends Polygon {

    protected Lane lane;

    public Group getGroup() {
        return group;
    }

    public void setGroup(Group group) {
        this.group = group;
    }

    private Group group;

    public LaneArrow(Lane lane, Line roadLine, ResizeFactor resizeFactor) {
        this.lane = lane;

        double arrowResizeFactor = resizeFactor.getResizeX() * 2.5;

        this.getPoints().addAll(
                0.0 * arrowResizeFactor, 5.0 * arrowResizeFactor,
                -5.0 * arrowResizeFactor, -5.0 * arrowResizeFactor,
                5.0 * arrowResizeFactor, -5.0 * arrowResizeFactor
        );


        roadLine.setStrokeWidth(2 * resizeFactor.getResizeY()); //TODO avoid har
dcode

        roadLine.setStrokeWidth(2 * resizeFactor.getResizeY()); //TODO avoid har
dcode

        if (lane.getState() == 1) {
            roadLine.setStroke(Color.WHITE);
            this.setFill(Color.WHITE);
        } else {
            //lane not enabled
            roadLine.setStroke(Color.RED);
            this.setFill(Color.RED);
        }

        double angle = 0.0;

        switch (lane.getMovingDirection())
        {
            case NORTH:
                angle = -90;
```

```
                this.setTranslateY(roadLine.getStartY());
                this.setTranslateX(roadLine.getStartX());
                break;
            case SOUTH:
                angle = 90;
                this.setTranslateY(roadLine.getEndY());
                this.setTranslateX(roadLine.getEndX());
                break;
            case EAST:
                angle = 0.0;
                this.setTranslateY(roadLine.getEndY());
                this.setTranslateX(roadLine.getEndX());
                break;
            case WEST:
                angle = 180;
                this.setTranslateY(roadLine.getStartY());
                this.setTranslateX(roadLine.getStartX());
                break;
        }

        this.setRotate(angle - 90);
        //roadLine.setRotate(angle - 90);

        Group group = new Group();

        group.getChildren().addAll(roadLine,this);

        //group.setRotate(angle - 90);

        this.setGroup(group);

    }
}
```

```
package londonsw.view.simulation;

import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;
import londonsw.model.simulation.components.ResizeFactor;

/**
 * This class provides the basis for drawing map components to the GUI screen
 */
public class LayoutGUI {

    private int height;
    private int width;
    private ResizeFactor resizeFactor;

    /**
     * Gets the resize factor for the GUI square
     * @return the resize factor for this square
     */
    public ResizeFactor getResizeFactor() {
        return resizeFactor;
    }

    /**
     * Sets the resize factor for this GUI square
     * @param resizeFactor the resize factor to set for this square
     */
    public void setResizeFactor(ResizeFactor resizeFactor) {
        this.resizeFactor = resizeFactor;
    }

    /**
     * Sets the width of this GUI square
     * @param width the width to set for this GUI square
     */
    public void setWidth(int width) {
        this.width = width;
    }

    /**
     * Sets the height of this GUI square
     * @param height the height to set for this GUI square
     */
    public void setHeight(int height) {
        this.height = height;
    }

    /**
     * This method gets called to draw a Grass square in the map, i.e. when ther
e is no other map component there,
     * it will look like grass
     * @return the StackPane representing grass
     */
    public StackPane drawGrass() {
        return drawLayout("Grass.png");
    }

    /**
```

```
     * The generic function that gets called by any map component drawing functi
on. It draws the map component given
     * a path to an image.
     * @param path the path to the image for this GUI square
     * @return the StackPane representing this GUI square, based on the provided
 image
     */
    private StackPane drawLayout(String path) {
        Image image = new Image(path);

        Image im  = new Image(path,image.getWidth()*this.getResizeFactor().getRe
sizeX(),image.getHeight()*this.getResizeFactor().getResizeY(),false,false);

        ImageView iv = new ImageView(im);

        StackPane stackPane = new StackPane();

        stackPane.getChildren().add(iv);

        return stackPane;
    }
}
```

```
package londonsw.view.simulation;

/**
 * Created by felix on 12/03/2016.
 */

import londonsw.model.simulation.Map;
import londonsw.model.simulation.components.*;

/**
 * Created by felix on 12/03/2016.
 */
public class MapExamples {



    public static Map drawMap1() throws Exception {
        Map map = new Map(15,15);

        Road r1 = new Road(new Coordinate(2,1),new Coordinate(10,1));
        Road r2 = new Road(new Coordinate(2,10),new Coordinate(10,10));
        Road r3 = new Road(new Coordinate(1,2), new Coordinate(1,9));
        Road r4 = new Road(new Coordinate(11,2), new Coordinate(11,9));
        Road rExit = new Road(new Coordinate(12,1), new Coordinate(12,1));

        Road roadMatrix[] = {r1,r2,r3,r4 , rExit};

        Lane laneR1L1 = new Lane(r1.getStartLocation(),r1.getEndLocation(), MapD
irection.EAST);
        Lane laneR1L2 = new Lane(r1.getStartLocation(),r1.getEndLocation(), MapD
irection.EAST);
        Lane laneR1L3 = new Lane(r1.getEndLocation(),r1.getStartLocation(), MapD
irection.WEST);
        Lane laneR1L4 = new Lane(r1.getEndLocation(),r1.getStartLocation(), MapD
irection.WEST);

        Lane laneR2L1 = new Lane(r2.getStartLocation(),r2.getEndLocation(), MapD
irection.EAST);
        Lane laneR2L2 = new Lane(r2.getStartLocation(),r2.getEndLocation(), MapD
irection.EAST);
        Lane laneR2L3 = new Lane(r2.getEndLocation(),r2.getStartLocation(), MapD
irection.WEST);
        Lane laneR2L4 = new Lane(r2.getEndLocation(),r2.getStartLocation(), MapD
irection.WEST);

        Lane laneR3L1 = new Lane(r3.getEndLocation(),r3.getStartLocation(), MapD
irection.NORTH);
        Lane laneR3L2 = new Lane(r3.getEndLocation(),r3.getStartLocation(), MapD
irection.NORTH);
        Lane laneR3L3 = new Lane(r3.getStartLocation(),r3.getEndLocation(), MapD
irection.SOUTH);
        Lane laneR3L4 = new Lane(r3.getStartLocation(),r3.getEndLocation(), MapD
irection.SOUTH);

        Lane laneR4L1 = new Lane(r4.getEndLocation(),r4.getStartLocation(), MapD
irection.NORTH);
        Lane laneR4L2 = new Lane(r4.getEndLocation(),r4.getStartLocation(), MapD
irection.NORTH);
        Lane laneR4L3 = new Lane(r4.getStartLocation(),r4.getEndLocation(), MapD
```

```
irection.SOUTH);
        Lane laneR4L4 = new Lane(r4.getStartLocation(),r4.getEndLocation(), MapD
irection.SOUTH);

        Lane laneExitL1 = new Lane(rExit.getStartLocation(),rExit.getEndLocation
(),MapDirection.EAST);

        r1.addLane(laneR1L1);
        //r1.addLane(laneR1L2);
        //r1.addLane(laneR1L3);
        r1.addLane(laneR1L4);

        r2.addLane(laneR2L1);
        //r2.addLane(laneR2L2);
        //r2.addLane(laneR2L3);
        r2.addLane(laneR2L4);

        r3.addLane(laneR3L1);
        //r3.addLane(laneR3L2);
        //r3.addLane(laneR3L3);
        r3.addLane(laneR3L4);

        //laneR4L3.setState(0);
        //laneR4L4.setState(0);

        r4.addLane(laneR4L1);
        //r4.addLane(laneR4L2);
        //r4.addLane(laneR4L3);
        r4.addLane(laneR4L4);

        rExit.addLane(laneExitL1);

        for(int i = 0 ; i < roadMatrix.length; i++)
            map.addRoad(roadMatrix[i]);

        Intersection i1 = new Intersection(new Coordinate(1,1));
        i1.setEastRoad(r1);
        i1.setSouthRoad(r3);


        i1.setDefaultTrafficLightsForRoads();

        Intersection i2 = new Intersection(new Coordinate(11,1));
        i2.setWestRoad(r1);
        i2.setSouthRoad(r4);
        i2.setEastRoad(rExit);
        i2.setDefaultTrafficLightsForRoads();

        Intersection i3 = new Intersection(new Coordinate(1,10));
        i3.setEastRoad(r2);
        i3.setNorthRoad(r3);
        i3.setDefaultTrafficLightsForRoads();

        Intersection i4 = new Intersection(new Coordinate(11,10));

        i4.setWestRoad(r2);
        i4.setNorthRoad(r4);
        i4.setDefaultTrafficLightsForRoads();
```

```
        map.addIntersection(i1);
        map.addIntersection(i2);
        map.addIntersection(i3);
        map.addIntersection(i4);

        return  map;
    }

    public static Map drawMap1_1() throws Exception{

        Map map = new Map(6,6);

        Road R1 = new Road(new Coordinate(1,1),new Coordinate(4,1));

        Lane laneR1L1 = new Lane(R1.getStartLocation(),R1.getEndLocation(),MapDi
rection.EAST);

        R1.addLane(laneR1L1);

        map.addRoad(R1);

        return map;

    }


    public static Map dratMap4 () throws Exception {

        Map map = new Map(22, 22);

        Road r04 = new Road(new Coordinate(1, 11), new Coordinate(1, 14));
        Road r08 = new Road(new Coordinate(20, 11), new Coordinate(20, 14));
        Road r09 = new Road(new Coordinate(9, 15), new Coordinate(19, 15));
        Road r10 = new Road(new Coordinate(2, 15), new Coordinate(7, 15));
        Road r13 = new Road(new Coordinate(2, 10), new Coordinate(7, 10));
        Road r14 = new Road(new Coordinate(9, 10), new Coordinate(12, 10));
        Road r15 = new Road(new Coordinate(14, 10), new Coordinate(19, 10));
        Road r16 = new Road(new Coordinate(8, 11), new Coordinate(8, 14));
        Road r20 = new Road(new Coordinate(1, 16), new Coordinate(1, 19));
        Road r21 = new Road(new Coordinate(8, 16), new Coordinate(8, 19));
        Road r22 = new Road(new Coordinate(2, 20), new Coordinate(7, 20));
        Road r23 = new Road(new Coordinate(9, 20), new Coordinate(19, 20));
        Road r24 = new Road(new Coordinate(20, 16), new Coordinate(20, 19));

        /*This is a one lane Map
        If you want two lanes map
        comment these Lines , and uncomment The two lane code
        at the bottom
         */

/*
        r01.addLane(new Lane(r01.getStartLocation(),r01.getEndLocation(),MapDire
ction.EAST));
        r02.addLane(new Lane (r02.getEndLocation(),r02.getStartLocation(),MapDir
ection.NORTH));
        r03.addLane(new Lane(r03.getEndLocation(),r03.getStartLocation(),MapDire
ction.NORTH));
        r04.addLane(new Lane(r04.getEndLocation(),r04.getStartLocation(),MapDire
```

