Description:

Java Swing application that uses event handlers, listeners and incorporates Java's concurrency functionality and the use of threads. This application displays time, traffic signals and other information for traffic analysts. The GUI design includes viewing ports/panels to display the following components of the simulation:

1. Current time stamps in 1 second intervals
2. Real-time Traffic light display for three major intersections
3. X, Y positions and speed of up to 3 cars as they traverse each of the 3 intersections

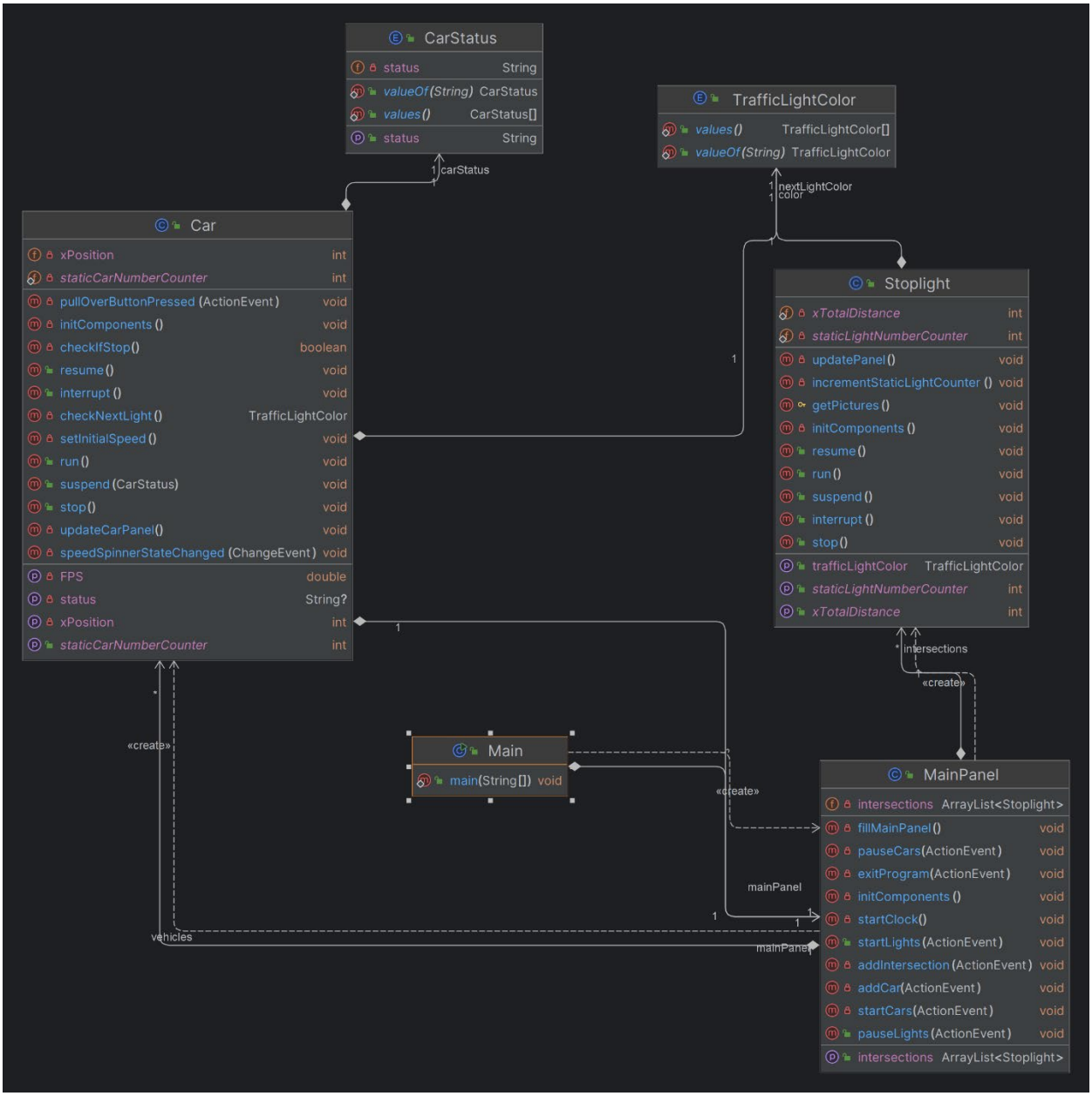
Details of the Simulation:

1. The components listed above run in separate threads.
2. Ability to loop through the simulation with button(s) providing the ability to start, pause, stop and continue the simulation.
4. Assume a straight distance between each traffic light of 1000 meters.
5. Assume Y = 0 for your X, Y positions.
6. Provides the ability to add more cars and intersections to the simulation through GUI.

