CMSC 335 - Final Project 3

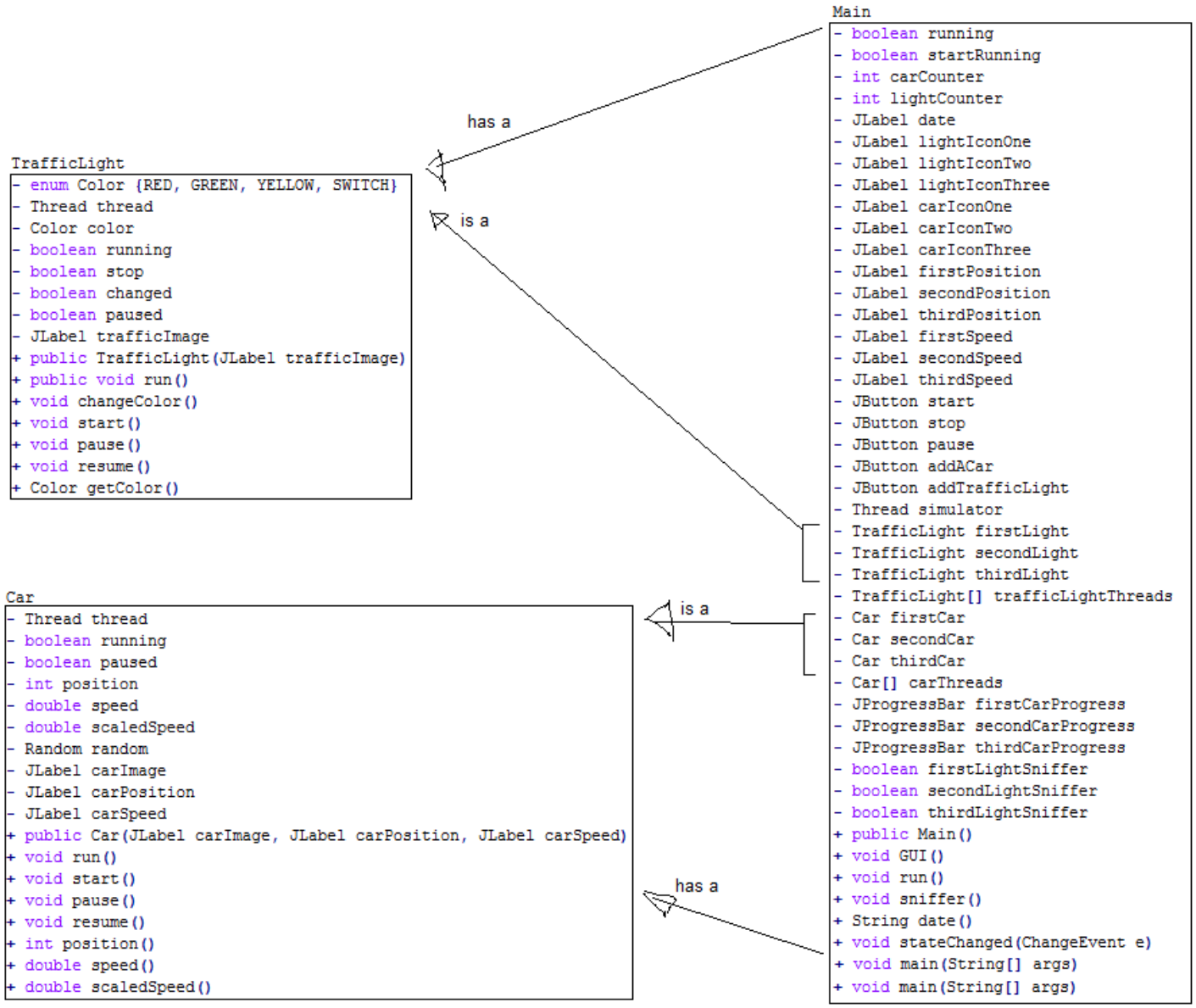
Traffic Simulation GUI

Angel Lee

Oct 2, 2021

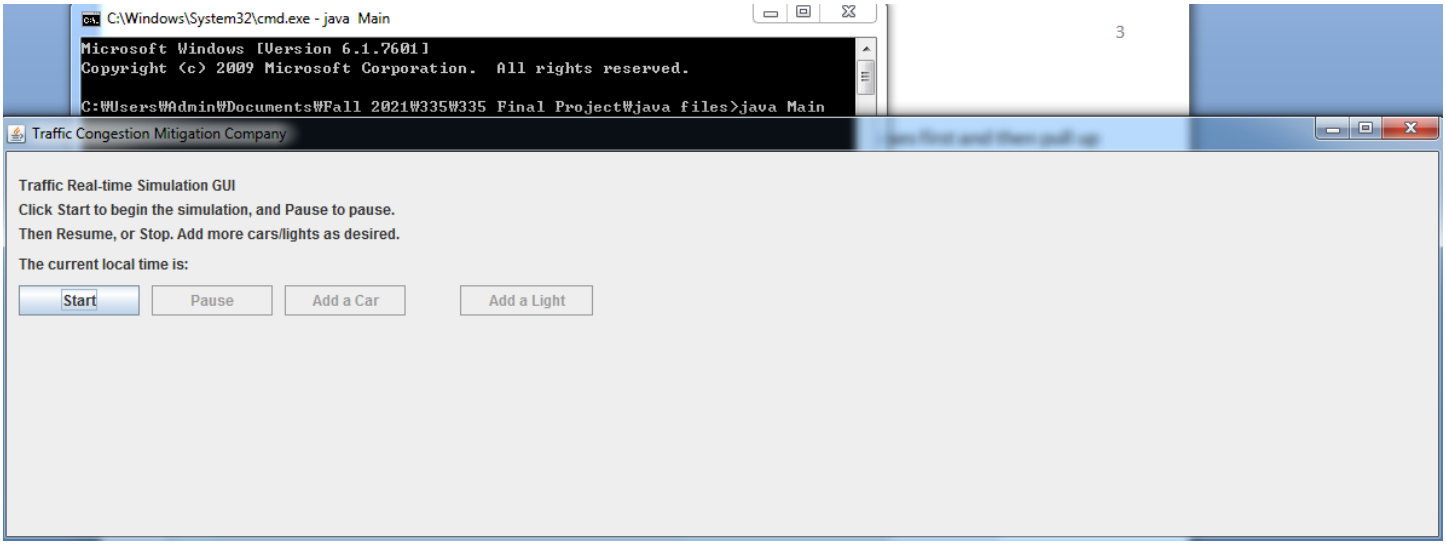
UML Diagram:

UML Class diagram of my program demonstration "has a" relationship (separate PNG file also in zip) plus the threads in the Main class which "is a" and then points to the superclasses.



Compilation on Command Prompt/Java SDK:

When opening this with the command prompt compiler, we first make the java classes first and then pull up the Main class. I type "cmd" onto the address bar to pop the command prompt right in the folder that I want. The start:



