Developer's Guide to Project 3

Eric Sampson

University of Maryland Global Campus

CMSC 335 7381 Object-Oriented and Concurrent Programming

Professor Seely

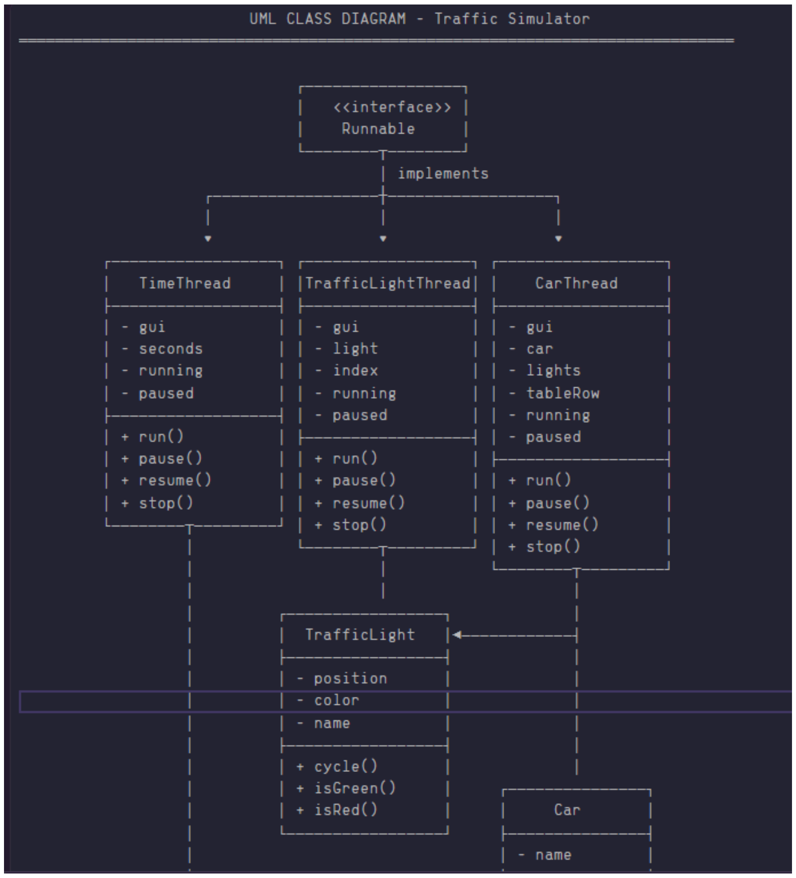
December 7, 2025

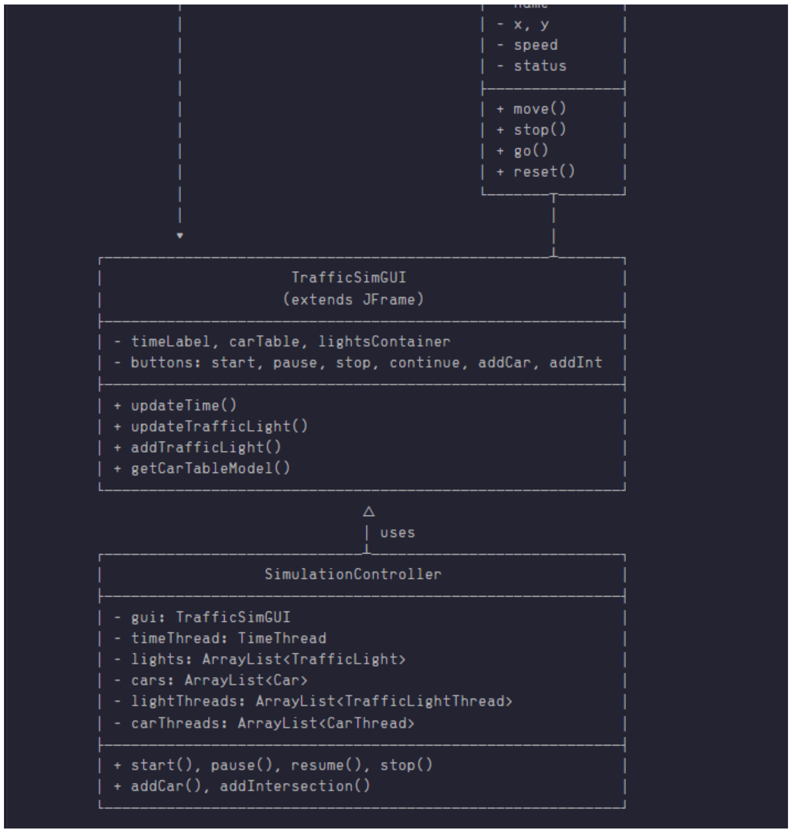
Developer's Guide

Compile and Execute

To run this program successfully, open and execute the 'TrafficSimGUI.java' file in any supported IDE or code editor with the appropriate Java extensions installed. Alternatively, you can compile and run the program from the command line by navigating to the project directory and entering the following commands: 'TrafficSimGUI.java' and 'TrafficSimGUI.java'.

Class Diagram





```
                                   | - x, y          |
                                   | - speed         |
                                   | - status        |
                                   |-----------------|
                                   | + move()        |
                                   | + stop()        |
                                   | + go()          |
                                   | + reset()       |
                                   -------------------

 ------------------------------------------------------------
 |                      TrafficSimGUI                       |
 |                    (extends JFrame)                      |
 |----------------------------------------------------------|
 | - timeLabel, carTable, lightsContainer                   |
 | - buttons: start, pause, stop, continue, addCar, addInt  |
