Final Project

Traffic Simulator

Samuel Scalf

CMSC 335 · Dr. Osama Morad

University of Maryland Global Campus

Table of Contents

User's Guide

System Requirements

Space: Less than 1 MB

Software: Java Development Kit (JDK) version 8 update 261 or later

Compiling the Program

Before compiling the program, the downloaded zip file needs to be uncompressed. The means of uncompressing, or extracting, a zip file is not covered here. Open the directory containing the extracted files, if not already there. Open the "src" folder. Compilation of the program requires the JDK version 8. The procedures required for this are not covered in this guide, but can be found on Oracle's website.

Windows 10

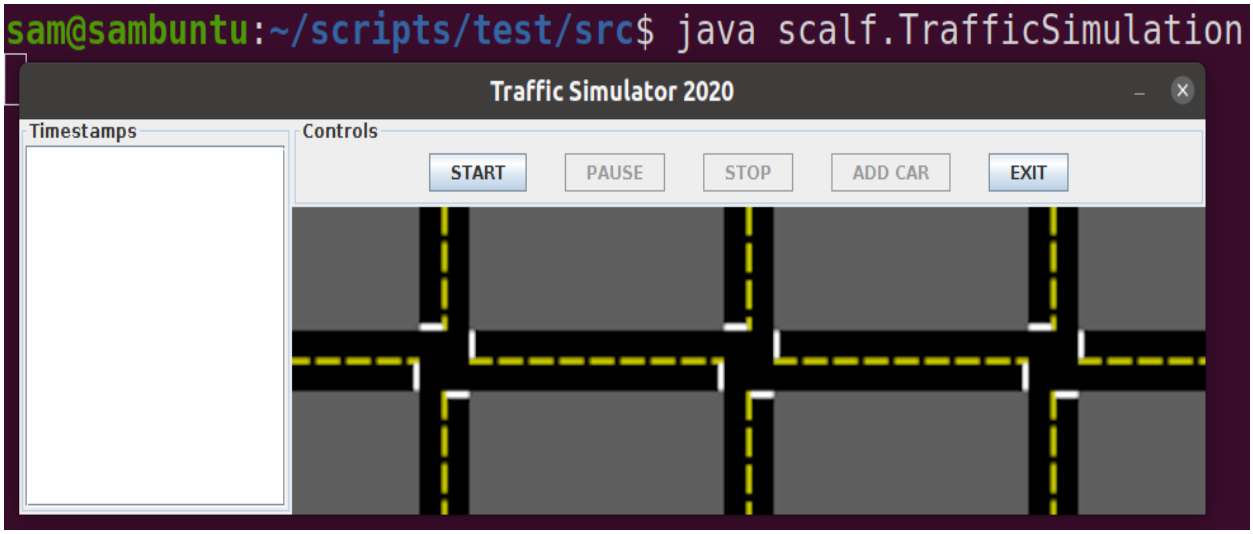
Click File > Open Command Prompt > Open Command Prompt. Type "javac .\scalf\TrafficSimulator.java" without the quotes and press the "Enter" key.

Linux

Open a terminal in the location of the files. Type "javac ./scalf/TrafficSimulator.java" without the quotes and press the "Enter" key.

Using the Program

To run the program, from the command prompt or terminal (see Compilation steps), type "java scalf.TrafficSimulator" without the quotes and press the "Enter" key. The program is command line-driven, so all user input will be in the command prompt or terminal. To respond to questions, type your answer and press the "Enter" key.



Documentation

Additional information about the packages and classes can be found by opening "index.html" from the downloaded "doc" folder in a web browser. The website was created using javadoc, so it has the same format as the Oracle Java 8 API.