Program only allows the Start button to be an option at first, and you can add up to 2 more cars on top of the first car.

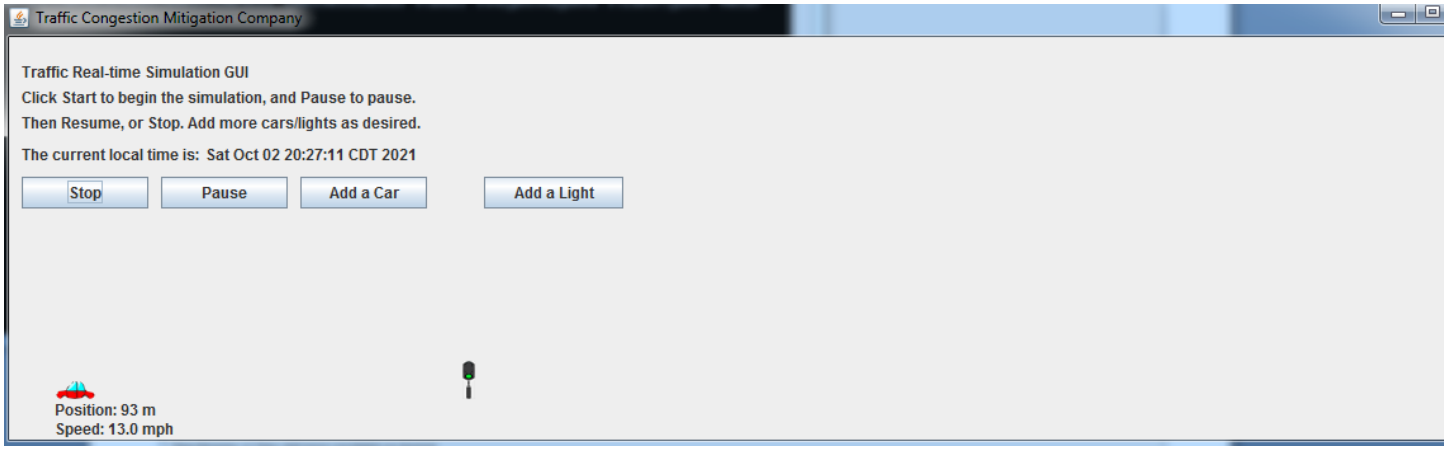
Test cases:

I have made 12 cases testing different buttons and observations to see how this program is functioning. There are 4 total buttons, but the buttons change or even enable/disable other buttons.
The cars in this simulation are all running horizontally, and the Position in this project is the x-position, assume that y-position is 0, so I didn't include it.