 |----------------------------------------------------------|
 | + updateTime()                                           |
 | + updateTrafficLight()                                   |
 | + addTrafficLight()                                      |
 | + getCarTableModel()                                     |
 ------------------------------------------------------------
                              △
                              | uses
 ------------------------------------------------------------
 |                  SimulationController                    |
 |----------------------------------------------------------|
 | - gui: TrafficSimGUI                                     |
 | - timeThread: TimeThread                                 |
 | - lights: ArrayList<TrafficLight>                        |
 | - cars: ArrayList<Car>                                   |
 | - lightThreads: ArrayList<TrafficLightThread>            |
 | - carThreads: ArrayList<CarThread>                       |
 |----------------------------------------------------------|
 | + start(), pause(), resume(), stop()                     |
 | + addCar(), addIntersection()                            |
 ------------------------------------------------------------
```

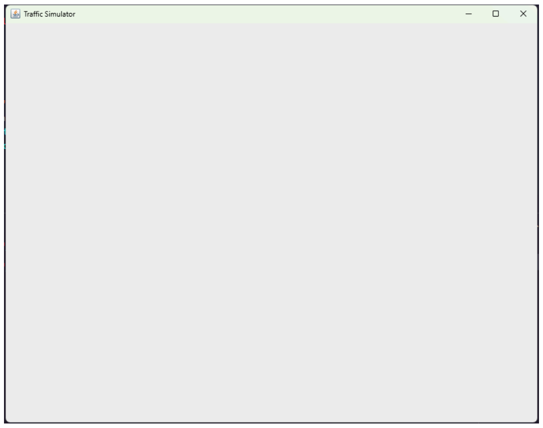
Test Plan

Test # 1

Expected Result: The GUI is initialized with a blank panel and a title.

Actual Result: GUI initialized with blank panel and title.

Pass/Fail: Pass

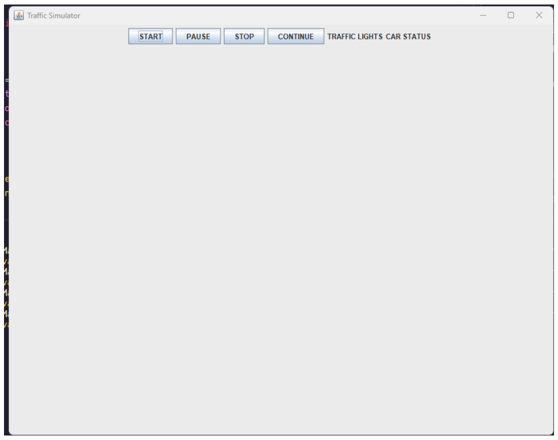


Test # 2

Expected Result: Creates necessary buttons and labels.