```
ction.NORTH));
        Lane l5 = new Lane(r05.getStartLocation(),r05.getEndLocation(),MapDirect
ion.EAST);
        //l5.setState(0);
        r05.addLane(l5);
        Lane l6 = new Lane(r06.getStartLocation(),r06.getEndLocation(),MapDirect
ion.EAST);
        r06.addLane(l6);
        //l6.setState(0);
        r07.addLane(new Lane(r07.getStartLocation(),r07.getEndLocation(),MapDire
ction.SOUTH));
        r08.addLane(new Lane(r08.getStartLocation(),r08.getEndLocation(),MapDire
ction.SOUTH));
        r09.addLane(new Lane(r09.getEndLocation(),r09.getStartLocation(),MapDire
ction.WEST));
        r10.addLane(new Lane(r10.getEndLocation(),r10.getStartLocation(),MapDire
ction.WEST));
        r11.addLane(new Lane(r11.getEndLocation(),r11.getStartLocation(),MapDire
ction.NORTH));
        r12.addLane(new Lane(r12.getStartLocation(),r12.getEndLocation(),MapDire
ction.EAST));
        r13.addLane(new Lane(r13.getStartLocation(),r13.getEndLocation(),MapDire
ction.EAST));
        r14.addLane(new Lane(r14.getStartLocation(),r14.getEndLocation(),MapDire
ction.EAST));
        r15.addLane(new Lane(r15.getStartLocation(),r15.getEndLocation(),MapDire
ction.EAST));
        r16.addLane(new Lane(r16.getEndLocation(),r16.getStartLocation(),MapDire
ction.NORTH));
        r17.addLane(new Lane(r17.getEndLocation(),r17.getStartLocation(),MapDire
ction.NORTH));
        r18.addLane(new Lane(r18.getEndLocation(),r18.getStartLocation(),MapDire
ction.NORTH));
        r19.addLane(new Lane(r19.getStartLocation(),r19.getEndLocation(),MapDire
ction.EAST));
        r20.addLane(new Lane(r20.getEndLocation(),r20.getStartLocation(),MapDire
ction.NORTH));
        r21.addLane(new Lane(r21.getEndLocation(),r21.getStartLocation(),MapDire
ction.NORTH));
        r22.addLane(new Lane(r22.getEndLocation(),r22.getStartLocation(),MapDire
ction.WEST));
        r23.addLane(new Lane(r23.getEndLocation(),r23.getStartLocation(),MapDire
ction.WEST));
        r24.addLane(new Lane(r24.getStartLocation(),r24.getEndLocation(),MapDire
ction.SOUTH));
*/

          /*This is a two lane Map
        If you want one lanes map
        comment these Lines , and uncomment The one lane code above
         */



        r04.addLane(new Lane(r04.getEndLocation(), r04.getStartLocation(), MapDi
rection.NORTH));
        r08.addLane(new Lane(r08.getEndLocation(), r08.getStartLocation(), MapDi
rection.NORTH));
        Lane l9=new Lane(r09.getStartLocation(), r09.getEndLocation(), MapDirect
```

```
ion.EAST);
         l9.setState(0);
        r09.addLane(l9);
        r10.addLane(new Lane(r10.getStartLocation(), r10.getEndLocation(), MapDi
rection.EAST));
        r13.addLane(new Lane(r13.getStartLocation(), r13.getEndLocation(), MapDi
rection.EAST));
        r14.addLane(new Lane(r14.getStartLocation(), r14.getEndLocation(), MapDi
rection.EAST));
        r15.addLane(new Lane(r15.getStartLocation(), r15.getEndLocation(), MapDi
rection.EAST));
        Lane l16 =new Lane(r16.getEndLocation(), r16.getStartLocation(), MapDire
ction.NORTH);
        l16.setState(0);
        r16.addLane(l16);
        r20.addLane(new Lane(r20.getEndLocation(), r20.getStartLocation(), MapDi
rection.NORTH));
        r21.addLane(new Lane(r21.getEndLocation(), r21.getStartLocation(), MapDi
rection.NORTH));
        r22.addLane(new Lane(r22.getStartLocation(), r22.getEndLocation(), MapDi
rection.EAST));
        r23.addLane(new Lane(r23.getStartLocation(), r23.getEndLocation(), MapDi
rection.EAST));
        r24.addLane(new Lane(r24.getEndLocation(), r24.getStartLocation(), MapDi
rection.NORTH));


        r04.addLane(new Lane(r04.getStartLocation(), r04.getEndLocation(), MapDi
rection.SOUTH));
        r08.addLane(new Lane(r08.getStartLocation(), r08.getEndLocation(), MapDi
rection.SOUTH));
        r09.addLane(new Lane(r09.getEndLocation(), r09.getStartLocation(), MapDi
rection.WEST));
        Lane l10=new Lane(r10.getEndLocation(), r10.getStartLocation(), MapDirec
tion.WEST);
        l10.setState(0);
        r10.addLane(l10);
        r13.addLane(new Lane(r13.getEndLocation(), r13.getStartLocation(), MapDi
rection.WEST));
        r14.addLane(new Lane(r14.getEndLocation(), r14.getStartLocation(), MapDi
rection.WEST));
        r15.addLane(new Lane(r15.getEndLocation(), r15.getStartLocation(), MapDi
rection.WEST));
        r16.addLane(new Lane(r16.getStartLocation(), r16.getEndLocation(), MapDi
rection.SOUTH));
        r20.addLane(new Lane(r20.getStartLocation(), r20.getEndLocation(), MapDi
rection.SOUTH));
        Lane l21=new Lane(r21.getStartLocation(), r21.getEndLocation(), MapDirec
tion.SOUTH);
        l21.setState(0);
        r21.addLane(l21);
        r22.addLane(new Lane(r22.getEndLocation(), r22.getStartLocation(), MapDi
rection.WEST));
        r23.addLane(new Lane(r23.getEndLocation(), r23.getStartLocation(), MapDi
rection.WEST));
        r24.addLane(new Lane(r24.getStartLocation(), r24.getEndLocation(), MapDi
rection.SOUTH));
```

```
        Intersection i03 = new Intersection(new Coordinate(1, 10));
        i03.setSouthRoad(r04);
        i03.setEastRoad(r13);
        //i03.setDefaultTrafficLightsForRoads();

        Intersection i04 = new Intersection(new Coordinate(1, 15));
        i04.setNorthRoad(r04);
        i04.setEastRoad(r10);
        i04.setSouthRoad(r20);
       // i04.setDefaultTrafficLightsForRoads();


        Intersection i08 = new Intersection(new Coordinate(20, 10));
        i08.setSouthRoad(r08);
        i08.setWestRoad(r15);
       // i08.setDefaultTrafficLightsForRoads();

        Intersection i09 = new Intersection(new Coordinate(20, 15));
        i09.setNorthRoad(r08);
        i09.setWestRoad(r09);
        i09.setSouthRoad(r24);
        //i09.setDefaultTrafficLightsForRoads();

        Intersection i10 = new Intersection(new Coordinate(8, 15));
        i10.setEastRoad(r09);
        i10.setWestRoad(r10);
        i10.setNorthRoad(r16);
        i10.setSouthRoad(r21);
        i10.setDefaultTrafficLightsForRoads();


        Intersection i12 = new Intersection(new Coordinate(8, 10));
        i12.setWestRoad(r13);
        i12.setEastRoad(r14);
        i12.setSouthRoad(r16);
       // i12.setDefaultTrafficLightsForRoads();

        Intersection i13 = new Intersection(new Coordinate(13, 10));
        i13.setWestRoad(r14);
        i13.setEastRoad(r15);
       // i13.setDefaultTrafficLightsForRoads();

        Intersection i15 = new Intersection(new Coordinate(1, 20));
        i15.setNorthRoad(r20);
        i15.setEastRoad(r22);
       // i15.setDefaultTrafficLightsForRoads();

        Intersection i16 = new Intersection(new Coordinate(8, 20));
        i16.setWestRoad(r22);
        i16.setNorthRoad(r21);
        i16.setEastRoad(r23);
       // i16.setDefaultTrafficLightsForRoads();

        Intersection i17 = new Intersection(new Coordinate(20, 20));
        i17.setWestRoad(r23);
        i17.setNorthRoad(r24);
        //i17.setDefaultTrafficLightsForRoads();
```

```
        map.addRoad(r04);
        map.addRoad(r08);
        map.addRoad(r09);
        map.addRoad(r10);
        map.addRoad(r13);
        map.addRoad(r14);
        map.addRoad(r15);
        map.addRoad(r16);
        map.addRoad(r20);
        map.addRoad(r21);
        map.addRoad(r22);
        map.addRoad(r23);
        map.addRoad(r24);


        map.addIntersection(i03);
        map.addIntersection(i04);
        map.addIntersection(i08);
        map.addIntersection(i09);
        map.addIntersection(i10);
        map.addIntersection(i12);
        map.addIntersection(i13);
        map.addIntersection(i15);
        map.addIntersection(i16);
        map.addIntersection(i17);

        return map;


    }


    public static Map drawTestMapBasic() throws Exception {

        Map map = new Map(10, 10);

        Road r1 = new Road(new Coordinate(2, 1), new Coordinate(8, 1));
        Road r2 = new Road(new Coordinate(1, 2), new Coordinate(1, 8));
        Road r3 = new Road(new Coordinate(9, 2), new Coordinate(9, 8));
        Road r4 = new Road(new Coordinate(2, 9), new Coordinate(8, 9));

        r1.addLane(new Lane(r1.getStartLocation(), r1.getEndLocation(), MapDirec
tion.EAST));

        r2.addLane(new Lane(r2.getEndLocation(), r2.getStartLocation(), MapDirec
tion.NORTH));

        r3.addLane(new Lane(r3.getStartLocation(), r3.getEndLocation(), MapDirec
tion.SOUTH));

        r4.addLane(new Lane(r4.getEndLocation(), r4.getStartLocation(), MapDirec
tion.WEST));

        map.addRoad(r1);
        map.addRoad(r2);
```

```
        map.addRoad(r3);
        map.addRoad(r4);

        Intersection i1 = new Intersection(new Coordinate(1, 1));
        i1.setEastRoad(r1);
        i1.setSouthRoad(r2);

        Intersection i2 = new Intersection(new Coordinate(9, 1));
        i2.setWestRoad(r1);
        i2.setSouthRoad(r3);
        Intersection i3 = new Intersection(new Coordinate(1, 9));
        i3.setNorthRoad(r2);
        i3.setEastRoad(r4);
        Intersection i4 = new Intersection(new Coordinate(9, 9));
        i4.setNorthRoad(r3);
        i4.setWestRoad(r4);

        map.addIntersection(i1);
        map.addIntersection(i2);
        map.addIntersection(i3);
        map.addIntersection(i4);

        return map;
    }

    public static Map drawTestMapSingleLine() throws Exception {

        Map map = new Map(25, 25);

        Road r1 = new Road(new Coordinate(2, 1), new Coordinate(8, 1));
        Road r2 = new Road(new Coordinate(1, 2), new Coordinate(1, 8));
        Road r3 = new Road(new Coordinate(9, 2), new Coordinate(9, 8));
        Road r4 = new Road(new Coordinate(2, 9), new Coordinate(8, 9));

        Road r5 = new Road(new Coordinate(10, 1), new Coordinate(16, 1));
        Road r6 = new Road(new Coordinate(17, 2), new Coordinate(17, 8));
        Road r7 = new Road(new Coordinate(10, 9), new Coordinate(16, 9));

        Lane disabledLane = new Lane(r3.getStartLocation(), r3.getEndLocation(),
 MapDirection.SOUTH);

        //disable lane
        disabledLane.setState(0);

        r1.addLane(new Lane(r1.getStartLocation(), r1.getEndLocation(), MapDirec
tion.EAST));

        r2.addLane(new Lane(r2.getEndLocation(), r2.getStartLocation(), MapDirec
tion.NORTH));

        r3.addLane(disabledLane);

        r4.addLane(new Lane(r4.getEndLocation(), r4.getStartLocation(), MapDirec
tion.WEST));

        r5.addLane(new Lane(r5.getStartLocation(), r5.getEndLocation(), MapDirec
tion.EAST));

        r6.addLane(new Lane(r6.getStartLocation(), r6.getEndLocation(), MapDirec
```

```
tion.SOUTH));

        r7.addLane(new Lane(r7.getEndLocation(), r7.getStartLocation(), MapDirec
tion.WEST));

        map.addRoad(r1);
        map.addRoad(r2);
        map.addRoad(r3);
        map.addRoad(r4);
        map.addRoad(r5);
        map.addRoad(r6);
        map.addRoad(r7);

        Intersection i1 = new Intersection(new Coordinate(1, 1));
        i1.setEastRoad(r1);
        i1.setSouthRoad(r2);

        Intersection i2 = new Intersection(new Coordinate(9, 1));
        i2.setWestRoad(r1);
        i2.setSouthRoad(r3);
        i2.setEastRoad(r5);

        Intersection i3 = new Intersection(new Coordinate(1, 9));
        i3.setNorthRoad(r2);
        i3.setEastRoad(r4);