Test case:	Input:	Expected Output:	Pass?
1	Press Start button.	Other buttons become available, first traffic light appears and first car starts running, position and speed next to it on the traffic diagram. Start button switches to Stop and all the other buttons are also click-enabled. The date starts ticking in seconds.	Y
2	Press "Add A Car"	Another car with position and speed is added to the traffic diagram, and the button switches to "Add Another Car"	y
3	Press "Pause"	Everything should pause, including traffic light(s), and Pause button should switch to Resume	Y
4	Press "Resume"	Everything should go back to running, including the lights.	Y
5	Press "Add Another Car"	3rd car is added to the simulator diagram, and then button disabled	Y
6	Watch the lights change and ensure cars stop when it's red light	Cars within a certain distance (250km) from incoming intersection stop near red light	Y
7	Stop button is pressed	Everything should stop, and the Stop button switches to Exit so one can close the program	Y
8	Press "Add A Car" while threads are paused.	No different than test case #2, another car is added, and button switches to "Add Another Car"	Y
9	Press "Add Another Car" while threads are paused.	No different than test case #8, 3rd car is added to the diagram, and then button disabled	Y
10	Press "Resume" to let all the threads run	All three car threads run	y
11	Press "Add a Light" to add new intersection	2nd light is added, button changes to "Add Another Light", and cars now stop at the new light as well	Y
12	Press "Add Another Light" to add new intersection	3rd light is added, then button disables. The cars now stop at the 3rd light as well.	Y

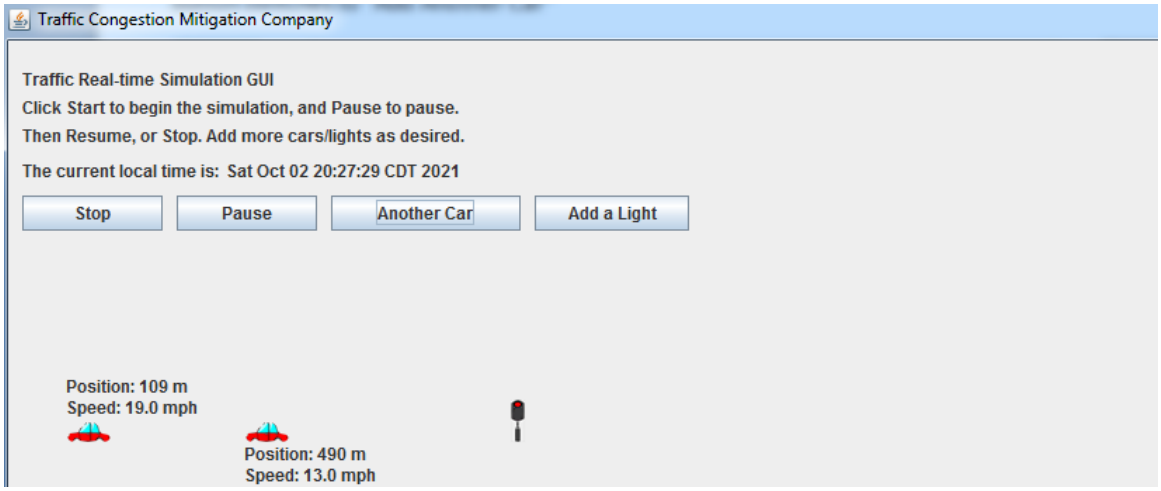
Test case 1:

Other buttons become available, first traffic light appears and first car starts running, position and speed next to it on the traffic diagram. Start button switches to Stop and all the other buttons are also click-enabled. The date starts ticking in seconds.