Testing

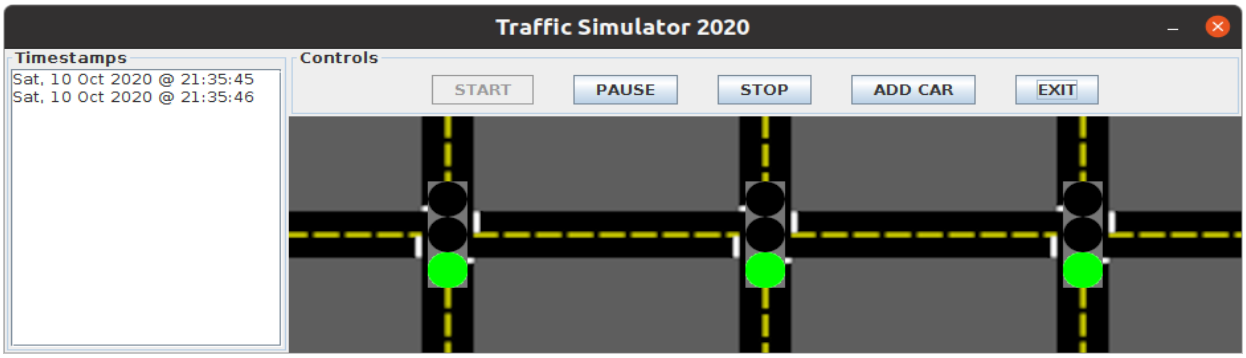
The only test cases I could think of for this program were the only interactive features: the buttons. Since there are five buttons, each test case will be for a different button. Since cars cannot be added when the program is stopped or paused, it will be test case 2.

Test Case 0: Timestamps

While the program is active, timestamps should be appearing every second in the Timestamps area. These should pause and resume with everything else. The timestamps should be cleared when the simulation is stopped, not paused. This was marked as a test case, because the timestamps were a requirement. I needed to verify that it was functioning properly along with everything else.