        Intersection i4 = new Intersection(new Coordinate(9, 9));
        i4.setNorthRoad(r3);
        i4.setWestRoad(r4);
        i4.setEastRoad(r7);

        Intersection i5 = new Intersection(new Coordinate(17, 1));
        i5.setWestRoad(r5);
        i5.setSouthRoad(r6);

        Intersection i6 = new Intersection(new Coordinate(17, 9));
        i6.setNorthRoad(r6);
        i6.setWestRoad(r7);

        map.addIntersection(i1);
        map.addIntersection(i2);
        map.addIntersection(i3);
        map.addIntersection(i4);
        map.addIntersection(i5);
        map.addIntersection(i6);

        return map;
    }



    public static Map drawTestMapSimple() throws Exception {

        Map map = new Map(10,10);

        Road r1 = new Road(new Coordinate(2,1), new Coordinate(8,1));
        Road r2 = new Road(new Coordinate(1,2), new Coordinate(1,8));
        Road r3 = new Road(new Coordinate(9,2), new Coordinate(9,8));
        Road r4 = new Road(new Coordinate(2,9), new Coordinate(8,9));
```

```
        r1.addLane(new Lane(r1.getStartLocation(),r1.getEndLocation(), MapDirect
ion.EAST));
        r1.addLane(new Lane(r1.getEndLocation() ,r1.getStartLocation(), MapDirec
tion.WEST));

        r2.addLane(new Lane(r2.getEndLocation(),r2.getStartLocation(), MapDirect
ion.NORTH));
        r2.addLane(new Lane(r2.getStartLocation(),r2.getEndLocation(), MapDirect
ion.SOUTH));

        r3.addLane(new Lane(r3.getEndLocation(),r3.getStartLocation(), MapDirect
ion.NORTH));
        r3.addLane(new Lane(r3.getStartLocation(),r3.getEndLocation(), MapDirect
ion.SOUTH));

        r4.addLane(new Lane(r4.getStartLocation(),r4.getEndLocation(), MapDirect
ion.EAST));
        r4.addLane(new Lane(r4.getEndLocation(),r4.getStartLocation(), MapDirect
ion.WEST));

        map.addRoad(r1);
        map.addRoad(r2);
        map.addRoad(r3);
        map.addRoad(r4);

        Intersection i1 = new Intersection(new Coordinate(1,1));
        i1.setEastRoad(r1);
        i1.setSouthRoad(r2);

        Intersection i2 = new Intersection(new Coordinate(9,1));
        i2.setWestRoad(r1);
        i2.setSouthRoad(r3);
        Intersection i3 = new Intersection(new Coordinate(1,9));
        i3.setNorthRoad(r2);
        i3.setEastRoad(r4);
        Intersection i4 = new Intersection(new Coordinate(9,9));
        i4.setNorthRoad(r3);
        i4.setWestRoad(r4);

        map.addIntersection(i1);
        map.addIntersection(i2);
        map.addIntersection(i3);
        map.addIntersection(i4);

        return map;
    }

    public static Map drawTestMapBig() throws Exception {

        Map map = new Map(20,20);

        Road r1 = new Road(new Coordinate(2,1), new Coordinate(8,1));
        Road r2 = new Road(new Coordinate(1,2), new Coordinate(1,8));
        Road r3 = new Road(new Coordinate(9,2), new Coordinate(9,8));
        Road r4 = new Road(new Coordinate(2,9), new Coordinate(8,9));
```

```
        Road r5 = new Road(new Coordinate(10,1), new Coordinate(16,1));
        Road r6 = new Road(new Coordinate(17,2), new Coordinate(17,8));
        Road r7 = new Road(new Coordinate(10,9), new Coordinate(16,9));

        Lane disabledLane = new Lane(r3.getStartLocation(),r3.getEndLocation(),
MapDirection.SOUTH);

        //disable lane
        disabledLane.setState(0);

        r1.addLane(new Lane(r1.getStartLocation(),r1.getEndLocation(), MapDirect
ion.EAST));
        r1.addLane(new Lane(r1.getEndLocation() ,r1.getStartLocation(), MapDirec
tion.WEST));

        r2.addLane(new Lane(r2.getEndLocation(),r2.getStartLocation(), MapDirect
ion.NORTH));
        r2.addLane(new Lane(r2.getStartLocation(),r2.getEndLocation(), MapDirect
ion.SOUTH));

        r3.addLane(new Lane(r3.getEndLocation(),r3.getStartLocation(), MapDirect
ion.NORTH));
        r3.addLane(disabledLane);

        r4.addLane(new Lane(r4.getStartLocation(),r4.getEndLocation(), MapDirect
ion.EAST));
        r4.addLane(new Lane(r4.getEndLocation(),r4.getStartLocation(), MapDirect
ion.WEST));

        r5.addLane(new Lane(r5.getStartLocation(),r5.getEndLocation(), MapDirect
ion.EAST));
        r5.addLane(new Lane(r5.getEndLocation(),r5.getStartLocation(), MapDirect
ion.WEST));

        r6.addLane(new Lane(r6.getEndLocation(),r6.getStartLocation(), MapDirect
ion.NORTH));
        r6.addLane(new Lane(r6.getStartLocation(),r6.getEndLocation(), MapDirect
ion.SOUTH));

        r7.addLane(new Lane(r7.getStartLocation(),r7.getEndLocation(), MapDirect
ion.EAST));
        r7.addLane(new Lane(r7.getEndLocation(),r7.getStartLocation(), MapDirect
ion.WEST));

        map.addRoad(r1);
        map.addRoad(r2);
        map.addRoad(r3);
        map.addRoad(r4);
        map.addRoad(r5);
        map.addRoad(r6);
        map.addRoad(r7);

        Intersection i1 = new Intersection(new Coordinate(1,1));
        i1.setEastRoad(r1);
        i1.setSouthRoad(r2);

        Intersection i2 = new Intersection(new Coordinate(9,1));
        i2.setWestRoad(r1);
        i2.setSouthRoad(r3);
```

```
        i2.setEastRoad(r5);

        Intersection i3 = new Intersection(new Coordinate(1,9));
        i3.setNorthRoad(r2);
        i3.setEastRoad(r4);

        Intersection i4 = new Intersection(new Coordinate(9,9));
        i4.setNorthRoad(r3);
        i4.setWestRoad(r4);
        i4.setEastRoad(r7);

        Intersection i5 = new Intersection(new Coordinate(17,1));
        i5.setWestRoad(r5);
        i5.setSouthRoad(r6);

        Intersection i6 = new Intersection(new Coordinate(17,9));
        i6.setNorthRoad(r6);
        i6.setWestRoad(r7);

        map.addIntersection(i1);
        map.addIntersection(i2);
        map.addIntersection(i3);
        map.addIntersection(i4);
        map.addIntersection(i5);
        map.addIntersection(i6);

        return map;
    }

    //Added new Map with two lanes
    public static Map drawTestMapExample() throws Exception {

        Map map = new Map(22,22);

        Road r01 = new Road(new Coordinate(2,1), new Coordinate(4,1));
        Road r02 = new Road(new Coordinate(1,2), new Coordinate(1,4));
        Road r03 = new Road(new Coordinate(1,6), new Coordinate(1,9));
        Road r04 = new Road(new Coordinate(1,11), new Coordinate(1,14));
        Road r05 = new Road(new Coordinate(6,1), new Coordinate(12,1));
        Road r06 = new Road(new Coordinate(14,1), new Coordinate(19,1));
        Road r07 = new Road(new Coordinate(20,2), new Coordinate(20,9));
        Road r08 = new Road(new Coordinate(20,11), new Coordinate(20,14));
        Road r09 = new Road(new Coordinate(9,15), new Coordinate(19,15));
        Road r10 = new Road(new Coordinate(2,15), new Coordinate(7,15));
        Road r11 = new Road(new Coordinate(5,2), new Coordinate(5,4));
        Road r12 = new Road(new Coordinate(2,5), new Coordinate(4,5));
        Road r13 = new Road(new Coordinate(2,10), new Coordinate(7,10));
        Road r14 = new Road(new Coordinate(9,10), new Coordinate(12,10));
        Road r15 = new Road(new Coordinate(14,10), new Coordinate(19,10));
        Road r16 = new Road(new Coordinate(8,11), new Coordinate(8,14));
        Road r17 = new Road(new Coordinate(13,2), new Coordinate(13,4));
        Road r18 = new Road(new Coordinate(13,6), new Coordinate(13,9));
        Road r19 = new Road(new Coordinate(6,5),new Coordinate(12,5));
        Road r20 = new Road(new Coordinate(1,16),new Coordinate(1,19));
        Road r21 = new Road(new Coordinate(8,16),new Coordinate(8,19));
        Road r22 = new Road(new Coordinate(2,20),new Coordinate(7,20));
        Road r23 = new Road(new Coordinate(9,20),new Coordinate(19,20));
        Road r24 = new Road(new Coordinate(20,16), new Coordinate(20,19));
```

```
        r01.addLane(new Lane(r01.getStartLocation(),r01.getEndLocation(),MapDire
ction.EAST));
        r02.addLane(new Lane (r02.getEndLocation(),r02.getStartLocation(),MapDir
ection.NORTH));
        r03.addLane(new Lane(r03.getEndLocation(),r03.getStartLocation(),MapDire
ction.NORTH));
        r04.addLane(new Lane(r04.getEndLocation(),r04.getStartLocation(),MapDire
ction.NORTH));
        Lane l5 = new Lane(r05.getStartLocation(),r05.getEndLocation(),MapDirect
ion.EAST);
        //l5.setState(0);
        r05.addLane(l5);
        Lane l6 = new Lane(r06.getStartLocation(),r06.getEndLocation(),MapDirect
ion.EAST);
        r06.addLane(l6);
        //l6.setState(0);
        r07.addLane(new Lane(r07.getEndLocation(),r07.getStartLocation(),MapDire
ction.NORTH));
        r08.addLane(new Lane(r08.getEndLocation(),r08.getStartLocation(),MapDire
ction.NORTH));

        Lane lane9Closed = new Lane(r09.getStartLocation(),r09.getEndLocation(),
MapDirection.EAST);

        //lane9Closed.setState(0);

        r09.addLane(lane9Closed);


        r10.addLane(new Lane(r10.getStartLocation(),r10.getEndLocation(),MapDire
ction.EAST));
        r11.addLane(new Lane(r11.getEndLocation(),r11.getStartLocation(),MapDire
ction.NORTH));
        r12.addLane(new Lane(r12.getStartLocation(),r12.getEndLocation(),MapDire
ction.EAST));

        r13.addLane(new Lane(r13.getStartLocation(),r13.getEndLocation(),MapDire
ction.EAST));
        r14.addLane(new Lane(r14.getStartLocation(),r14.getEndLocation(),MapDire
ction.EAST));
        r15.addLane(new Lane(r15.getStartLocation(),r15.getEndLocation(),MapDire
ction.EAST));
        r16.addLane(new Lane(r16.getEndLocation(),r16.getStartLocation(),MapDire
ction.NORTH));
        r17.addLane(new Lane(r17.getEndLocation(),r17.getStartLocation(),MapDire
ction.NORTH));
        r18.addLane(new Lane(r18.getEndLocation(),r18.getStartLocation(),MapDire
ction.NORTH));
        r19.addLane(new Lane(r19.getStartLocation(),r19.getEndLocation(),MapDire
ction.EAST));
        r20.addLane(new Lane(r20.getEndLocation(),r20.getStartLocation(),MapDire
ction.NORTH));