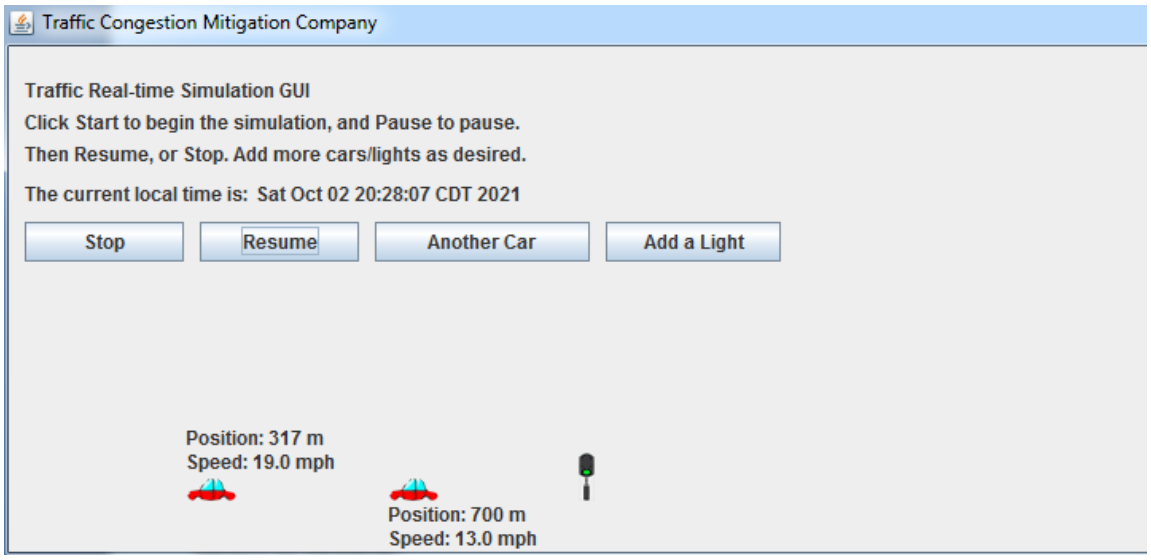
Test case 2:

When "Add A Car" is pressed, another car with position and speed is added to the traffic diagram, and the button switches to "Add Another Car"



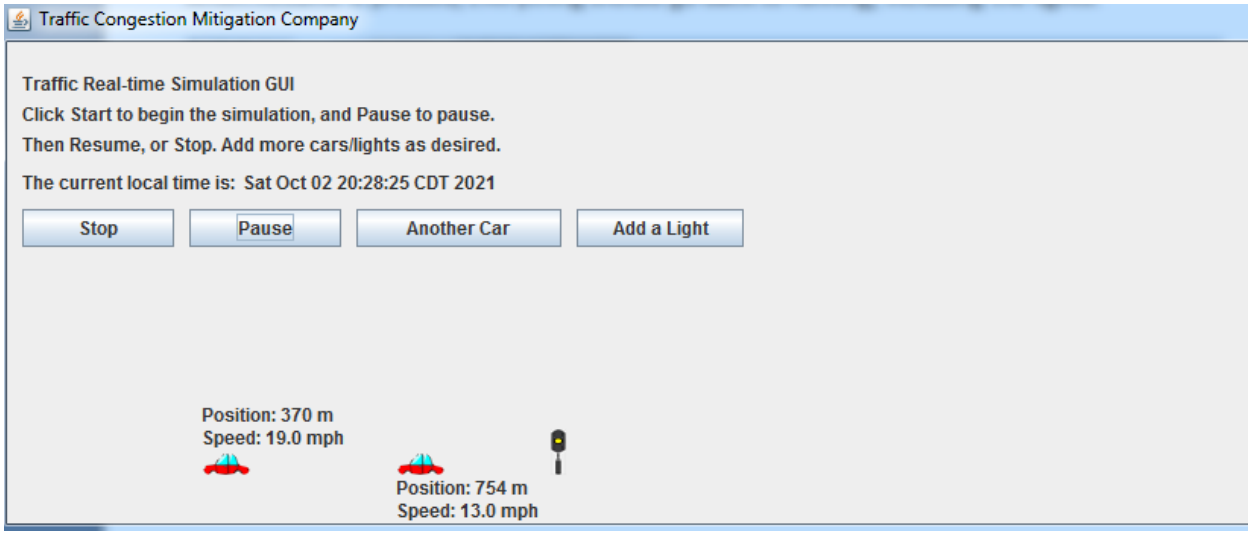
Test case 3:

The Pause button is pressed: Everything should pause, including traffic light(s), and Pause button should switch to Resume.