Operating Instructions:

1. Open the program source folder.
2. Run the "Main.java". Make sure there is no error in any file (error could occur due to library importing issue).

UML DIAGRAM



TEST PLAN

CMSC 335 - Project 3 Test Cases				
What aspect Is Tested	**Input**	**Expected Output**	**Actual Output**	**Pass/Fail**
Exit	Press Exit Button	Jdialog Popup, Graceful Shutdown	expected output	Pass
Start Cars	press Start Cars Button	All Cars begin traveling at respective speeds. Progress bars update as cars travel cars status display Driving@[currentSpeed]	expected output	Pass
Pause Cars	press Pause Cars Button	All Cars stop traveling progress bars stop increasing	expected output	Pass
Resume Cars	press Resume Cars Button	All cars resume similar to pressing Start Cars	expected output	Pass
Add Car	press Add Car Button	car is added with random speed between 35 and 100 mph car starts driving	expected output	Pass
Change Speed	Change Speed Spinner	Speed spinner value changed. Status updated to Driving@[newSpeed] progressBar update speed reflects new speed as it progresses	expected output	Pass
pullOver	press Pull Over Button	cars location is paused progress bar paused status updated to "pulled over"	expected output	Pass
Start Lights	press Start Lights Button	All lights begin to change colors on expected intervals	expected output	Pass
Pause Lights	press Pause Lights Button	All lights pause updating. Light status updated to "MAINTENANCE"	expected output	Pass
Resume Lights	press Resume Lights Button	All Lights resume operation	expected output	Pass
Add Light	press Add Lights Button	Distance in cars xPosition/Total distance is updated cars ProgressBar positions are adjusted to reflect additional	expected output	Pass

		road distance Light is added and starts running		
Concurrent Operation	Add Car Button Add Light Button Start Lights Start Cars Car Pulled Over	Car Added Light Added Lights started Cars Started With cars and lights both active, lights continuously change. Cars continuously progress. Cars Stop at yellow and red Lights And their status is updated to "At Light". When their current light changes to green, each car resumes progress. Pulled over car pauses as intended while other cars remain traveling	expected output	Pass

LESSONS LEARNED

GUI Thread Hierarchy:

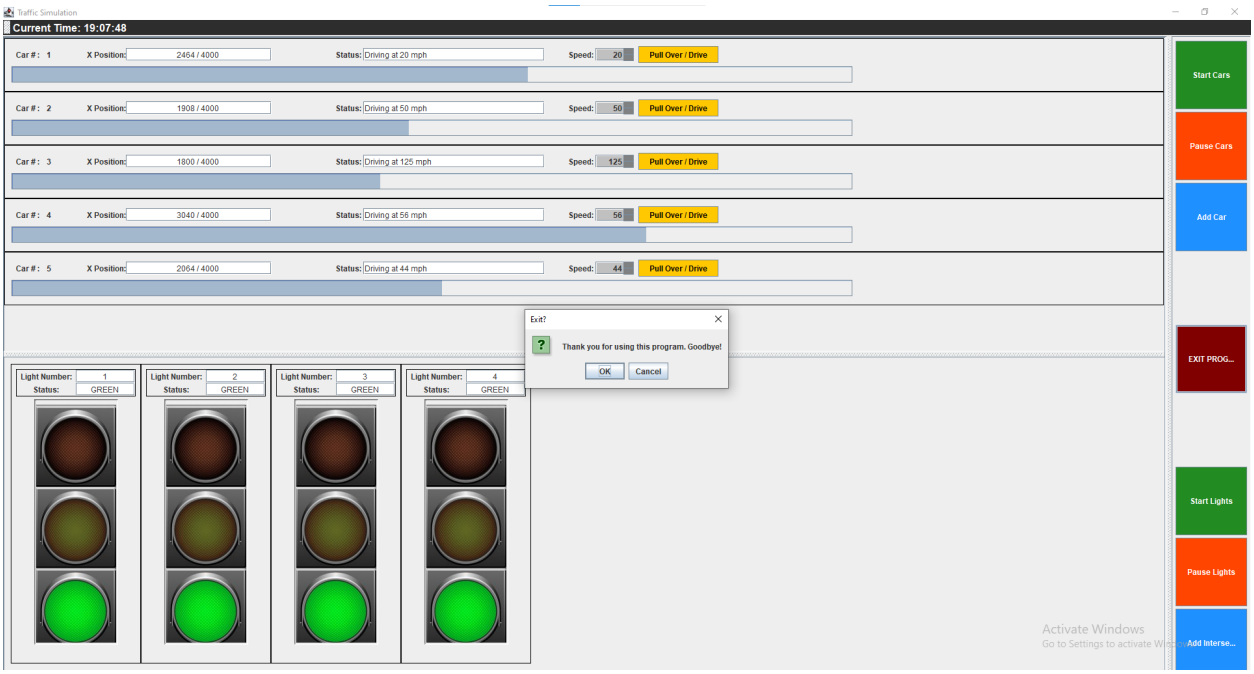
When dealing with concurrent threads, they must be passed by reference. Any class that is executing concurrently must have synchronized getters and setters if there is a chance that other classes will try to access a variable at the same time. By synchronizing methods, I was able to prevent deadlock and thread false updates of values within concurrently running objects.

Creating Runnable JPanel components:

Because each Swing component primarily executes code on the Event Dispatcher Thread, to achieve concurrency, I created each threaded runnable Panel as an independent class. This way, every Stoplight can be addressed directly, and be allowed to invoke its own individual Swing Worker and be paused and resumed independently. Rather than linking many java class files to a single GUI Panel, each Stoplight and Car operates within its own Panel. This also allows for mobility of code; I could implement the Car panel or the Stoplight panel anywhere a JPanel can be created, and it will run perfectly fine. If I wanted to connect them to a racing game, I could add them on the bottom of another application very easily, without having to recode any GUI.

RELEVANT PROJECT IMAGES


Traffic Simulation
Current Time: 19:07:30
Car # : 1 X Position: 1974 / 4000 Status: Driving at 20 mph Speed: 20 Pull Over / Drive
Car # : 2 X Position: 648 / 4000 Status: Driving at 50 mph Speed: 50 Pull Over / Drive
Car # : 3 X Position: 2925 / 4000 Status: At red light Speed: 125 Pull Over / Drive
Car # : 4 X Position: 1620 / 4000 Status: Driving at 56 mph Speed: 56 Pull Over / Drive
Car # : 5 X Position: 944 / 4000 Status: Driving at 44 mph Speed: 44 Pull Over / Drive
Light Number: 1 Status: GREEN
Light Number: 2 Status: RED
Light Number: 3 Status: RED
Light Number: 4 Status: RED
Start Cars
Pause Cars
Add Car
EXIT PROG...
Start Lights
Pause Lights
Add Interse...
Activate Windows


Traffic Simulation
Current Time: 19:07:48
Car # : 1 X Position: 2464 / 4000 Status: Driving at 20 mph Speed: 20 Pull Over / Drive
Car # : 2 X Position: 1908 / 4000 Status: Driving at 50 mph Speed: 50 Pull Over / Drive
Car # : 3 X Position: 1800 / 4000 Status: Driving at 125 mph Speed: 125 Pull Over / Drive
Car # : 4 X Position: 3040 / 4000 Status: Driving at 56 mph Speed: 56 Pull Over / Drive
Car # : 5 X Position: 2064 / 4000 Status: Driving at 44 mph Speed: 44 Pull Over / Drive
Exit?
Thank you for using this program. Goodbye!
OK
Cancel
Light Number: 1 Status: GREEN
Light Number: 2 Status: GREEN
Light Number: 3 Status: GREEN
Light Number: 4 Status: GREEN
Start Cars
Pause Cars
Add Car
EXIT PROG...
Start Lights
Pause Lights
Add Interse...
Activate Windows