        Lane lane21NClosed = new Lane(r21.getEndLocation(),r21.getStartLocation(
),MapDirection.NORTH);
        //lane21Closed.setState(0);

        r21.addLane(lane21NClosed);
```

```
        r22.addLane(new Lane(r22.getStartLocation(),r22.getEndLocation(),MapDire
ction.EAST));
        r23.addLane(new Lane(r23.getStartLocation(),r23.getEndLocation(),MapDire
ction.EAST));
        r24.addLane(new Lane(r24.getEndLocation(),r24.getStartLocation(),MapDire
ction.NORTH));


        r01.addLane(new Lane(r01.getEndLocation(),r01.getStartLocation(),MapDire
ction.WEST));
        r02.addLane(new Lane (r02.getStartLocation(),r02.getEndLocation(),MapDir
ection.SOUTH));
        r03.addLane(new Lane(r03.getStartLocation(),r03.getEndLocation(),MapDire
ction.SOUTH));
        r04.addLane(new Lane(r04.getStartLocation(),r04.getEndLocation(),MapDire
ction.SOUTH));
        r05.addLane(new Lane(r05.getEndLocation(),r05.getStartLocation(),MapDire
ction.WEST));
        r06.addLane(new Lane(r06.getEndLocation(),r06.getStartLocation(),MapDire
ction.WEST));
        r07.addLane(new Lane(r07.getStartLocation(),r07.getEndLocation(),MapDire
ction.SOUTH));
        r08.addLane(new Lane(r08.getStartLocation(),r08.getEndLocation(),MapDire
ction.SOUTH));
        r09.addLane(new Lane(r09.getEndLocation(),r09.getStartLocation(),MapDire
ction.WEST));
        r10.addLane(new Lane(r10.getEndLocation(),r10.getStartLocation(),MapDire
ction.WEST));
        r11.addLane(new Lane(r11.getStartLocation(),r11.getEndLocation(),MapDire
ction.SOUTH));
        r12.addLane(new Lane(r12.getEndLocation(),r12.getStartLocation(),MapDire
ction.WEST));

        Lane lane13WClosed = new Lane(r13.getEndLocation(),r13.getStartLocation(
),MapDirection.WEST);
        //lane13WClosed.setState(0);

        r13.addLane(lane13WClosed);
        r14.addLane(new Lane(r14.getEndLocation(),r14.getStartLocation(),MapDire
ction.WEST));
        r15.addLane(new Lane(r15.getEndLocation(),r15.getStartLocation(),MapDire
ction.WEST));
        r16.addLane(new Lane(r16.getStartLocation(),r16.getEndLocation(),MapDire
ction.SOUTH));
        r17.addLane(new Lane(r17.getStartLocation(),r17.getEndLocation(),MapDire
ction.SOUTH));
        r18.addLane(new Lane(r18.getStartLocation(),r18.getEndLocation(),MapDire
ction.SOUTH));
        r19.addLane(new Lane(r19.getEndLocation(),r19.getStartLocation(),MapDire
ction.WEST));
        r20.addLane(new Lane(r20.getStartLocation(),r20.getEndLocation(),MapDire
ction.SOUTH));

        Lane lane21SClosed = new Lane(r21.getStartLocation(),r21.getEndLocation(
),MapDirection.SOUTH);
        //lane21SClosed.setState(0);

        r21.addLane(lane21SClosed);
```

```
        r22.addLane(new Lane(r22.getEndLocation(),r22.getStartLocation(),MapDire
ction.WEST));
        r23.addLane(new Lane(r23.getEndLocation(),r23.getStartLocation(),MapDire
ction.WEST));
        r24.addLane(new Lane(r24.getStartLocation(),r24.getEndLocation(),MapDire
ction.SOUTH));


        Intersection i01 = new Intersection(new Coordinate(1,1));
        i01.setEastRoad(r01);
        i01.setSouthRoad(r02);
        i01.setDefaultTrafficLightsForRoads();

        Intersection i02 = new Intersection(new Coordinate(1,5));
        i02.setNorthRoad(r02);
        i02.setSouthRoad(r03);
        i02.setEastRoad(r12);
        i02.setDefaultTrafficLightsForRoads();

        Intersection i03 = new Intersection(new Coordinate(1,10));
        i03.setNorthRoad(r03);
        i03.setSouthRoad(r04);
        i03.setEastRoad(r13);
        i03.setDefaultTrafficLightsForRoads();

        Intersection i04 = new Intersection(new Coordinate(1,15));
        i04.setNorthRoad(r04);
        i04.setEastRoad(r10);
        i04.setSouthRoad(r20);
        i04.setDefaultTrafficLightsForRoads();

        Intersection i05 = new Intersection(new Coordinate(5,1));
        i05.setWestRoad(r01);
        i05.setEastRoad(r05);
        i05.setSouthRoad(r11);
        i05.setDefaultTrafficLightsForRoads();

        Intersection i06 = new Intersection(new Coordinate(13,1));
        i06.setWestRoad(r05);
        i06.setEastRoad(r06);
        i06.setSouthRoad(r17);
        i06.setDefaultTrafficLightsForRoads();

        Intersection i07 = new Intersection(new Coordinate(20,1));
        i07.setWestRoad(r06);
        i07.setSouthRoad(r07);
        i07.setDefaultTrafficLightsForRoads();

        Intersection i08 = new Intersection(new Coordinate(20,10));
        i08.setNorthRoad(r07);
        i08.setSouthRoad(r08);
        i08.setWestRoad(r15);
        i08.setDefaultTrafficLightsForRoads();

        Intersection i09 = new Intersection(new Coordinate(20,15));
        i09.setNorthRoad(r08);
        i09.setWestRoad(r09);
        i09.setSouthRoad(r24);
        i09.setDefaultTrafficLightsForRoads();
```

```
        Intersection i10 = new Intersection(new Coordinate(8,15));
        i10.setEastRoad(r09);
        i10.setWestRoad(r10);
        i10.setNorthRoad(r16);
        i10.setSouthRoad(r21);
        i10.setDefaultTrafficLightsForRoads();

        Intersection i11 = new Intersection(new Coordinate(5,5));
        i11.setWestRoad(r12);
        i11.setNorthRoad(r11);
        i11.setEastRoad(r19);
        i11.setDefaultTrafficLightsForRoads();

        Intersection i12 = new Intersection(new Coordinate(8,10));
        i12.setWestRoad(r13);
        i12.setEastRoad(r14);
        i12.setSouthRoad(r16);
        i12.setDefaultTrafficLightsForRoads();    //enables tl

        Intersection i13 = new Intersection(new Coordinate(13,10));
        i13.setWestRoad(r14);
        i13.setEastRoad(r15);
        i13.setNorthRoad(r18);
        i13.setDefaultTrafficLightsForRoads();

        Intersection i14 =new Intersection((new Coordinate(13,5)));
        i14.setNorthRoad(r17);
        i14.setSouthRoad(r18);
        i14.setWestRoad(r19);
        i14.setDefaultTrafficLightsForRoads();

        Intersection i15 =new Intersection(new Coordinate(1,20));
        i15.setNorthRoad(r20);
        i15.setEastRoad(r22);
        i15.setDefaultTrafficLightsForRoads();

        Intersection i16 =new Intersection(new Coordinate(8,20));
        i16.setWestRoad(r22);
        i16.setNorthRoad(r21);
        i16.setEastRoad(r23);
        i16.setDefaultTrafficLightsForRoads();

        Intersection i17= new Intersection (new Coordinate(20,20));
        i17.setWestRoad(r23);
        i17.setNorthRoad(r24);
        i17.setDefaultTrafficLightsForRoads();

        map.addRoad(r01);
        map.addRoad(r02);
        map.addRoad(r03);
        map.addRoad(r04);
        map.addRoad(r05);
        map.addRoad(r06);
        map.addRoad(r07);
        map.addRoad(r08);
        map.addRoad(r09);
        map.addRoad(r10);
        map.addRoad(r11);
```

```
        map.addRoad(r12);
        map.addRoad(r13);
        map.addRoad(r14);
        map.addRoad(r15);
        map.addRoad(r16);
        map.addRoad(r17);
        map.addRoad(r18);
        map.addRoad(r19);
        map.addRoad(r20);
        map.addRoad(r21);
        map.addRoad(r22);
        map.addRoad(r23);
        map.addRoad(r24);

        map.addIntersection(i01);
        map.addIntersection(i02);
        map.addIntersection(i03);
        map.addIntersection(i04);
        map.addIntersection(i05);
        map.addIntersection(i06);
        map.addIntersection(i07);
        map.addIntersection(i08);
        map.addIntersection(i09);
        map.addIntersection(i10);
        map.addIntersection(i11);
        map.addIntersection(i12);
        map.addIntersection(i13);
        map.addIntersection(i14);
        map.addIntersection(i15);
        map.addIntersection(i16);
        map.addIntersection(i17);

        return map;
    }

}
```

```
package londonsw.view.simulation;

import londonsw.model.simulation.MapGrid;
import londonsw.model.simulation.components.Component;
import londonsw.model.simulation.components.IMapGrid;

/**
 * Created by felix on 25/02/2016.
 */
public abstract class MapGridDecorator implements IMapGrid {
    protected MapGrid decoratedMapGrid;

    public MapGridDecorator(MapGrid decoratedMapGrid) {
        this.decoratedMapGrid = decoratedMapGrid;
    }

    @Override
    public Component[][] getGrid() {
        return this.decoratedMapGrid.getGrid();
    }

    @Override
    public int getWidth() {
        return this.decoratedMapGrid.getWidth();
    }

    @Override
    public int getHeight() {
        return this.decoratedMapGrid.getHeight();
    }

    @Override
    public boolean addComponent(Component component) {
        return this.decoratedMapGrid.addComponent(component);
    }
}
```

```
package londonsw.view.simulation;

import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.shape.Polygon;
import londonsw.controller.MapMakerController;
import londonsw.model.simulation.MapGrid;
import londonsw.model.simulation.components.*;
import londonsw.view.mapcreation.ComponentType;

import java.util.ArrayList;

/**
 * This is the class that handles drawing of the entire map. It cycles through t
he Model grid and draws
 * each map component on the screen.
 */
public class MapGridGUIDecorator extends MapGridDecorator {

    private ResizeFactor resizeFactor;

    /**
     * Creates an instance of this decorator to represent a single Map
     * @param decoratedMapGrid the MapGrid (underlying Map data structure) to re
present graphically
     */
    public MapGridGUIDecorator(MapGrid decoratedMapGrid) {
        super(decoratedMapGrid);
    }

    /**
     * Gets the resize factor for this decorator
     * @return the resize factor for this decorator
     */
    public ResizeFactor getResizeFactor() {
        return resizeFactor;
    }

    /**
     * Sets the resize factor for this decorator
     * @param resizeFactor the resize factor to set for this decorator
     */
    public void setResizeFactor(ResizeFactor resizeFactor) {
        this.resizeFactor = resizeFactor;
    }

    /**
     * Draws the components for this Map
     * @return a GridPane representing the map, where every cell in the GridPane
 corresponds to a coordinate in the Map
     * @throws Exception
     */
    public GridPane drawComponents() throws Exception {
        GridPane rootGP = new GridPane();
```

```
        StackPane roadPane;

        int roadCounter = 0;

        ArrayList<RoadGUIDecorator> roadArray = new ArrayList<>();

        for (int y = 0; y < this.getHeight(); y++) {
            for (int x = 0; x < this.getWidth(); x++) {

                Component current = this.getGrid()[y][x];

                if (current instanceof Road) { // draws a Road

                    RoadGUIDecorator roadGUIDecorator = new RoadGUIDecorator((Ro
ad) current);

                    roadGUIDecorator.setResizeFactor(this.getResizeFactor());

                    roadPane = roadGUIDecorator.drawRoad();

                    roadPane.getChildren().get(1).setOnMouseClicked(event ->
                            {

                                if (event.getTarget() instanceof LaneArrow) {
                                    LaneArrow laneArrow = (LaneArrow) event.getT
arget();

                                    for (RoadGUIDecorator rd : roadArray
                                            ) {


                                        if (rd.decoratedRoad.getId() == roadGUID
ecorator.decoratedRoad.getId()) {

                                            Node nGroup = rd.getPane().getChildr
en().get(1);

                                            Group gRoad = (Group) nGroup;

                                            Group g = (Group) gRoad.getChildren(
).get(laneArrow.lane.getRoadIndex());

                                            Line lineArrow = (Line) g.getChildre
n().get(0);
                                            Polygon arrow = (Polygon) g.getChild
ren().get(1);