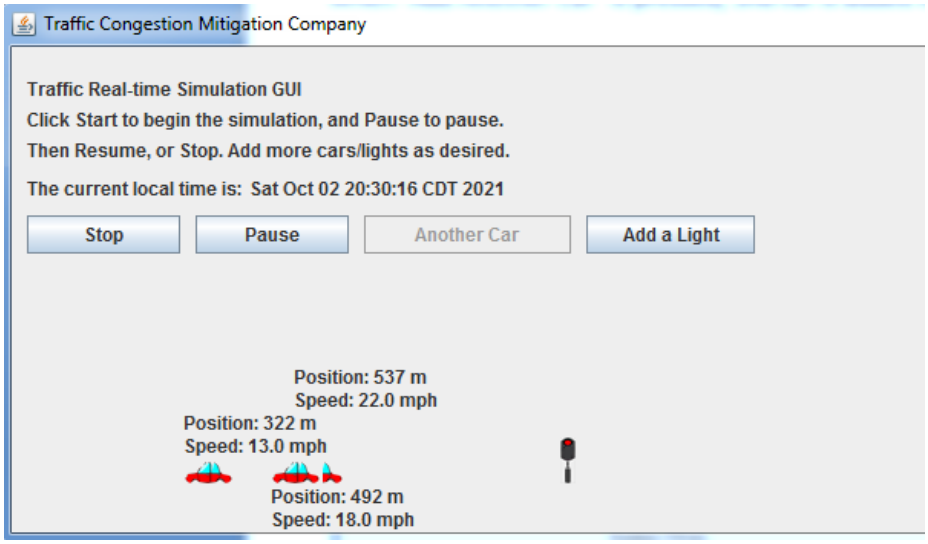
Test case 4:

When Resume is pressed, Everything should go back to running, including the lights.



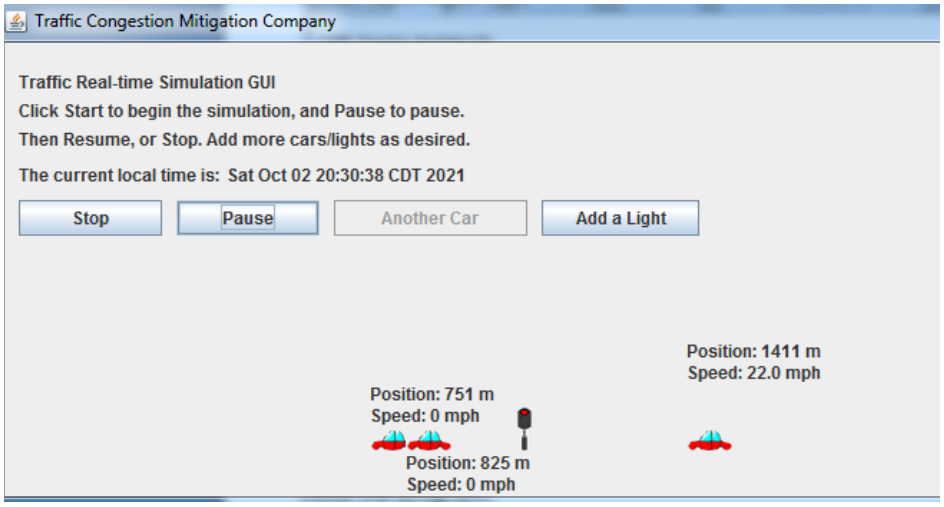
Test case 5:

When "Add Another Car" is pressed, 3nd car is added to the simulator diagram, and then button disabled.



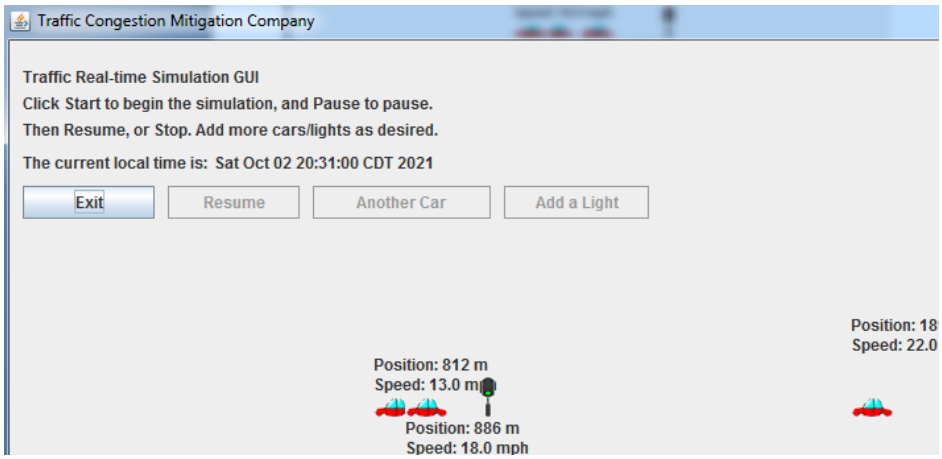
Test case 6:

When the light is red, the cars closing in on the intersections should stop. The two cars within a certain distance (250km, so from 750km to 1000km) from the incoming intersection stop near red light.