Actual Result: Creates necessary buttons and labels.

Pass/Fail: Pass

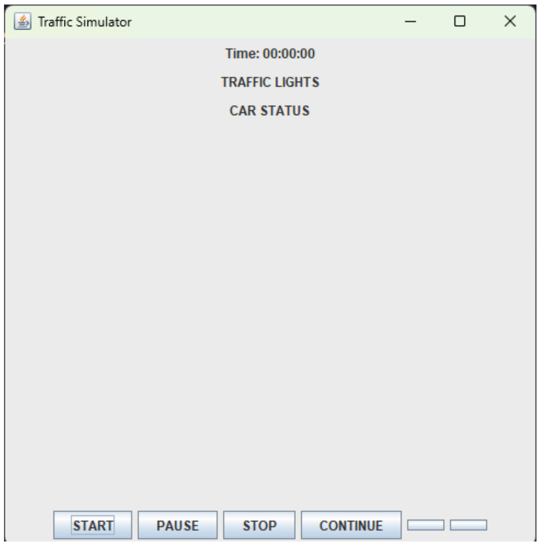


Test # 3

Expected Result: GUI displays changes in layout and added panels.

Actual Result: GUI displays layout changes and added panels.

Pass/Fail: Pass

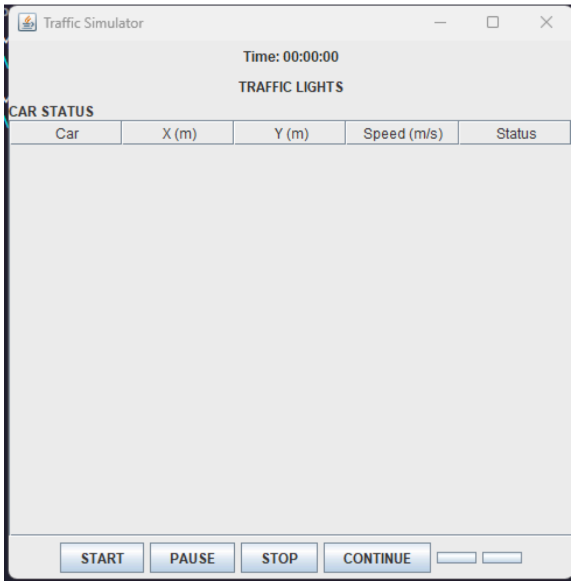


Test # 4

Expected Result: Displays the car table.

Actual Result: Displays the car table.

Pass/Fail: Pass



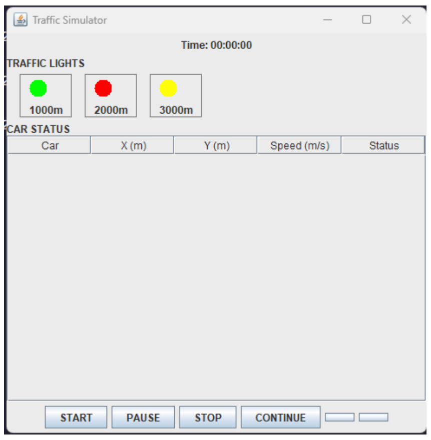
Car	X (m)	Y (m)	Speed (m/s)	Status

Test # 5

Expected Result: GUI displays traffic light indicators.

Actual Result: GUI displays traffic light indicators.