                                            lineArrow.setStroke(lineArrow.getStr
oke() == Color.RED ? Color.WHITE : Color.RED);
                                            arrow.setFill(arrow.getFill() == Col
or.RED ? Color.WHITE : Color.RED);

                                        }

                                    }

                                    System.out.println("Lane ID: " + laneArrow.lan
e.getId() + " Lane State: " + laneArrow.lane.getState());
                                    laneArrow.lane.setState(laneArrow.lane.getSt
```

```
ate() == 0 ? 1 : 0);

                                    }
                                }
                    );

                    roadGUIDecorator.setCell(roadCounter);
                    roadGUIDecorator.setPane(roadPane);
                    roadGUIDecorator.setGridPaneCoordinates(new Coordinate(x, y)
);

                    roadArray.add(roadGUIDecorator);

                    roadCounter++;

                } else if (current instanceof Intersection) { // draws an Inters
ection
                    roadCounter = 0;

                    IntersectionDecorator intersectionDecorator = new Intersecti
onDecorator((Intersection) current);
                    intersectionDecorator.setResizeFactor(this.getResizeFactor()
);
                    roadPane = intersectionDecorator.drawIntersection();
                } else { // draws Grass
                    roadCounter = 0;

                    LayoutGUI grassGUI = new LayoutGUI();
                    grassGUI.setHeight(this.getHeight());
                    grassGUI.setWidth(this.getWidth());
                    grassGUI.setResizeFactor(this.getResizeFactor());
                    roadPane = grassGUI.drawGrass();
                }

                rootGP.add(roadPane, x, y);

            }
        }

        rootGP.setGridLinesVisible(true);

        return rootGP;
    }

    /**
     * Redraws the cell based on what is currently in the cell. Used by MapMaker
 mode.
     * @param x the x coordinate of the new component to be redrawn
     * @param y the y coordinate of the new component to be redrawn
     * @param gp the GridPane representing the Map
     * @return a StackPane representation of the newly redrawn grid cell
     */
    public StackPane redrawCell(int x, int y, GridPane gp) {
        Component component = this.getGrid()[y][x];
        StackPane sp = new StackPane();
        if(component instanceof Intersection) {
            IntersectionDecorator intersectionDecorator = new IntersectionDecora
tor((Intersection) component);
            intersectionDecorator.setResizeFactor(this.getResizeFactor());
```

```
            sp = intersectionDecorator.drawIntersection();
        }
        else if(component instanceof Road) {
            RoadGUIDecorator roadGUIDecorator = new RoadGUIDecorator((Road) comp
onent);
            roadGUIDecorator.setResizeFactor(this.getResizeFactor());
            sp = roadGUIDecorator.drawRoad();
        }
        else if(component == null) { // Grass
            LayoutGUI grassGUI = new LayoutGUI();
            grassGUI.setHeight(this.getHeight());
            grassGUI.setWidth(this.getWidth());
            grassGUI.setResizeFactor(this.getResizeFactor());
            sp = grassGUI.drawGrass();
        }
        gp.add(sp, x, y);
        return sp;
    }
}
```

```
package londonsw.view.simulation;

import londonsw.model.simulation.components.*;

import java.util.ArrayList;

/**
 * Created by felix on 25/02/2016.
 */
public abstract class RoadDecorator implements IRoad {

    protected Road decoratedRoad;

    public RoadDecorator(Road decoratedRoad) {
        this.decoratedRoad = decoratedRoad;
    }

    @Override
    public ArrayList<Lane> getLanes() {
        return this.decoratedRoad.getLanes();
    }

    @Override
    public void addLane(Lane lane) {
        this.decoratedRoad.addLane(lane);
    }

    @Override
    public Lane getLaneAtIndex(int index) {
        return this.decoratedRoad.getLaneAtIndex(index);
    }

    @Override
    public Coordinate getEndLocation() {
        return this.decoratedRoad.getEndLocation();
    }

    @Override
    public int getNumberLanes() {
        return decoratedRoad.getNumberLanes();
    }

    @Override
    public Intersection getIntersection() {
        return this.decoratedRoad.getIntersection();
    }

    @Override
    public void setIntersection(Intersection intersection) {
        this.decoratedRoad.setIntersection(intersection);
    }

    @Override
    public int getLength() {
        return this.decoratedRoad.getLength();
    }

    @Override
    public boolean runsVertically() {
```

```
        return this.decoratedRoad.runsVertically();
    }

    public boolean runsVertically(MapDirection mapDirection) {
        return this.decoratedRoad.runsVertically(mapDirection);
    }

}
```

```
package londonsw.view.simulation;

import javafx.scene.Group;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.shape.Line;
import londonsw.model.simulation.components.*;
import java.util.ArrayList;

/**
 * This class defines how roads are drawn and displayed in the view. Each Road i
nstance will
 * have exactly one of these decorators associated with it.
 */
public class RoadGUIDecorator extends RoadDecorator {

    private ResizeFactor resizeFactor;
    private Coordinate gridPaneCoordinates;
    private Pane paneRoad;
    private int Cell;

    /**
     * Creates a new instance of this decorator class for the given Road instanc
e
     * @param decoratedRoad the Road instance to associate this decorator with
     */
    public RoadGUIDecorator(Road decoratedRoad) {
        super(decoratedRoad);
    }

    /**
     * Gets the pane associated with this decorator
     * @return the pane for this decorator
     */
    public Pane getPane() {
        return paneRoad;
    }

    /**
     * Sets the pane for this decorator
     * @param paneRoad the pane to set for this decorator
     */
    public void setPane(Pane paneRoad) {
        this.paneRoad = paneRoad;
    }

    /**
     * Gets the cell for this road
     * @return the cell
     */
    public int getCell() {
        return Cell;
    }

    /**
     * Sets the cell for this road
     * @param cell the cell for this road
```

```
     */
    public void setCell(int cell) {
        Cell = cell;
    }

    /**
     * Gets the grid pane coordinate of this road
     * @return the coordinate for this road
     */
    public Coordinate getGridPaneCoordinates() {
        return gridPaneCoordinates;
    }

    /**
     * Sets the grid pane coordinate for this road
     * @param gridPaneCoordinates the coordinate for this road
     */
    public void setGridPaneCoordinates(Coordinate gridPaneCoordinates) {
        gridPaneCoordinates = gridPaneCoordinates;
    }

    /**
     * Gets the resize factor for this road
     * @return the resize factor for this road
     */
    public ResizeFactor getResizeFactor() {
        return resizeFactor;
    }

    /**
     * Sets the resize factor for this road
     * @param resizeFactor the resize factor to set for this road
     */
    public void setResizeFactor(ResizeFactor resizeFactor) {
        this.resizeFactor = resizeFactor;
    }

    /**
     * This method draws the road and returns the StackPane representation of th
is road. Each cell has a road background
     * image. Each cell also contains an arrow that displays the moving directio
n of that cell.
     * @return the StackPane representation of this road cell
     */
    public StackPane drawRoad() {

        String roadBackgroundPath = "RoadBackground.png";

        Image image = new Image(roadBackgroundPath);

        Image im = new Image(roadBackgroundPath, image.getHeight() * getResizeFa
ctor().getResizeX(), image.getWidth() * getResizeFactor().getResizeY(), false, f
alse);

        ImageView iv = new ImageView(im);

        StackPane stackPane = new StackPane();

        //draw amount of lines
```

```
        Group arrowLines = new Group();

        int numberLanes = this.getNumberLanes();

        ArrayList<Lane> lanes = this.getLanes();

        double division = im.getHeight();

        division = division / (numberLanes * 2);

        LaneArrow arrow;

        int j = 0;

        if (!this.runsVertically(lanes.get(0).getMovingDirection())) {
            for (int i = 0; i < numberLanes * 2; i++) {
                if (i % 2 == 0) {

                    Lane lane = lanes.get(j);

                    double lineStartX = 5;
                    double lineStartY = division * (i + 1);

                    double lineEndX = im.getWidth() - 10;
                    double lineEndY = division * (i + 1);

                    Line roadLine = new Line(lineStartX, lineStartY, lineEndX, l
ineEndY);

                    arrow = new LaneArrow(lane,roadLine,resizeFactor);

                    arrowLines.getChildren().addAll(arrow.getGroup());

                    j++;

                }
            }
        } else
            for (int i = 0; i < numberLanes * 2; i++) {

                if (i % 2 == 0) {

                    Lane lane = lanes.get(j);

                    double lineStartX = division * (i + 1);
                    double lineStartY = 5;
                    double lineEndX = division * (i + 1);
                    double lineEndY = im.getHeight() - 10;

                    Line roadLine = new Line(lineStartX, lineStartY, lineEndX, l
ineEndY);

                    arrow = new LaneArrow(lane,roadLine,resizeFactor);
                    arrowLines.getChildren().addAll(arrow.getGroup());

                    j++;

                }
            }
```

```
        stackPane.getChildren().add(iv);
        stackPane.getChildren().add(arrowLines);

        return stackPane;
    }
}
```

```
package londonsw.view.simulation;


import javafx.application.Platform;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.fxml.FXMLLoader;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
import javafx.stage.Stage;
import londonsw.controller.StartUpController;
import londonsw.controller.TrafficLightController;
import londonsw.controller.VehicleController;
import londonsw.model.simulation.Map;
import londonsw.model.simulation.Ticker;
import londonsw.model.simulation.components.ResizeFactor;
import javafx.scene.text.FontWeight;
import londonsw.model.simulation.components.Lane;
import londonsw.model.simulation.components.Road;
import londonsw.model.simulation.components.vehicles.Ambulance;
import londonsw.model.simulation.components.vehicles.Car;
import londonsw.model.simulation.components.vehicles.Vehicle;
import rx.Subscriber;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Optional;
import java.util.Random;


@SuppressWarnings("Duplicates")
public class SimulationScreen {

    private Map map;
    private int initCar = 0;
    private int flag = 0;
    private int systemState = 0;
    private int maxCarSize;
    Subscriber<Long> timeLabelSubscriber;
    Label carNumberSituation;
    Label timeStanding;

    public SimulationScreen(Map map) {
        this.map = map;
    }

    public void drawScreen(Stage primaryStage) throws Exception {
        // the entire screen building and logic will go here
        // http://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

        BorderPane borderPane = new BorderPane();
```

```
        borderPane.setStyle("-fx-background-color:papayawhip");

        //Title
        Label logo = new Label("LondonSW Traffic Simulation");
        logo.setFont(Font.font("System Bold Italic", FontWeight.EXTRA_BOLD, 20));
        logo.setAlignment(Pos.CENTER);
        logo.setTextAlignment(TextAlignment.CENTER);
        logo.setPadding(new Insets(10, 10, 10, 10));
        borderPane.setTop(logo);

        //Map
        StackPane mapStackPane = new StackPane();
        Map map = this.map;
        MapGridGUIDecorator mapGridGUIDecorator = new MapGridGUIDecorator(map.ge
tGrid());
        ResizeFactor rf = ResizeFactor.getSuggestedResizeFactor(map.getWidth(),
map.getHeight());
        mapGridGUIDecorator.setResizeFactor(rf);
        GridPane mapGridPane = mapGridGUIDecorator.drawComponents();
        mapStackPane.setPadding(new Insets(0, 0, 5, 5));

        mapStackPane.getChildren().add(mapGridPane);
        borderPane.setCenter(mapStackPane);

        //Start&Reset
        VBox simulationControl = new VBox();

        simulationControl.setPadding(new Insets(0,10,10,10));
        simulationControl.setSpacing(8);
        simulationControl.setAlignment(Pos.TOP_CENTER);

        carNumberSituation = new Label();
        carNumberSituation.setFont(Font.font("System Bold Italic",FontWeight.BOLD,13)
);
        carNumberSituation.setText("Number of cars: " + String.valueOf(initCar));
        simulationControl.getChildren().add(carNumberSituation);

        Label tickerSituation = new Label();
        tickerSituation.setFont(Font.font("System Bold Italic",FontWeight.BOLD,13));
        tickerSituation.setText("Ticker Interval: " + Ticker.getTickInterval() + " ms"
);
        simulationControl.getChildren().add(tickerSituation);