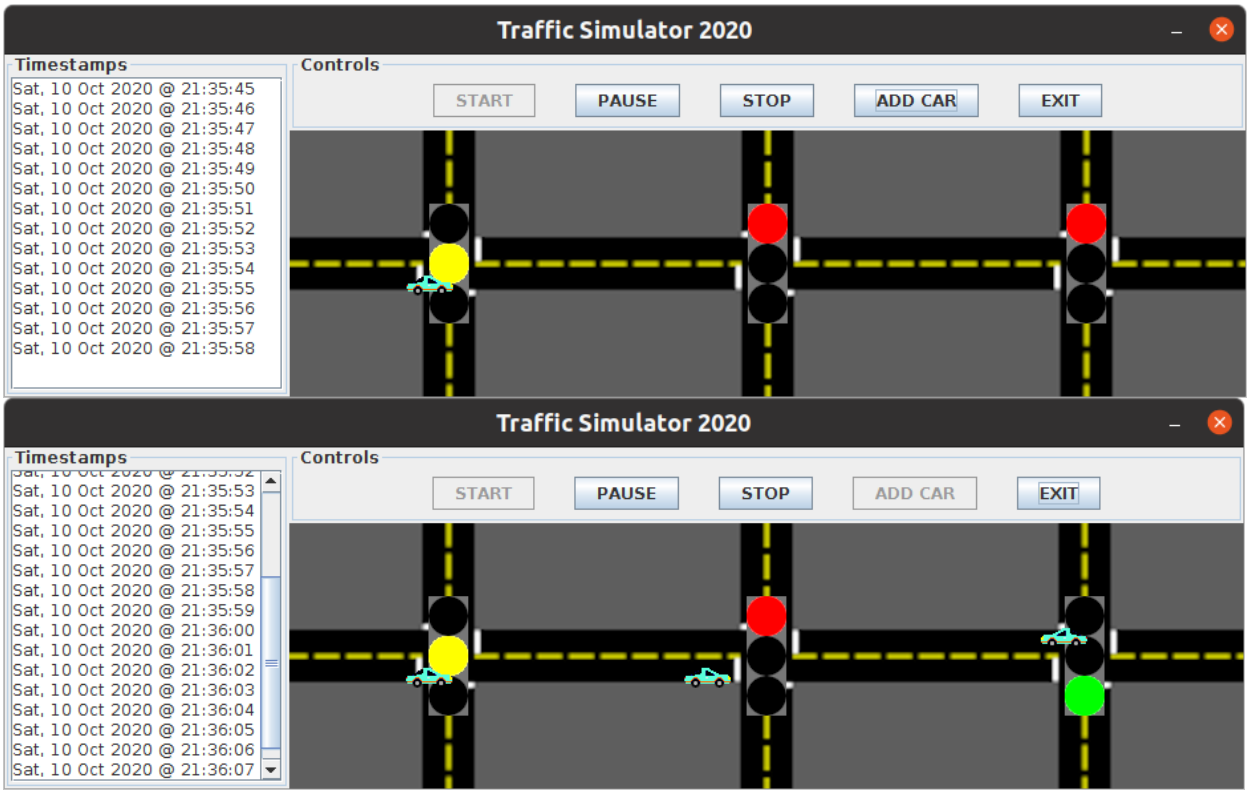
Test Case 1: Start Button

When the “START” button is pressed, a few things are expected to happen. First, the three traffic lights are expected to appear and start changing lights. Also the timestamps should start to appear in the “Timestamps” area every second. The “START” button should become disabled, while the “PAUSE”, “STOP”, and “ADD CAR” buttons become enabled.



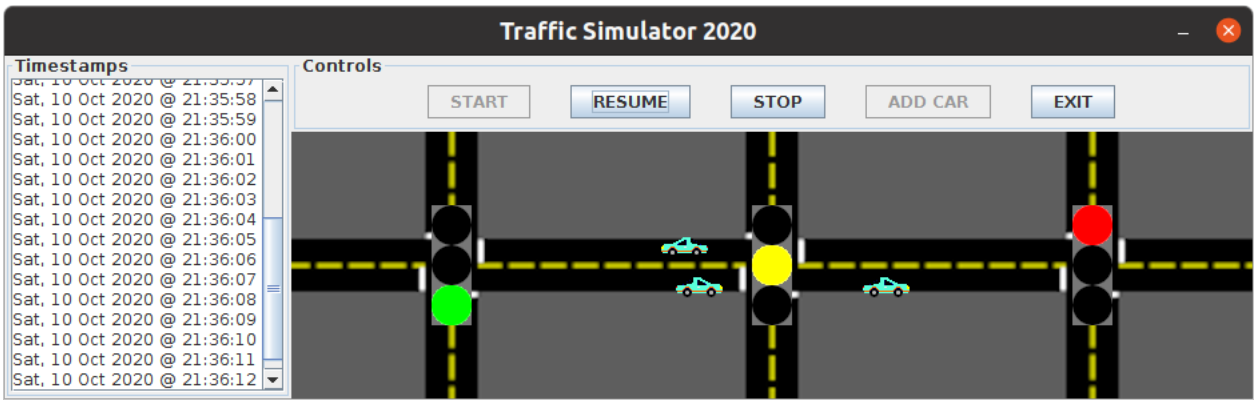
Test Case 2: Add Car Button

When the “ADD CAR” button is pressed, a car should appear either in the right-bound lane on the left or the left-bound lane on the right. This should hold for up to three cars, at which point the “ADD CAR” button becomes disabled (second screenshot) until a car leaves the screen (not captured).