Pass/Fail: Pass



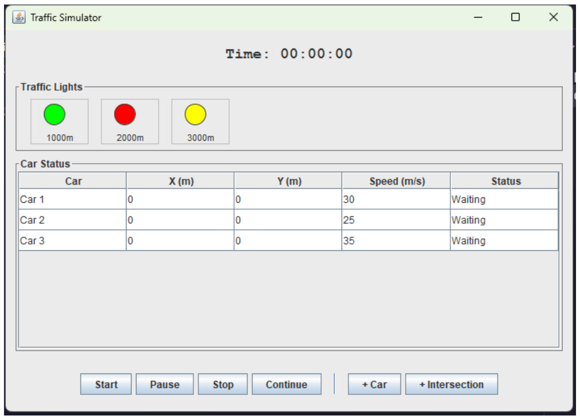
Car	X (m)	Y (m)	Speed (m/s)	Status

Test # 6 *AI suggestions and polish implementations

Expected Result: Polished GUI with mock data implementation,

Actual Result: Polished GUI with mock data implementation,

Pass/Fail: Pass



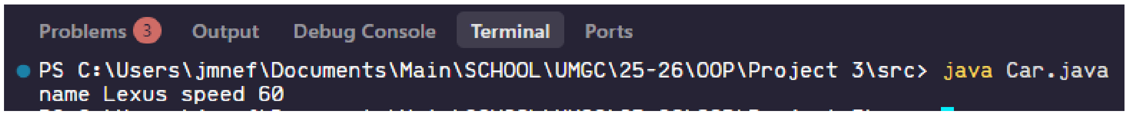
Car	X (m)	Y (m)	Speed (m/s)	Status
Car 1	0	0	30	Waiting
Car 2	0	0	25	Waiting
Car 3	0	0	35	Waiting

Test # 7

Expected Result: Car class constructor initialized properly.

Actual Result: Car class constructor initialized properly.

Pass/Fail: Pass



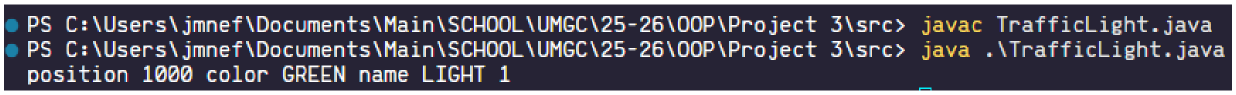
```
PS C:\Users\jmnef\Documents\Main\SCHOOL\UMGC\25-26\OOP\Project 3\src> java Car.java
name Lexus speed 60
```

Test # 8

Expected Result: Traffic light constructor initialized properly.

Actual Result: Traffic light constructor initialized properly.

Pass/Fail: Pass



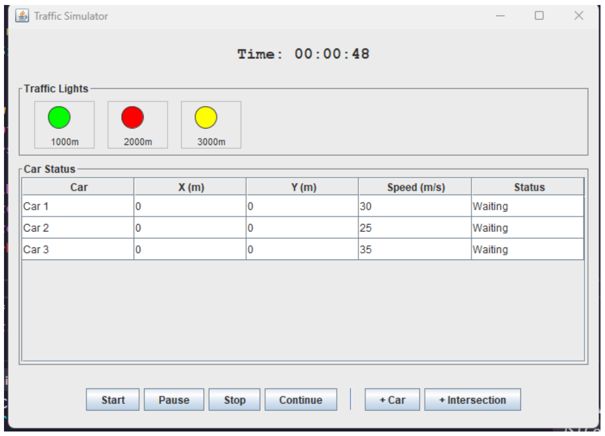
```
PS C:\Users\jmnef\Documents\Main\SCHOOL\UMGC\25-26\OOP\Project 3\src> javac TrafficLight.java
PS C:\Users\jmnef\Documents\Main\SCHOOL\UMGC\25-26\OOP\Project 3\src> java .\TrafficLight.java
position 1000 color GREEN name LIGHT 1
```

Test # 9

Expected Result: Created TimeThread handles GUI time calculation.

Actual Result: Created TimeThread handles GUI time calculation.

Pass/Fail: Pass



Time: 00:00:48

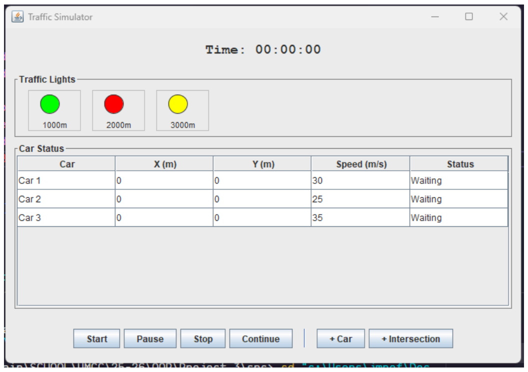
Car	X (m)	Y (m)	Speed (m/s)	Status
Car 1	0	0	30	Waiting
Car 2	0	0	25	Waiting
Car 3	0	0	35	Waiting

Test # 10

Expected Result: TimeThread starts the simulation timer when “Start” is pressed.