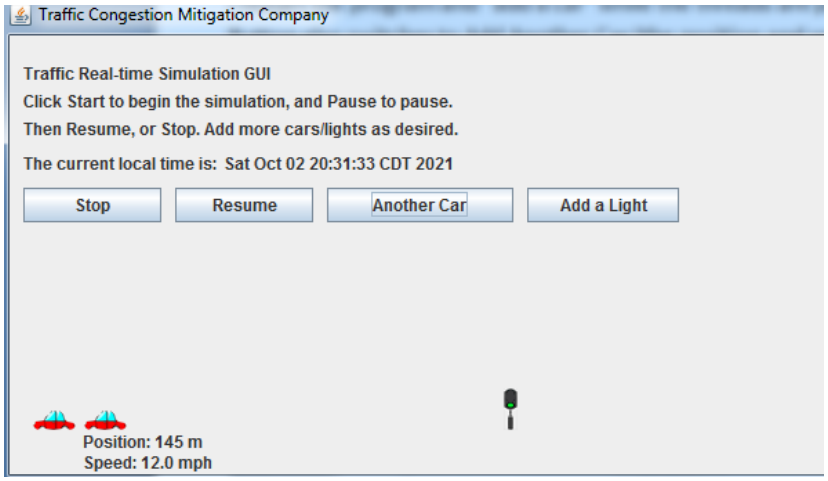
Test case 7:

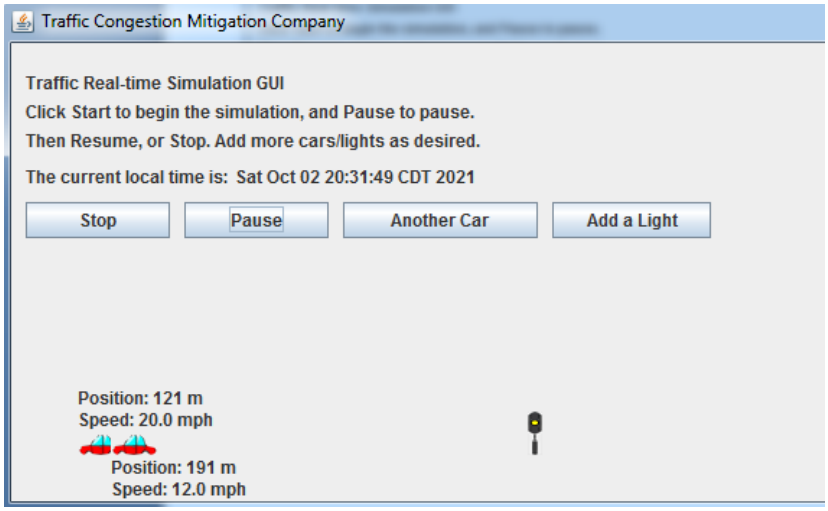
When Stop button is used, all the threads should stop (well technically Paused, as stopping a thread can cause errors and it should be last resort). Everything should stop, and the Stop button switches to Exit so one can close the program.



Test Case 8:

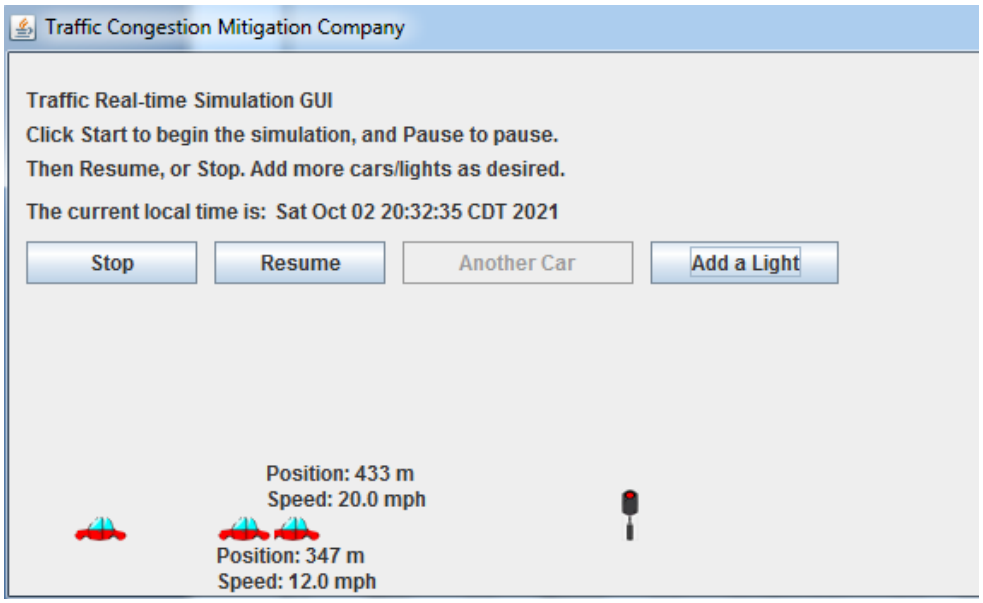
I restart the program and "add a car" while the threads are paused, to ensure there is no error - and it works. Button also switches to Add Another Car (the position and speed will start after resuming threads).