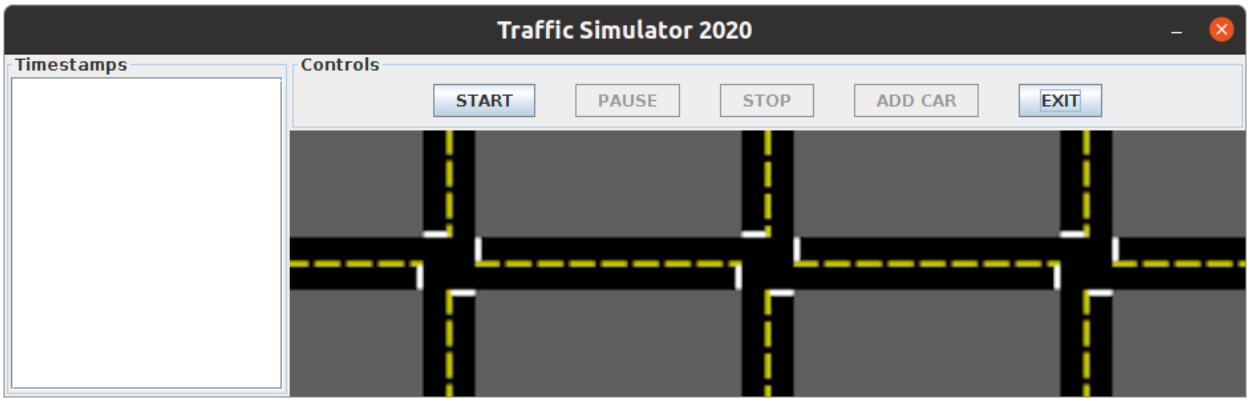
Test Case 3: Pause Button

When the “PAUSE” button is pressed, all threads, other than the main or GUI threads, should pause, or **wait** for a notification to resume. The text of the button should also change to “RESUME” and the “ADD CAR” button should become disabled.



Test Case 4: Stop Button

Clicking the “STOP” button should stop all threads and put the simulation back to the default, pre-run state. The “PAUSE”, or “RESUME,” button should become disabled and display “PAUSED.” The “STOP” and “ADD CAR” buttons should become disabled, while the “START” button becomes enabled.



Test Case 5: Exit Button

Clicking the “EXIT” button should exit the program completely and return the user to the generating shell.

Lessons Learned

I've learned more by making this project than I have from the rest of the course combined. Actively thinking in multiple threads took a while to get used to, but my brain eventually made the switch. I think the hardest thing was capturing a "mutual" lock for all threads, updating the GUI properly, and getting the cars to wait at red lights and behind other cars.

After exploring the Producer-Consumer problem, I began to understand the synchronized keyword, but that still felt like more of a one-at-a-time thing. I was able to get multiple threads running, but I was also using my Synchronizer object as the lock, which was causing timing issues. I was finally able to resolve the issues by creating a ReentrantLock and passing the "reference" to the lock to all Threads, I was able to use "lock.notifyAll()" from the Synchronizer class that instantiated the lock.

Once I got multiple threads, starting, stopping, and pausing properly, I needed to get them displaying properly. I originally had the traffic lights extend JLabel and changed the icon, but I quickly realized I could not get accurate hit-boxes for these. So I threw out the idea and rebuilt the traffic lights to simply have a getImage() method that could be called from the TrafficLight object that was passed to the GUI. It was fun finding out the absolute coordinates needed to line up the lights at their respective intersections. I also learned that the Cars definitely needed a Thread.sleep(long) method call or they would move far too fast.

The last biggest hurdle was collision detection for cars-on-lights and cars-on-cars. I at first just checked if the hit-boxes intersected, but quickly found this could cause a car to stop right before finishing passing a red light. This was because the back of the car still intersected