Actual Result: The simulation timer does not start when the “Start” button is pressed.

Pass/Fail: Fail



Time: 00:00:00

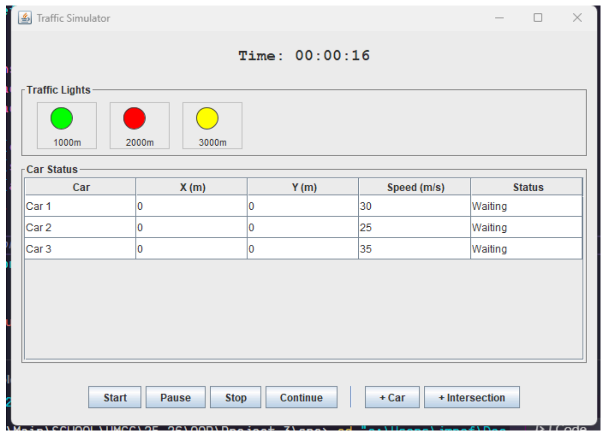
Car	X (m)	Y (m)	Speed (m/s)	Status
Car 1	0	0	30	Waiting
Car 2	0	0	25	Waiting
Car 3	0	0	35	Waiting

Test # 11

Expected Result: TimeThread starts the simulation timer when “Start” is pressed.

Actual Result: TimeThread starts the simulation timer when “Start” is pressed.

Pass/Fail: Pass



Time: 00:00:16

Car	X (m)	Y (m)	Speed (m/s)	Status
Car 1	0	0	30	Waiting
Car 2	0	0	25	Waiting
Car 3	0	0	35	Waiting

Test # 12

Expected Result: Buttons operate as intended with TimeThread.

Actual Result: Buttons operate as intended with TimeThread.

Pass/Fail: Pass

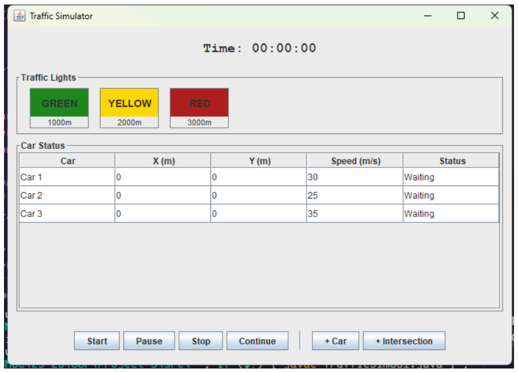
note: SS omitted because it is difficult to capture button functionality

Test # 13

Expected Result: TrafficLightThreads cycle.

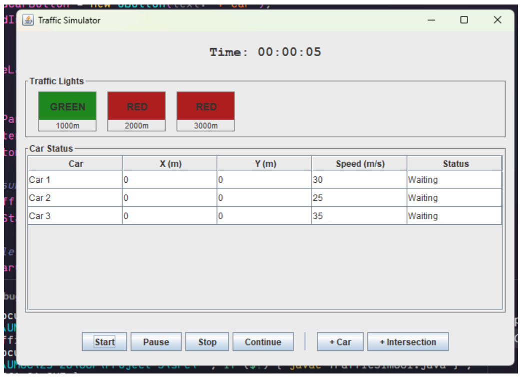
Actual Result: TrafficLightThreads cycle.