        Label trafficLightLabel = new Label();
        trafficLightLabel.setFont(Font.font("System Bold Italic",FontWeight.BOLD,13))
;
        trafficLightLabel.setText("Traffic Light Duration: " + TrafficLightController.ge
tInstance().getDurationLength()/1000 + " ticks");
        simulationControl.getChildren().add(trafficLightLabel);

        Label timeLabel = new Label();
        timeLabel.setFont(Font.font("System Bold Italic",FontWeight.BOLD,13));
        timeLabel.setText("Times ticked: 0");
        simulationControl.getChildren().add(timeLabel);

        timeStanding = new Label();
        timeStanding.setFont(Font.font("System Bold Italic",FontWeight.BOLD,13));
        timeStanding.setText("Vehicle Time Spent Standing: 0.0%");
        simulationControl.getChildren().add(timeStanding);
```

```
        Button startSimulation = new Button("Start");
        startSimulation.setFont(Font.font("System Bold Italic", FontWeight.BOLD, 13))
;
        startSimulation.setStyle("-fx-base:Gold");

        Button resetSimulation = new Button("Reset");
        resetSimulation.setFont(Font.font("System Bold Italic", FontWeight.BOLD, 13))
;
        resetSimulation.setStyle("-fx-base:Gold");

        Button backButton = new Button("Back");
        backButton.setFont(Font.font("System Bold Italic", FontWeight.BOLD, 16));
        backButton.setStyle("-fx-base:Gold");

        startSimulation.setPadding(new Insets(10, 10, 10, 10));
        startSimulation.setPrefSize(90, 30);
        simulationControl.getChildren().add(startSimulation);

        resetSimulation.setPadding(new Insets(10, 10, 10, 10));
        resetSimulation.setPrefSize(90, 30);
        simulationControl.getChildren().add(resetSimulation);
        resetSimulation.setDisable(true);

        Button ambulanceAddDelete = new Button("Add Ambulance");
        ambulanceAddDelete.setFont(Font.font("System Bold Italic", FontWeight.BOLD, 1
3));
        ambulanceAddDelete.setStyle("-fx-base:Gold");
        ambulanceAddDelete.setPrefSize(200, 30);
        ambulanceAddDelete.setDisable(true);
        simulationControl.getChildren().add(ambulanceAddDelete);

        Button trafficLightInterval = new Button("Set Traffic Light Duration");
        trafficLightInterval.setFont(Font.font("System Bold Italic", FontWeight.BOLD,
 13));
        trafficLightInterval.setStyle("-fx-base:Gold");
        trafficLightInterval.setPrefSize(200, 30);
        simulationControl.getChildren().add(trafficLightInterval);

        Button enableDisableLights = new Button("Disable Traffic Lights");
        enableDisableLights.setFont(Font.font("System Bold Italic", FontWeight.BOLD,
13));
        enableDisableLights.setStyle("-fx-base:Gold");
        enableDisableLights.setPrefSize(200, 30);
        simulationControl.getChildren().add(enableDisableLights);

        Label instructions = new Label("Click on the lane arrows to\nenable and disable lanes.");
        instructions.setFont(Font.font("System Bold Italic", FontWeight.BOLD, 13));
        instructions.setPadding(new Insets(5,0,0,0));
        simulationControl.getChildren().add(instructions);

        //carSlider
        VBox sliderControl = new VBox();
        sliderControl.setPadding(new Insets(10, 10, 5, 10));
        Pane carLabel = new Pane();
        Label carNumber = new Label("Car Number");
        carNumber.setFont(Font.font("System Bold Italic", FontWeight.BOLD, 13));
        carLabel.getChildren().add(carNumber);
        sliderControl.getChildren().add(carLabel);
```

```
        Pane carSlider = new Pane();
        carSlider.setPadding(new Insets(10,10,10,10));
        Slider slider = new Slider();
        int maxSize = determineMaxCarSize(map);
        slider.setPrefWidth(250);
        slider.setMax(maxSize);
        slider.setMin(1);
        slider.setDisable(true);
        slider.setShowTickMarks(true);
        if(maxSize <= 20) {
            slider.setMajorTickUnit(5);
            slider.setMinorTickCount(2);
            slider.setBlockIncrement(1);
        }
        else if(maxSize <= 50) {
            slider.setMajorTickUnit(10);
            slider.setMinorTickCount(4);
            slider.setBlockIncrement(1);
        }
        else if(maxSize <= 100) {
            slider.setMajorTickUnit(20);
            slider.setMinorTickCount(10);
            slider.setBlockIncrement(1);
        }
        else {
            slider.setMajorTickUnit(20);
            slider.setMinorTickCount(2);
            slider.setBlockIncrement(1);
        }

        carSlider.getChildren().add(slider);
        sliderControl.getChildren().add(carSlider);
        simulationControl.getChildren().add(sliderControl);
        simulationControl.getChildren().add(backButton);
        borderPane.setRight(simulationControl);
        borderPane.setPickOnBounds(false);


        /**
         * Back to ChooseSimulationMode Screen
         */
        backButton.setOnMouseClicked(click->{
            ArrayList<Vehicle> vehicles = VehicleController.getVehicleList();
            int size = vehicles.size();
            for(int i = 0; i < size; i++) {
                VehicleController.removeVehicle(0);
            }
            Ticker.end();
            StartUpController.getInstance().goToChooseModeScreen(primaryStage);
            primaryStage.centerOnScreen();
            primaryStage.setResizable(false);

        });

        /**
         * Allows the user to change the traffic light interval
         */
        trafficLightInterval.setOnMouseClicked(click -> {
```

```
            Dialog<Long> dialog = new Dialog<Long>();
            dialog.setTitle("Choose Traffic Light Duration");
            dialog.setHeaderText("Choose a duration (in time ticks) for\nthe traffic lights in the system.
");
            dialog.setGraphic(null);
            dialog.getDialogPane().getButtonTypes().addAll(ButtonType.OK, Button
Type.CANCEL);
            GridPane grid = new GridPane();
            grid.setHgap(10);
            grid.setVgap(10);
            grid.setPadding(new Insets(20, 80, 10, 10));
            grid.add(new Label("Duration:"), 0, 0);
            Spinner<Double> spinner = new Spinner<Double>(1, 20, TrafficLightCon
troller.getInstance().getDurationLength()/1000, 1);
            grid.add(spinner, 1, 0);
            dialog.getDialogPane().setContent(grid);
            Platform.runLater(() -> spinner.requestFocus());

            dialog.setResultConverter(dialogButton -> {
                if(dialogButton == ButtonType.OK) {
                    double value =  spinner.getValue();
                    value *= 1000;
                    return (long) value;
                }
                return null;
            });

            Optional<Long> result = dialog.showAndWait();
            result.ifPresent((aLong -> {
                TrafficLightController.getInstance().setDurationLength(aLong);
                TrafficLightController.getInstance().setTrafficLightDuration(aLo
ng);
                trafficLightLabel.setText("Traffic Light Duration: " + TrafficLightContr
oller.getInstance().getDurationLength()/1000 + " ticks");
            }));

        });

        /**
         * Functionality for enabling and disabling the traffic lights
         */
        enableDisableLights.setOnMouseClicked(click -> {
            if(TrafficLightController.getInstance().areLightsEnabled()) {
                // disable the lights!
                TrafficLightController.getInstance().disableLights(true);
                enableDisableLights.setText("Enable Traffic Lights");
                trafficLightLabel.setText("Traffic Light Duration: DISABLED");
            }
            else {
                // enable the lights!
                TrafficLightController.getInstance().disableLights(false);
                enableDisableLights.setText("Disable Traffic Lights");
                trafficLightLabel.setText("Traffic Light Duration: " + TrafficLightContr
oller.getInstance().getDurationLength()/1000 + " ticks");
            }
        });

        /**
         * Using a slider control the number of cars in the system
```

```
         */
        slider.valueProperty().addListener(new ChangeListener<Number>() {
            @Override
            public void changed(ObservableValue<? extends Number> observable, Nu
mber oldValue, Number newValue) {
                initCar = oldValue.intValue();
                int newCar = newValue.intValue() - oldValue.intValue();

                //increase carNumber
                if (newCar >= 0) {
                    for (int i = 0; i < newCar; i++) {
                        generateCar(map, mapGridGUIDecorator, mapStackPane);
                    }
                }
                //decrease carNumber
                else {
                    int toDelete = newCar * -1;
                    for(int i = 0; i < toDelete; i++) {
                        ArrayList<Vehicle> vehicles = VehicleController.getVehic
leList();
                        if(vehicles.size() == 0) return;
                        Random rand = new Random();
                        int min = 0;
                        int max = vehicles.size();
                        int randomIndex = rand.nextInt((max - min)) + min;
                        while(vehicles.get(randomIndex) instanceof Ambulance) {
                            randomIndex = rand.nextInt((max - min)) + min;
                        }
                        VehicleController.removeVehicle(randomIndex);
                    }
                }
                ArrayList<Vehicle> vehicles = VehicleController.getVehicleList()
;
                carNumberSituation.setText("Number of cars: " + vehicles.size());
            }
        });

        /**
         * "Add/Delete Ambulance" Button click control
         * the first click adds an ambulance in the system, the next click will
delete the ambulance, next add...
         */
        ambulanceAddDelete.setOnMouseClicked(click -> {
            if (flag == 0) {
                generateAmbulance(map, mapGridGUIDecorator, mapStackPane);
                ambulanceAddDelete.setText("Delete Ambulance");
                flag = 1;
            } else {
                flag = 0;
                ArrayList<Vehicle> vehicles = VehicleController.getVehicleList()
;
                for (int i = 0; i < vehicles.size(); i++) {
                    if (vehicles.get(i).getVehiclePriority() == 5) {
                        VehicleController.removeVehicle(i);
                    }
                }
                ambulanceAddDelete.setText("Add Ambulance");
            }
            carNumberSituation.setText("Number of cars: " + VehicleController.getVeh
```

```
icleList().size());
        });

        Scene scene = new Scene(borderPane);

        primaryStage.setScene(scene);
        primaryStage.centerOnScreen();

        /**
         * Starts the simulation
         */
        startSimulation.setOnMouseClicked(click->{
            systemState = 1;
            ambulanceAddDelete.setDisable(false);
            slider.setDisable(false);
            initCar = (int) slider.getValue();
            carNumberSituation.setText("Number of cars: " + initCar);

            for (int i = 0; i < initCar; i++) {
                generateCar(map, mapGridGUIDecorator, mapStackPane);
            }

            startSimulation.setDisable(true);
            resetSimulation.setDisable(false);

            Platform.runLater(() -> slider.requestFocus());
            startTimeLabelTicker(timeLabel);
        });


        /**
         * To stop simulation
         */
        resetSimulation.setOnMouseClicked(click -> {
            systemState = 0;
            slider.setDisable(true);
            startSimulation.setDisable(false);
            resetSimulation.setDisable(true);
            ambulanceAddDelete.setText("Add Ambulance");
            ambulanceAddDelete.setDisable(true);

            //reset lanes to enabled //TODO

            ArrayList<Vehicle> vehicles = VehicleController.getVehicleList();
            int size = vehicles.size();
            for(int i = 0; i < size; i++) {
                VehicleController.removeVehicle(0);
            }
            carNumberSituation.setText("Number of cars: " + VehicleController.getVeh
icleList().size());
            Platform.runLater(() -> startSimulation.requestFocus());
            endTimeLabelTicker();
            timeLabel.setText("Times ticked: 0");
            timeStanding.setText("Vehicle Time Spent Standing: 0.0%");
        });
    }

    /**
     * Creates a subscriber that listens to the ticker to update the "times tick
```

```
ed" label
     * @param timeLabel the label to update on every tick
     */
    private void startTimeLabelTicker(Label timeLabel) {
        timeLabelSubscriber = new Subscriber<Long>() {
            int timesTicked = 0;
            @Override
            public void onCompleted() {