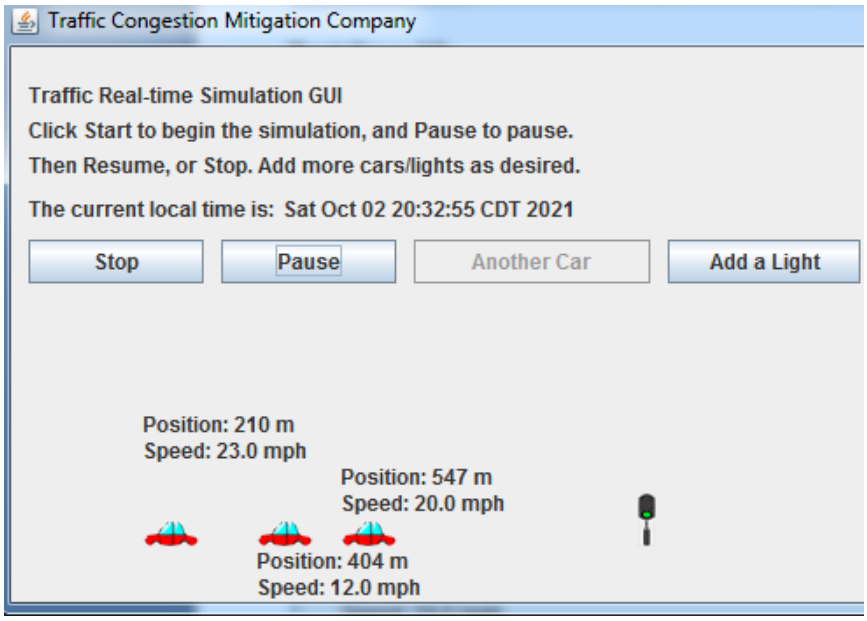
Test Case 9:

I "add another car" to ensure a car thread can be added without error, and it works. 3rd car (the one without position or speed yet) is added to the diagram, and then button disabled. No more car threads can be added.



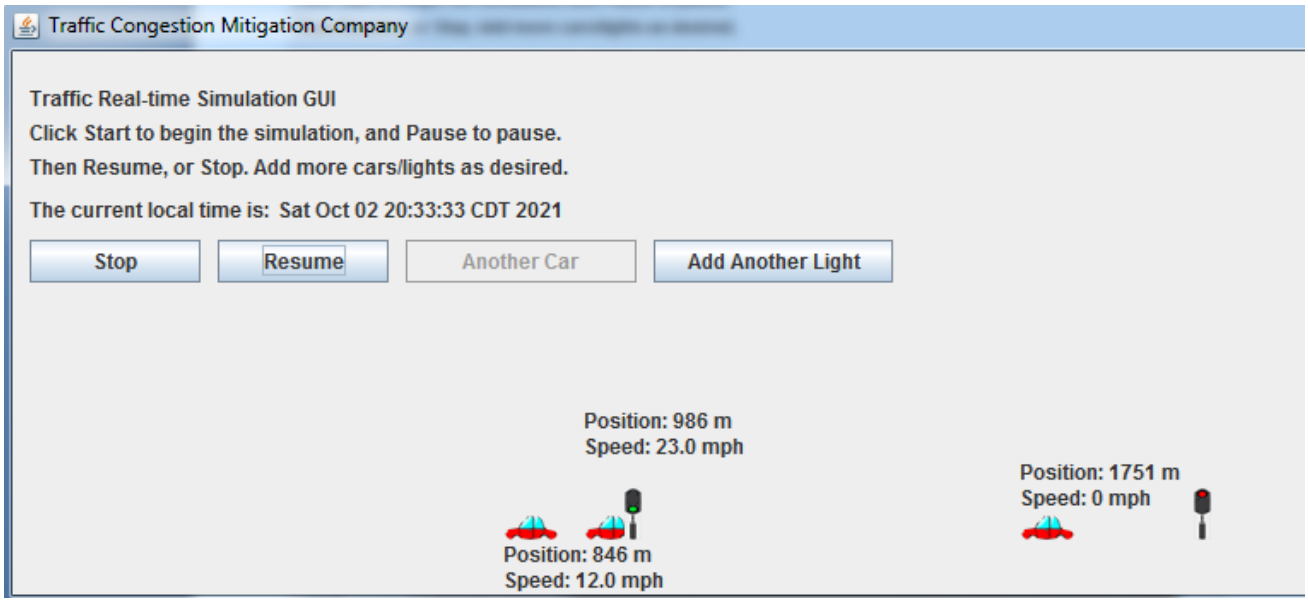
Test Case 10:

Resume thread runs after adding the two car threads, and it works. All three car threads run.