Pass/Fail: Pass



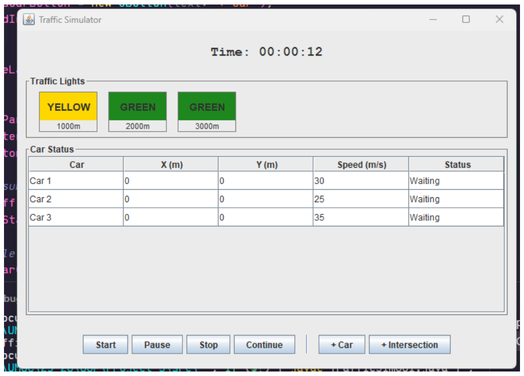
Time: 00:00:00

Car	X (m)	Y (m)	Speed (m/s)	Status
Car 1	0	0	30	Waiting
Car 2	0	0	25	Waiting
Car 3	0	0	35	Waiting



Time: 00:00:05

Car	X (m)	Y (m)	Speed (m/s)	Status
Car 1	0	0	30	Waiting
Car 2	0	0	25	Waiting
Car 3	0	0	35	Waiting



Time: 00:00:12

Car	X (m)	Y (m)	Speed (m/s)	Status
Car 1	0	0	30	Waiting
Car 2	0	0	25	Waiting
Car 3	0	0	35	Waiting

Test # 14

Expected Result: Button functionality works with TrafficLightThreads.

Actual Result: Lights continue to cycle when pause is pressed.

Pass/Fail: Fail

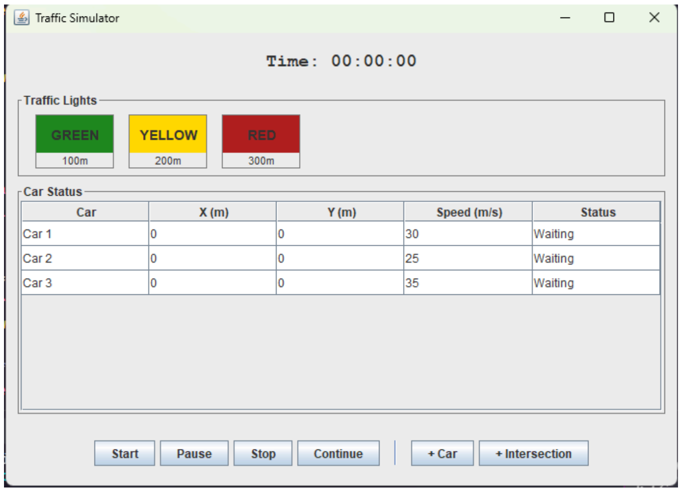
note: SS omitted because of difficulty in capturing logic

Test # 15

Expected Result: CarThreads move forward at intended speeds.

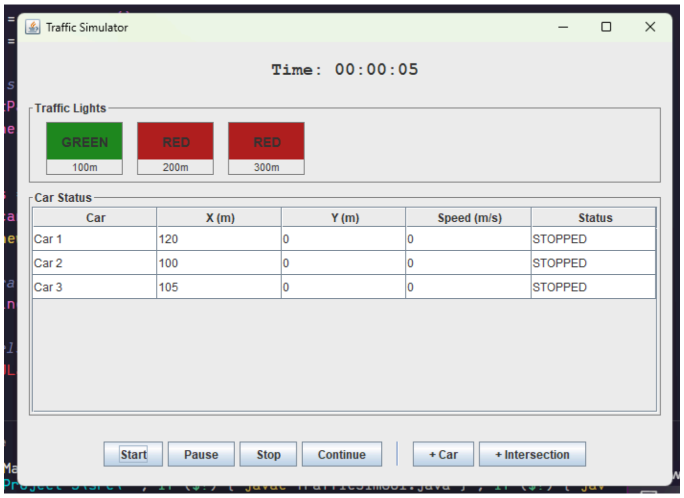
Actual Result: CarThreads progress at the intended speeds.