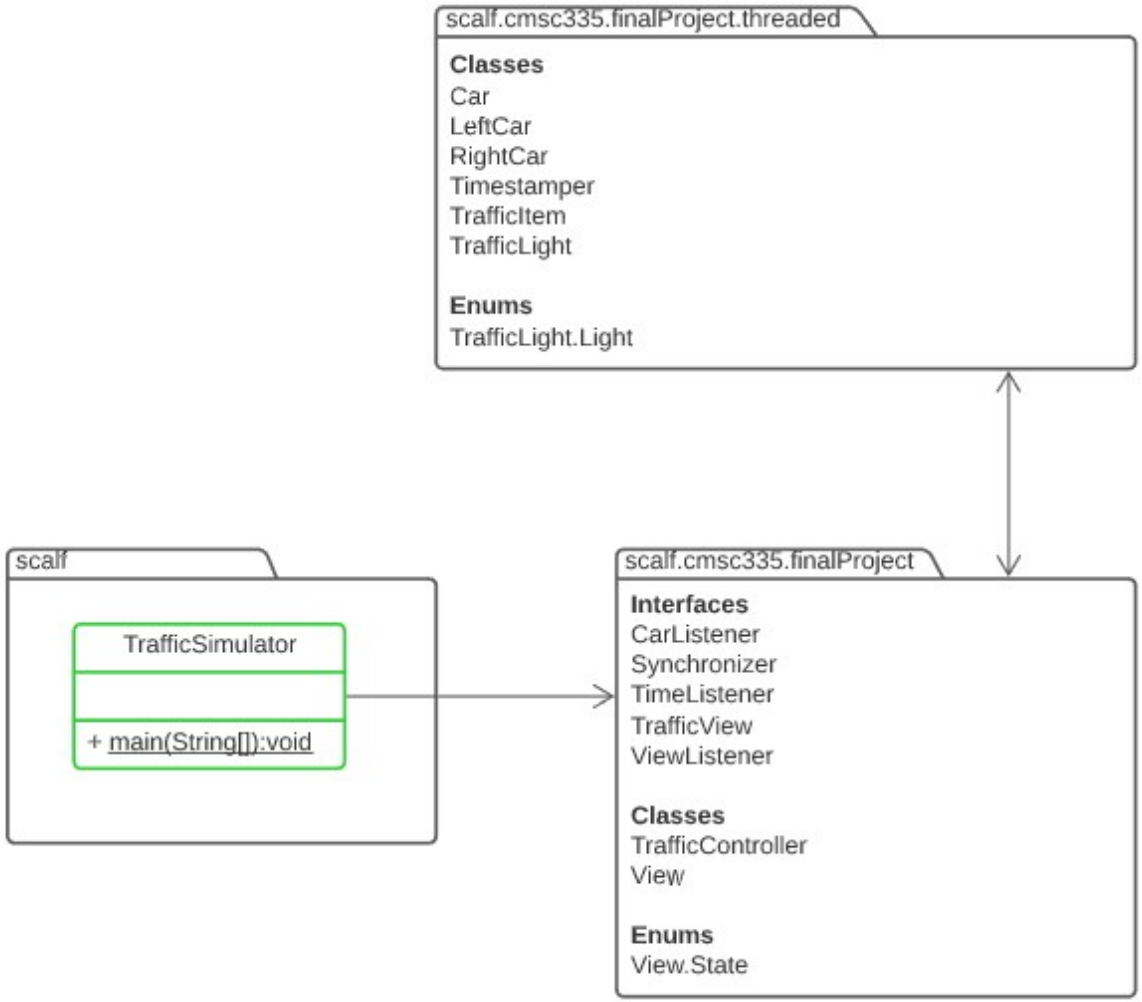
with the light. I knew I needed edge-detection, which I never truly figured out. I created a new hit-boxes on the left and right of the light and front of the cars. This was small enough that the cars were finally waiting at lights like good drivers. They were still running over each other, though, so I needed to check if the cars were stuck behind others. I used somewhat similar techniques used for the lights with some minor modifications. Once working, this led to some very relatable situations like a “fast” car catching up to a slow one and getting stuck behind it. I felt for those cars.

I also tried to hone some other skills in this project. I created all of the images for the traffic: background, lights, and cars. I also attempted to get my feet wet with the MVC model. I could definitely improve my implementation and truly abstract the classes away from each other. I still learned a lot from the experience, though.