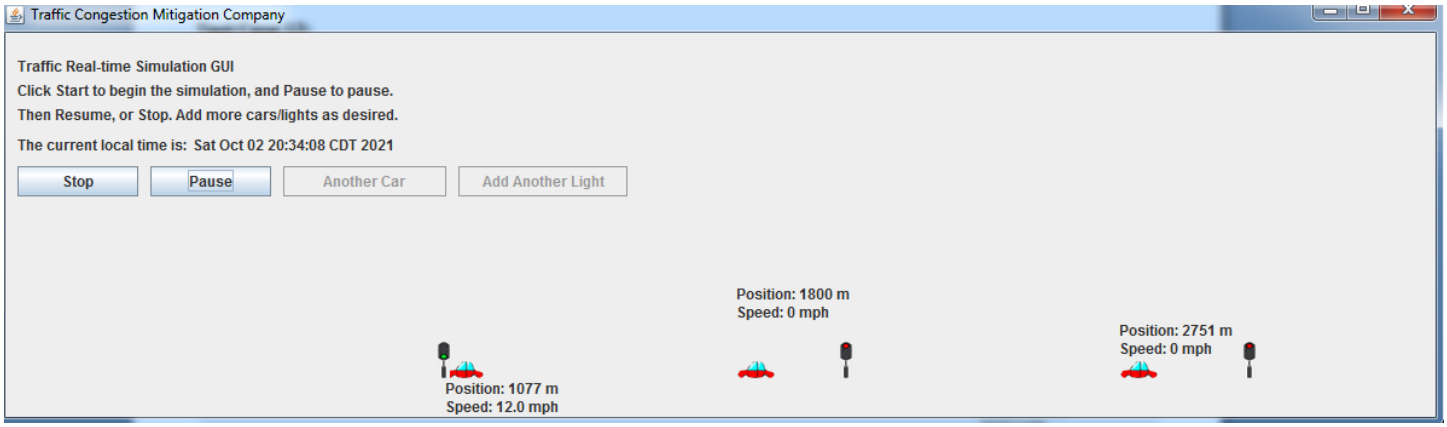
Test Case 11:

Add another traffic light with the button "Add a Light", and it works. A new light is added, and the button changes to "Add Another Light". The cars now stop at the new traffic light as well:



Test Case 12:

Add third traffic light with the button "Add Another Light", and it works. 3rd light is added, then button disables. The cars now stop at the 3rd light as well.



Conclusion:

The program has successfully done all the test cases as expected. This is definitely the hardest project out of the three we have done in this class (for me at least) and the longest code I've written.

What I learned from this program is how to execute multiple threads based off of a "superclass". One I got the hang of make a thread off of a class object, it was a matter of repeating the start of other threads then.

Not only did I have to learn to start threads, but also give the user the option of starting threads (both cars and intersections) at different times of the choice (through the JButtons) and in a way that won't interfere with the running of other threads. I also learned more deeply about the nature of threads, while I was trying to figure out how I will create a JButton that will add more car threads to the simulator. I spent an unnecessary amount of time trying to start more threads with JButtons, and thought that the way to do that was to get the program to check if the second car and light threads were "running" or "alive", which didn't work, so I resorted to counting mouse clicks instead (first click on the AddACar button starts Thread 2, while 2nd click will start Thread 3 and then disable). Figuring out how to make the cars stop near at the red light was another challenge, because I had to figure out not only how to make the speed show 0 and also pause the little car diagrams (and then resume them back to their original speed at green light) but also to detect which cars are nearing which traffic lights. So I made "sniffers" around each traffic light that would check the array of car threads and if a car was within a certain position (the positions right before the traffic light) then the car would stop when the traffic light's case is RED. But if I resumed at case GREEN, then that got in the way of the Pause button that was trying to pause the car threads, so I had to make a new enum case called SWITCH that would resume the cars for a split second but not get in the way of the Pause button's resume() method that would pause the cars.

I liked that I had more freedom regarding how to design this program, but on the other hand the options are endless so one has to figure out what works best. For example, I wanted to input pictures into a real-time simulating diagram, and had to figure out what tools in the JDK will help make the pictures move according to the position of the running cars. I used the TrafficLightDemo.java from the code samples provided in class and used the enum cases as a way to switch the traffic light pictures around based on color. Also, in order to put the cars on a horizontal position and keep updating the car positions on the diagram (repaint the JLabel image positions), I had to use JProgressBar, which worked with the "setBounds" position of the car icons based off the progress of the cars' JProgressBars.