Pass/Fail: Pass



Car Status

Car	X (m)	Y (m)	Speed (m/s)	Status
Car 1	0	0	30	Waiting
Car 2	0	0	25	Waiting
Car 3	0	0	35	Waiting



Car Status

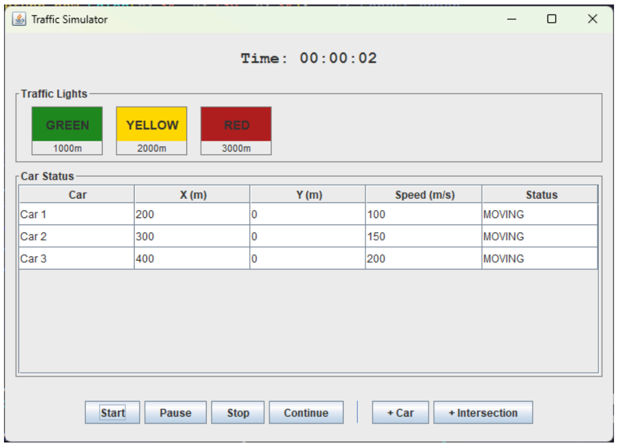
Car	X (m)	Y (m)	Speed (m/s)	Status
Car 1	120	0	0	STOPPED
Car 2	100	0	0	STOPPED
Car 3	105	0	0	STOPPED

Test # 16

Expected Result: Buttons function as intended with CarThreads.

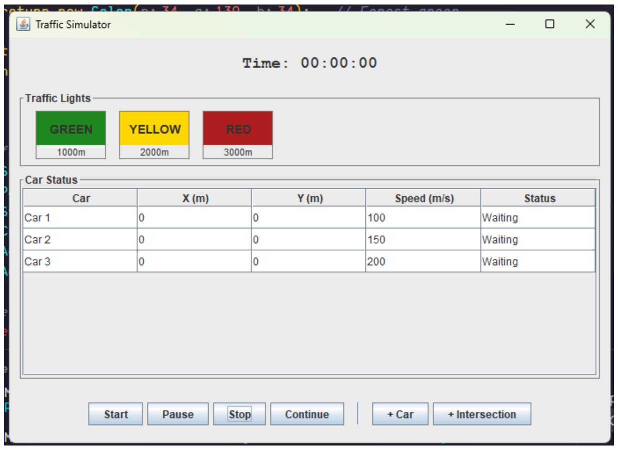
Actual Result: The “Stop” button does not reset the simulation when “Start” is pressed.

Pass/Fail: Fail



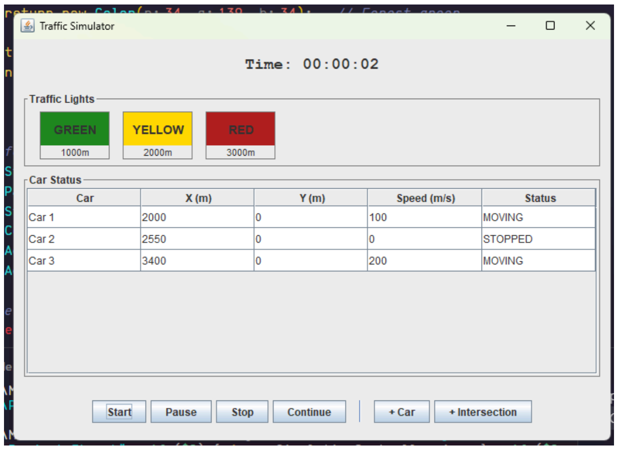
Time: 00:00:02

Car	X (m)	Y (m)	Speed (m/s)	Status
Car 1	200	0	100	MOVING
Car 2	300	0	150	MOVING
Car 3	400	0	200	MOVING



Time: 00:00:00

Car	X (m)	Y (m)	Speed (m/s)	Status
Car 1	0	0	100	Waiting
Car 2	0	0	150	Waiting
Car 3	0	0	200	Waiting



Time: 00:00:02

Car	X (m)	Y (m)	Speed (m/s)	Status
Car 1	2000	0	100	MOVING
Car 2	2550	0	0	STOPPED
Car 3	3400	0	200	MOVING

Test # 17

Expected Result: Buttons function as intended with CarThreads.

Actual Result: Buttons function as intended with CarThreads.