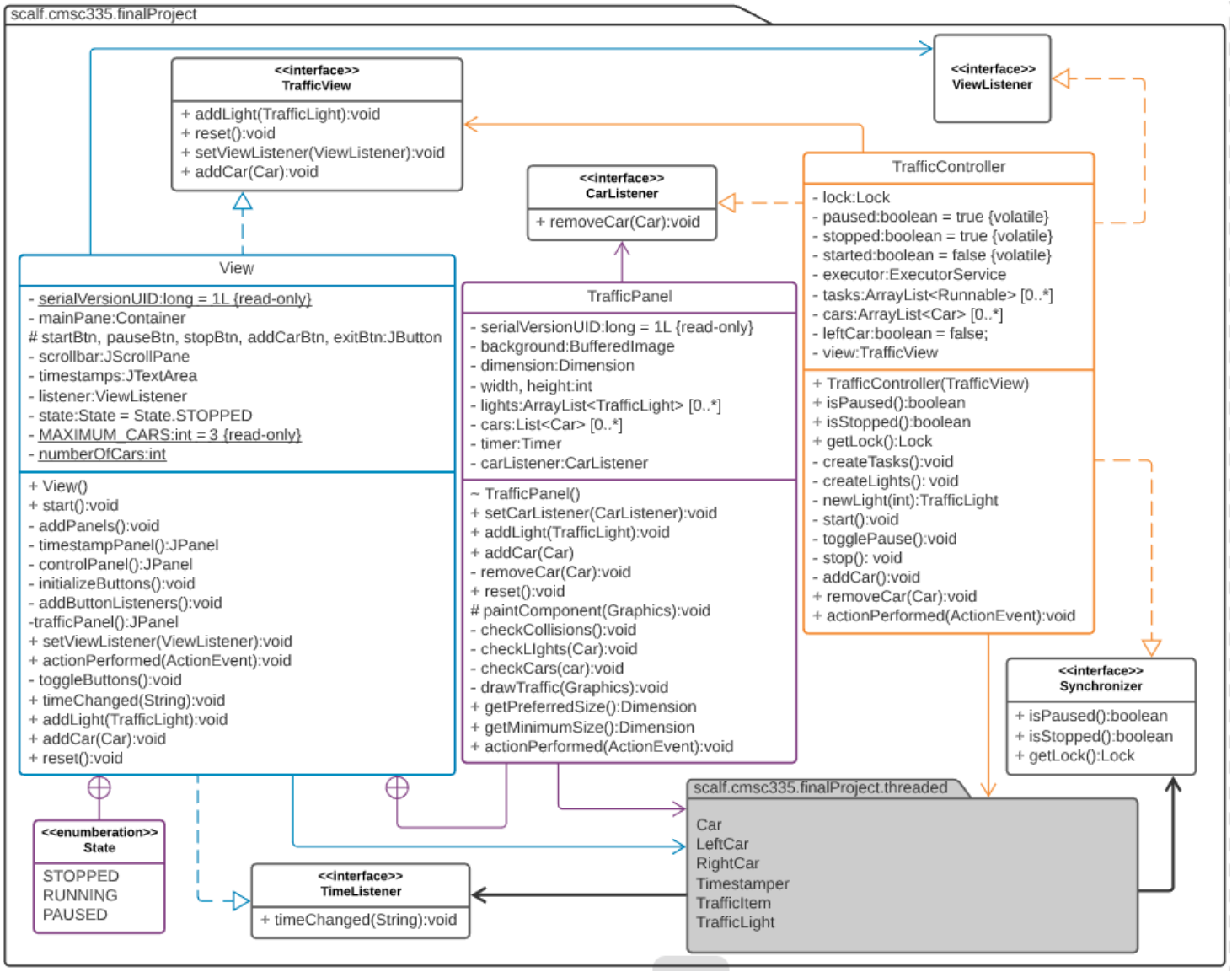
I’ve learned so much from this class, and I got inspiration for how I could use these new skills at work. It was a very rewarding challenge that I was not expecting. I do not feel I was prepared enough going into the project, but I was able to stay afloat despite swallowing a bit of water. I am looking forward to my next classes and the challenges they may present. It has been quite the ride.

Appendix A: UML Class Diagram

Package scalf



Package scalf.cmsc335.finalProject



Package scalf.cmsc335.finalProject.threaded