            }

            @Override
            public void onError(Throwable throwable) {

            }

            @Override
            public void onNext(Long aLong) {
                timeLabel.setText("Times ticked: " + timesTicked);
                timesTicked++;
                carNumberSituation.setText("Number of cars: " + VehicleController.ge
tVehicleList().size());
                timeStanding.setText("Vehicle Time Spent Standing: " + getPercentageStan
ding() + "%");
            }
        };
        Ticker.subscribe(timeLabelSubscriber);
    }

    /**
     * Stops the time ticker label from listening to the ticker
     */
    private void endTimeLabelTicker() {
        timeLabelSubscriber.unsubscribe();
    }

    private double getPercentageStanding() {
        int timeSpentStanding = VehicleController.getTotalTimeSpentStanding();
        int totalTimesTicked = VehicleController.getTotalTimesTicked();
        if(totalTimesTicked == 0) {
            return 0.0;
        }
        double ans = (double) timeSpentStanding / totalTimesTicked * 100;
        return Math.round(ans * 100.0) / 100.0;
    }

    /**
     * Determines the maximum number of cars that should go in the system
     * @param map the map the simulation is happening on
     * @return an upper bound on the number of cars that should be in the system
     */
    public int determineMaxCarSize(Map map){
        int numberSlots = 0;
        ArrayList<Road> roads = map.getRoads();
        for(Road i:roads){
            ArrayList<Lane> lanes = i.getLanes();
            for(Lane l:lanes){
                numberSlots += l.getLength();
            }
```

```
        }
        maxCarSize = (int)(0.6 * numberSlots);
        return maxCarSize;
    }

    /**
     * Car generator
     * @param map
     * @param mapGridGUIDecorator
     * @param sp
     * @return
     */
    public Car generateCar(Map map, MapGridGUIDecorator mapGridGUIDecorator, Sta
ckPane sp) {
        Lane L1 = map.getRandomLane();
        if (L1 != null) {
            for (int a = 0; a < map.getRoads().size(); a++) {
                for (int b = 0; b < map.getRoads().get(a).getNumberLanes(); b++)
 {
                    Lane L2 = map.getRoads().get(a).getLanes().get(b);
                    for (int i = 0; i < L1.getLength(); i++) {
                        if (L1.isCellEmpty(i)) {
                            Car C1 = new Car(i, L1);
                            //C1.setVehicleBehavior(VehicleBehavior.AGGRESSIVE);
                            VehicleGUIDecorator vehicleGUIDecorator = new Vehicl
eGUIDecorator(C1);
                            vehicleGUIDecorator.setResizeFactor(mapGridGUIDecora
tor.getResizeFactor());
                            vehicleGUIDecorator.drawCar();
                            Pane carPane = new Pane();

                            carPane.setPickOnBounds(false); //allows me to click
 intersections

                            carPane.getChildren().add(vehicleGUIDecorator.getRec
tangle());
                            sp.getChildren().add(carPane);
                            vehicleGUIDecorator.setPane(carPane);
                            vehicleGUIDecorator.setVehicleState(1);
                            return C1;

                        }
                    }
                }
            }
        }
        return null;
    }

    /**
     * Ambulance generator
     * @param map
     * @param mapGridGUIDecorator
     * @param sp
     * @return
     */
    public Ambulance generateAmbulance(Map map, MapGridGUIDecorator mapGridGUIDe
corator, StackPane sp) {
        Lane AmbLane = map.getRandomLane();
```

```
        if (AmbLane != null && (!AmbLane.isFull())) {
            for (int x = 0; x < map.getRoads().size(); x++) {
                for (int y = 0; y < map.getRoads().get(x).getNumberLanes(); y++)
 {
                    AmbLane = map.getRandomLane();
                    for (int z = 0; z < AmbLane.getLength(); z++) {
                        if (AmbLane.isCellEmpty(z)) {
                            Ambulance A = new Ambulance(z, AmbLane);
                            VehicleGUIDecorator ambulanceGUIDecorator = new Vehi
cleGUIDecorator(A);
                            ambulanceGUIDecorator.setResizeFactor(mapGridGUIDeco
rator.getResizeFactor());
                            ambulanceGUIDecorator.setColor(Color.RED);
                            ambulanceGUIDecorator.drawCar();
                            Pane alPane = new Pane();
                            alPane.setPickOnBounds(false);
                            alPane.getChildren().add(ambulanceGUIDecorator.getRe
ctangle());
                            ambulanceGUIDecorator.setPane(alPane);
                            sp.getChildren().add(alPane);
                            ambulanceGUIDecorator.setVehicleState(1);
                            return A;

                        }
                    }

                }

            }
        }
        return null;
    }
}
```

```
package londonsw.view.simulation;

import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import londonsw.controller.TrafficLightController;
import londonsw.model.simulation.components.LightColour;
import londonsw.model.simulation.components.TrafficLight;

import java.io.Serializable;

/**
 * Traffic Light GUI Logic is implemented here. One instance of this class is as
sociated with exactly one instance
 * of a TrafficLight.
 */
public class TrafficLightDecorator implements Serializable {

    private static final long serialVersionUID = 8123065437897754089L;
    private TrafficLight thisLight;
    private Circle circle;

    /**
     * Creates a new instance of the TrafficLightDecorator for the GUI.
     * @param thisLight the instance of a TrafficLight that this decorator is fo
r
     */
    public TrafficLightDecorator(TrafficLight thisLight) {
        this.thisLight = thisLight;
        circle = new Circle();
        TrafficLightController.getInstance().addTrafficLightAndDecoratorPair(thi
sLight, this);
    }

    /**
     * This gets called by the controller to set the color of the circle of the
traffic light gui
     * @param colour
     */
    public void setGUIColour(LightColour colour) {
        switch (colour) {
            case RED:
                circle.setFill(Color.RED);
                break;

            case GREEN:
                circle.setFill(Color.GREEN);
                break;
        }
    }

    /**
     * Returns the circle associated with this decorator
     * @return circle for this traffic light
     */
    public Circle getCircle() {
        return circle;
    }

    /**
```

```
     * Hides the circle representing the traffic light from the GUI. This is cal
led when enabling and disabling
     * traffic lights.
     * @param hide true will hide the circle, false will display the circle
     */
    public void hideCircle(boolean hide) {
        if(hide)
            circle.setVisible(false);
        else
            circle.setVisible(true);
    }

    /**
     * Draws the circle for the GUI
     * @param x the x-position in its pane
     * @param y the y-position in its pane
     * @param r the radius of the circle
     * @return a new Circle with those properties
     */
    public Circle drawLight(double x, double y, double r){
        circle.setCenterX(x);
        circle.setCenterY(y);
        circle.setRadius(r);
        circle.setFill(thisLight.getState() == LightColour.RED? Color.RED : Colo
r.GREEN);
        return circle;
    }

}
```

```
package londonsw.view.simulation;

import londonsw.model.simulation.components.Coordinate;
import londonsw.model.simulation.components.Lane;
import londonsw.model.simulation.components.TrafficLight;
import londonsw.model.simulation.components.VehicleBehavior;
import londonsw.model.simulation.components.vehicles.IVehicle;
import londonsw.model.simulation.components.vehicles.Vehicle;

import java.util.ArrayList;

/**
 * Created by felix on 26/02/2016.
 */
public abstract class VehicleDecorator implements IVehicle {

    protected Vehicle decoratedVehicle;

    public VehicleDecorator(Vehicle decoratedVehicle) {
        this.decoratedVehicle = decoratedVehicle;
    }

    @Override
    public Lane getPreviousLane() {
        return decoratedVehicle.getPreviousLane();
    }

    @Override
    public void setPreviousLane(Lane previousLane) {
        decoratedVehicle.setPreviousLane(previousLane);
    }

    public void setPreviousCoordinate(Coordinate coord) {
        decoratedVehicle.setPreviousCoordinate(coord);
    }

    @Override
    public void setCurrentCoordinate(Coordinate currentCoordinate) {
        decoratedVehicle.setCurrentCoordinate(currentCoordinate);
    }

    @Override
    public int getVehicleLength() {
        return decoratedVehicle.getVehicleLength();
    }

    @Override
    public double getVehicleSpeed() {
        return decoratedVehicle.getVehicleSpeed();
    }

    @Override
    public int getVehiclePriority() {
        return decoratedVehicle.getVehiclePriority();
    }

    @Override
    public Lane getCurrentLane() {
        return decoratedVehicle.getCurrentLane();
```

```
    }

    @Override
    public int getCurrentCell() {
        return decoratedVehicle.getCurrentCell();
    }

    @Override
    public int getVehicleState() {
        return decoratedVehicle.getVehicleState();
    }

    @Override
    public VehicleBehavior getVehicleBehavior() {
        return decoratedVehicle.getVehicleBehavior();
    }

    @Override
    public Coordinate getCurrentCoordinate() {
        return decoratedVehicle.getCurrentCoordinate();
    }

    public Coordinate getPreviousCoordinate() {
        return decoratedVehicle.getPreviousCoordinate();
    }


    public Coordinate getStoredCurrentCoordinate() {
        return decoratedVehicle.getStoredCurrentCoordinate();
    }

    @Override
    public void setVehicleLength(int vehicleLength) {
        decoratedVehicle.setVehicleLength(vehicleLength);
    }

    @Override
    public void setVehicleSpeed(double vehicleSpeed) {
        decoratedVehicle.setVehicleSpeed(vehicleSpeed);
    }

    @Override
    public void setVehiclePriority(int vehiclePriority) {
        decoratedVehicle.setVehiclePriority(vehiclePriority);
    }

    @Override
    public void setCurrentLane(Lane currentLane) throws Exception {
        decoratedVehicle.setCurrentLane(currentLane);
    }

    @Override
    public void setCurrentCell(int curCell, Lane currentLane) throws Exception {
        decoratedVehicle.setCurrentCell(curCell,currentLane);
    }

    @Override
    public void setVehicleState(int vehicleState) {
        decoratedVehicle.setVehicleState(vehicleState);
```

```
    }

    @Override
    public void setVehicleBehavior(VehicleBehavior vehicleBehavior) {
        decoratedVehicle.setVehicleBehavior(vehicleBehavior);
    }

    @Override
    public int moveVehicle(int step) throws Exception {
        return decoratedVehicle.moveVehicle(step);
    }

    @Override
    public void readTrafficLight() throws Exception {
        decoratedVehicle.readTrafficLight();
    }

    @Override
    public ArrayList<Lane> getLaneOptions() throws Exception {
        return decoratedVehicle.getLaneOptions();
    }

    @Override
    public Lane chooseLane () throws Exception{
        return decoratedVehicle.chooseLane();
    }

    @Override
    public int vehicleTurn(Lane l) throws Exception {
        return decoratedVehicle.vehicleTurn(l);
    }

    @Override
    public int getVehiclePriorityToTurn(){
        return decoratedVehicle.getVehiclePriorityToTurn();
    }

    @Override
    public void setVehiclePriorityToTurn(int vehiclePriorityToTurn){
         decoratedVehicle.setVehiclePriorityToTurn( vehiclePriorityToTurn);
    }
    @Override
    public TrafficLight getVehicleTrafficLight(){
        return decoratedVehicle.getVehicleTrafficLight();
    }
    @Override
    public void setVehicleTrafficLight(TrafficLight vehicleTrafficLight){
        decoratedVehicle.setVehicleTrafficLight(vehicleTrafficLight);
    }


}
```

```
package londonsw.view.simulation;

import javafx.animation.*;
import javafx.beans.property.DoubleProperty;
import javafx.scene.Group;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.util.Duration;
import londonsw.controller.VehicleController;
import londonsw.model.simulation.Ticker;
import londonsw.model.simulation.components.Coordinate;
import londonsw.model.simulation.components.MapDirection;
import londonsw.model.simulation.components.ResizeFactor;
import londonsw.model.simulation.components.vehicles.Vehicle;

/**
 * This class is to draw vehicles and display them in the GUI. Each vehicle will
 have exactly one VehicleGUIDecorator
 * associated with it.
 */
public class VehicleGUIDecorator extends VehicleDecorator {

    private double imageDimension;
    private Rectangle rectangle;
    private ResizeFactor resizeFactor;
    private Color color;
    private double verticalStartFudgeFactor;
    private double horizontalStartFudgeFactor;
    private Pane pane;