Pass/Fail: Pass

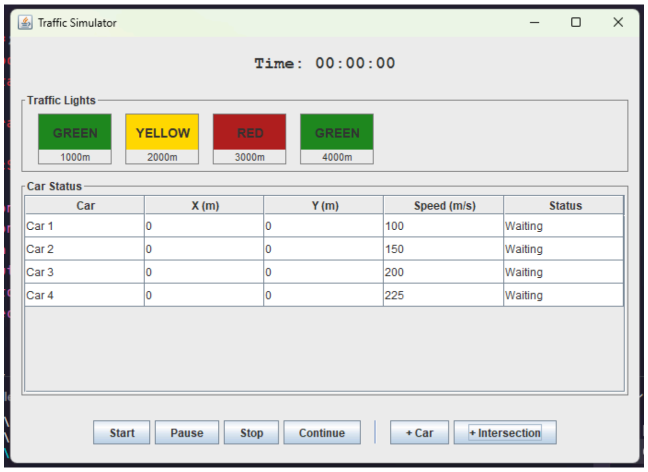
note: SS omitted, but "Stop" button functionality was fixed

Test # 18

Expected Result: Add car and intersection buttons function as intended.

Actual Result: Add car and intersection buttons function as intended.

Pass/Fail: Pass



Car	X (m)	Y (m)	Speed (m/s)	Status
Car 1	0	0	100	Waiting
Car 2	0	0	150	Waiting
Car 3	0	0	200	Waiting
Car 4	0	0	225	Waiting

Assumptions and Limitations

Assumptions

1. Y = 0 for all positions (straight road, no turns)
2. Cars stop instantly at red lights
3. Cars continue through yellow and green lights without slowing down
4. The distance between the lights is 1000 meters
5. Units are meters
6. Time increments in 1-second intervals
7. Traffic lights cycle: GREEN (10s) → YELLOW (3s) → RED (5s)
8. All cars start at x = 0

9. Cars cannot interact

Limitations

1. GUI display limit -> traffic lights can only display as many as the user's screen width
2. Lights complete their cycle after the "Pause" button is pressed
3. Fixed light durations
4. Infinite road
5. Same starting point for cars
6. Fixed speed calculations
7. No saving or loading of simulation states
8. Cannot remove cars

Lessons Learned

Project Planning & Organization

I went into this project less prepared than usual. Typically, I start by brainstorming how I want components to look and behave, but this time I dove into coding without a solid plan. This lack of organization messed up my workflow. You can see the impact in my test cases, where visual elements, like the traffic light designs and spacing, kept changing because I had not finalized them beforehand. If I had established clear guardrails and a visual design from the start, I could have avoided the confusion and reworking I faced.

Technical Challenge

I also gained a much deeper understanding of multithreading. Before this project, I felt confident in my theoretical knowledge of threads, but actually applying them was a challenge. I realized that simply launching threads is easy, but coordinating them is hard.

AI Statement

To complete this project, I used Cursor, which supported about 50% of the work. I mainly used the Cursor agent to clarify, contextualize, clean, and organize, while sometimes having it correct, implement, and debug my code. I am satisfied that I feel AI supported less of my efforts than the last project; however, I still aim to reduce this usage overall when tackling new concepts.

Although these tools contributed more than I initially intended, they still challenged me to grasp each implementation fully. While their assistance made the process faster and more manageable, I ultimately had to ensure that all code was clean, functional, and aligned with the project's requirements.

Prompts:

"Do not code, I just want you to understand that in this folder, I will be completing this project."

"Right now, I just have a blank frame. Create a checklist of what else needs to be added to the GUI before functional implementation."

"Implement the rest of the GUI layout and adjust according to the sample you created."

"Okay, let's start implementing on the Car. I do not want you to code, but I want you to walk me through the implementation."

"Check what I have completed so far."

"Check my progress so far."

"How would I test if the thread works?"

"Check over my logic, it does not seem right."

"Before we move on, let's go back into the GUI and change the images to text of what color the light is when initialized."

"Before we move on, I need some clarity. The light cycling, for some reason, the light at 3000m does not cycle until the first light has cycled. They are just showing some weird behavior."

"Now what is left to implement?"

"Just scan over the code, checking formatting and redundancy, making sure things are neat and clean."

"Compile all of my prompts into a single response, fix punctuation and spelling, and put them in air quotes for easy copy and paste."