    /**
     * Creates a new instance of this decorator class
     * @param decoratedVehicle the Vehicle to be associated with
     */
    public VehicleGUIDecorator(Vehicle decoratedVehicle) {
        super(decoratedVehicle);
        VehicleController.addVehicleAndDecoratorPair(decoratedVehicle,this);
        imageDimension = 100.0;
        verticalStartFudgeFactor = 0;
        horizontalStartFudgeFactor = 0;
    }

    /**
     * Sets the resize factor for this vehicle to display properly
     * @param resizeFactor the resize factor for the simulation
     */
    public void setResizeFactor(ResizeFactor resizeFactor) {
        this.resizeFactor = resizeFactor;
    }

    /**
     * Gets the resize factor for the simulation
     * @return
     */
    public ResizeFactor getResizeFactor() {
        return resizeFactor;
```

```
    }

    /**
     * Gets the rectangle of the vehicle
     * @return the rectangle representing the vehicle
     */
    public Rectangle getRectangle() {
        return rectangle;
    }

    /**
     * Sets the rectangle of the vehicle
     * @param rectangle the rectangle to represent the vehicle
     */
    public void setRectangle(Rectangle rectangle) {
        this.rectangle = rectangle;
    }

    /**
     * Gets the pane of the vehicle
     * @return the pane the vehicle resides in
     */
    public Pane getPane() {
        return pane;
    }

    /**
     * Sets the pane of the vehicle
     * @param pane the pane to set
     */
    public void setPane(Pane pane) {
        this.pane = pane;
    }

    /**
     * Gets the color of the vehicle
     * @return the Color of this vehicle
     */
    public Color getColor() {
        return color;
    }

    /**
     * Sets the color of the vehicle
     * @param color the color of the vehicle
     */
    public void setColor(Color color) {
        this.color = color;
    }

    /**
     * Gets the vehicle associated with this decorator class
     * @return the vehicle this decorator class is for
     */
    public Vehicle getVehicle() {
        return decoratedVehicle;
    }
```

```
    /**
     * Draws the vehicle to display in the simulation. Each vehicle would have t
his method called on it.
     */
    public void drawCar() {
        double cellDimension = imageDimension;

        int numberLanes = this.getCurrentLane().getRoad().getNumberLanes();
        double carDimensionX = cellDimension;
        double carDimensionY = cellDimension;
        double angle = 0.0;

        // determine the start location of the vehicle in the map
        Coordinate coordinate = this.getCurrentCoordinate();
        double[] start = coordinateToPixels(coordinate,this.getCurrentLane().get
MovingDirection());
        double startPointX = start[0];
        double startPointY = start[1];

        // determine the size of the vehicle based on the size of the map
        if(this.getCurrentLane().getRoad().runsVertically())
        {
            carDimensionX = cellDimension/numberLanes - 50*resizeFactor.getResiz
eX();
            carDimensionY -= 15;
            if(resizeFactor.getResizeX() <= 0.25) {
                carDimensionX = cellDimension/numberLanes - 80*resizeFactor.getR
esizeX();
            }
            verticalStartFudgeFactor = -carDimensionX*resizeFactor.getResizeX();
        }
        else {
            carDimensionY = cellDimension/numberLanes - 50*resizeFactor.getResiz
eX();
            carDimensionX -= 15;
            if(resizeFactor.getResizeX() <= 0.25) {
                carDimensionY = cellDimension/numberLanes - 80*resizeFactor.getR
esizeX();
            }
            horizontalStartFudgeFactor = -carDimensionY*resizeFactor.getResizeX(
);
        }

        Rectangle r = new Rectangle(
                startPointX,
                startPointY,
                carDimensionX * this.getResizeFactor().getResizeX(),
                carDimensionY * this.getResizeFactor().getResizeY());

        /**
         * Simulating Ambulance using fill Transitions
         *
         */
        FillTransition ft = new FillTransition();
        ft.setShape(r);
        ft.setDuration(Duration.millis(500));
        if(this.getVehiclePriority()==5) {
            ft.setFromValue(this.getColor());
            ft.setToValue(Color.BLUE);
```

```
            ft.setCycleCount(Timeline.INDEFINITE);
            ft.setAutoReverse(true);
            ft.setInterpolator(Interpolator.LINEAR);
            ft.play();
        }

        else
            r.setFill(Color.YELLOW);

        r.setRotate(angle);

        this.setRectangle(r);
    }

    /**
     * Moves the vehicle in the GUI based on the state of the vehicle. A vehicle
 is either stopped, moving through
     * an intersection, moving straight down a road, or left the simulation.
     * @param step how much to move the vehicle by
     * @param state the state of the vehicle
     */
    public void moveVehicleGUI(int step, int state) {

        final Timeline timeline = new Timeline();
        //timeline.setAutoReverse(true);

        /* move the car according to moving direction, below */
        if(state == 0) { // car must stop because red light, or something in the
 way
            timeline.stop();
        }
        else if(state == 2)  //car is at intersection
        {
            TranslateTransition tt = new TranslateTransition(Duration.millis(Tic
ker.getTickInterval()), this.getRectangle());
            MapDirection fromDirection = this.getPreviousLane().getMovingDirecti
on();
            MapDirection toDirection = this.getCurrentLane().getMovingDirection(
);

            Coordinate coordinate = this.getPreviousCoordinate();

            double fromXPixels = rectangle.xProperty().doubleValue();
            double fromYPixels = rectangle.yProperty().doubleValue();

            // determine new position of vehicle
            Coordinate fromTranslation = directionToTranslation(fromDirection);
            Coordinate toTranslation = directionToTranslation(toDirection);
            Coordinate overallTranslation = Coordinate.add(fromTranslation,toTra
nslation);
            Coordinate newPosition = Coordinate.add(overallTranslation,coordinat
e);

            // work out rotation
            int rotation = getRotationFromDirectionChange(fromDirection,toDirect
ion);

            // determine pixel locations for the actual animation
            double[] toPixels = coordinateToPixels(newPosition,toDirection);
```

```
            double toXPixels = toPixels[0];
            double toYPixels = toPixels[1];

            // set the movement translation
            tt.setToX(toXPixels - fromXPixels);
            tt.setToY(toYPixels - fromYPixels);

            // move and rotate the car
            RotateTransition rt = new RotateTransition();
            rt.setNode(tt.getNode());
            rt.setByAngle(rotation);
            rt.setDuration(Duration.millis(Ticker.getTickInterval()));
            rt.play();
            tt.play();
            this.setVehicleState(1);

        }
        else if(state == 3) {
            //Car deleted state
            this.getPane().getChildren().remove(this.getRectangle());
        }
        else // car driving straight down road
        {
            TranslateTransition tt = new TranslateTransition(Duration.millis(Tic
ker.getTickInterval()), this.getRectangle());

            Coordinate coordinate = this.getPreviousCoordinate();
            MapDirection toDirection = this.getCurrentLane().getMovingDirection(
);

            double fromXPixels = rectangle.xProperty().doubleValue();
            double fromYPixels = rectangle.yProperty().doubleValue();

            // determine new location for car to move to
            Coordinate toTranslation = directionToTranslation(toDirection);
            Coordinate newPosition = Coordinate.add(toTranslation,coordinate);

            // determine pixel locations for the actual animation
            double[] toPixels = coordinateToPixels(newPosition,toDirection);
            double toXPixels = toPixels[0];
            double toYPixels = toPixels[1];

            // set the translation animation based on moving direction
            if(toDirection == MapDirection.NORTH || toDirection == MapDirection.
SOUTH)
                tt.setToY(toYPixels - fromYPixels);
            else if(toDirection == MapDirection.EAST || toDirection == MapDirect
ion.WEST)
                tt.setToX(toXPixels - fromXPixels);

            // play the animation
            tt.play();
        }
    }

    /**
     * Determines the pixel location for a vehicle if it occupies Coordinate c a
nd traveling in the MapDirection direction.
     * Each grid cell location is determined by the image dimension, resized by
```

```
the resize factor, and the lane the
     * vehicle should occupy.
     * @param c the Coordinate where the vehicle will be
     * @param direction the MapDirection that coordinate is moving, to determine
 the proper lane for the vehicle
     * @return an array of length 2, where index 0 is the x-coordinate and index
 1 is the y-coordinate. This is so you can get
     * both x- and y-locations at once, without calling the function twice.
     */
    private double[] coordinateToPixels(Coordinate c, MapDirection direction) {
        double[] pixels = new double[2];
        int x = c.getX();
        int y = c.getY();
        if(direction == MapDirection.NORTH) {
            pixels[0] = x * imageDimension * resizeFactor.getResizeX() + (.1 * i
mageDimension * resizeFactor.getResizeX() + horizontalStartFudgeFactor);
            pixels[1] = y * imageDimension * resizeFactor.getResizeX() - horizon
talStartFudgeFactor;
        }
        else if(direction == MapDirection.EAST) {
            pixels[0] = x * imageDimension * resizeFactor.getResizeX() + (.1 * i
mageDimension * resizeFactor.getResizeX() - verticalStartFudgeFactor);
            pixels[1] = y * imageDimension * resizeFactor.getResizeX() + (.1 * i
mageDimension * resizeFactor.getResizeX() + verticalStartFudgeFactor);
        }
        else if(direction == MapDirection.WEST) {
            pixels[0] = x * imageDimension * resizeFactor.getResizeX() - vertica
lStartFudgeFactor;
            pixels[1] = y * imageDimension * resizeFactor.getResizeX() + (.6 * i
mageDimension * resizeFactor.getResizeX() + verticalStartFudgeFactor);
        }
        else if(direction == MapDirection.SOUTH) {
            pixels[0] = x * imageDimension * resizeFactor.getResizeX() + (.6 * i
mageDimension * resizeFactor.getResizeX()) + horizontalStartFudgeFactor;
            pixels[1] = y * imageDimension * resizeFactor.getResizeX() + (.1 * i
mageDimension * resizeFactor.getResizeX()) - horizontalStartFudgeFactor;
        }

        return pixels;
    }

    /**
     * Determines the vehicle rotation required based on where it is coming from
 and where it is going to. Possible
     * rotations are -90 for turning left, 90 for turning right, or 0 for drivin
g straight.
     * @param fromDirection the direction the vehicle is coming from
     * @param toDirection the direction the vehicle is moving to
     * @return integer representing how much to rotate the vehicle by (90, -90,
or 0)
     */
    private int getRotationFromDirectionChange(MapDirection fromDirection, MapDi
rection toDirection) {
        if(
                (fromDirection == MapDirection.SOUTH && toDirection == MapDirect
ion.EAST) ||
                (fromDirection == MapDirection.NORTH && toDirection == MapDirect
ion.WEST) ||
                (fromDirection == MapDirection.EAST && toDirection == MapDirecti
```

```
on.NORTH) ||
                (fromDirection == MapDirection.WEST && toDirection == MapDirecti
on.SOUTH)) {
            return -90;
        }
        else if(
                (fromDirection == MapDirection.EAST && toDirection == MapDirecti
on.SOUTH) ||
                (fromDirection == MapDirection.SOUTH && toDirection == MapDirect
ion.WEST) ||
                (fromDirection == MapDirection.WEST && toDirection == MapDirecti
on.NORTH) ||
                (fromDirection == MapDirection.NORTH && toDirection == MapDirect
ion.EAST)) {
            return 90;
        }
        else {
            return 0;
        }
    }

    /**
     * Translates a moving direction into a Coordinate translation. For instance
, moving north means moving 0 squares in the x
     * direction and -1 square in the y direction. This is used for determining
overall movement of vehicles.
     * @param fromDirection the direction the vehicle is moving from or to
     * @return a Coordinate representing the 2-d translation required for that m
ap direction
     */
    private Coordinate directionToTranslation(MapDirection fromDirection) {
        switch (fromDirection) {
            case NORTH:
                return new Coordinate(0,-1);
            case SOUTH:
                return new Coordinate(0,1);
            case EAST:
                return new Coordinate(1,0);
            case WEST:
                return new Coordinate(-1,0);
        }

        return null;
    }
}
```