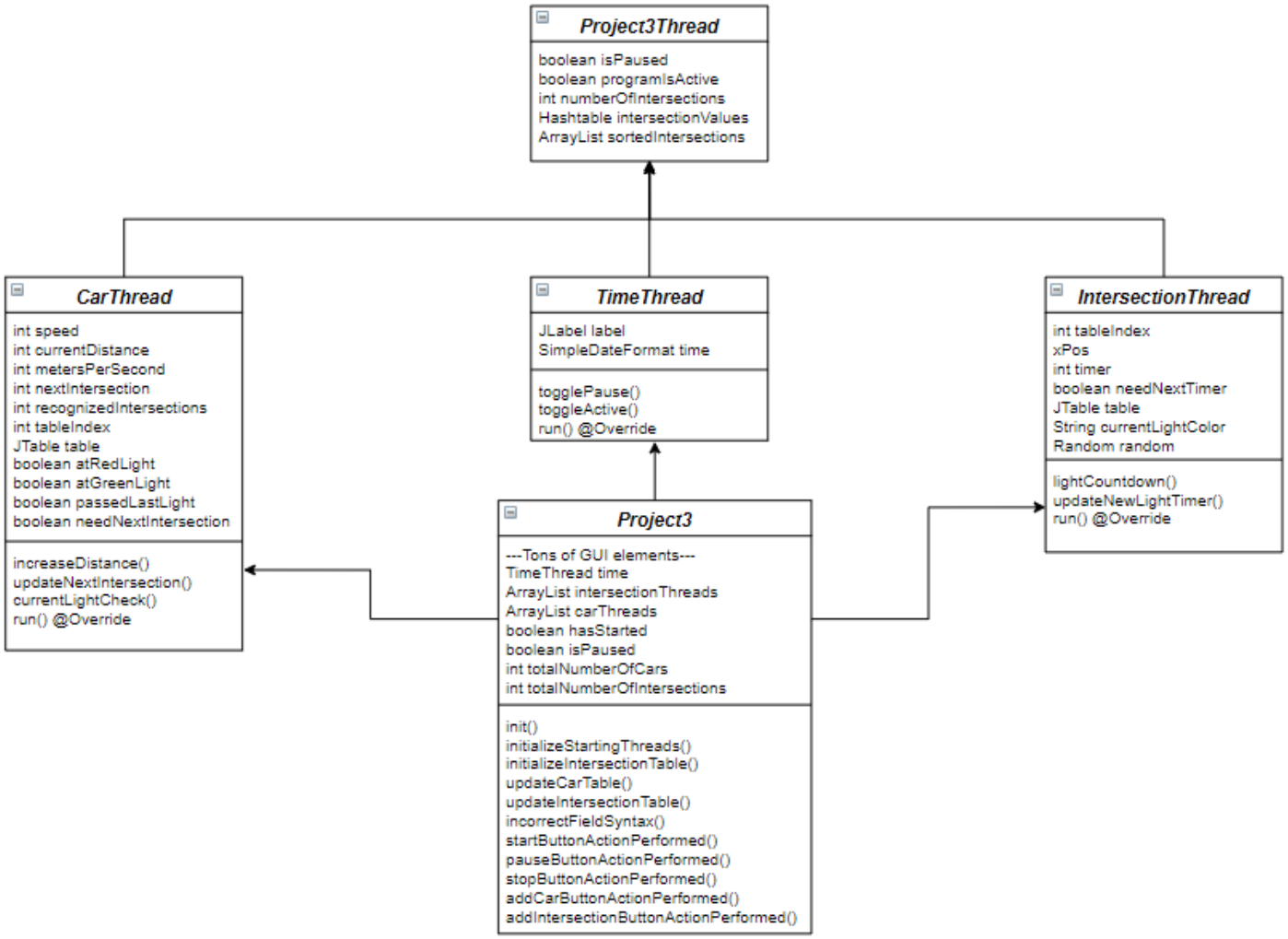

Figure 1. UML diagram of class relationships

Lessons learned:

I learned a ton about threads and how much can be done with multithreading. Originally, instead of using Booleans to determine when a thread should be "running" I used the deprecated methods suspend(), resume(), and stop(). Implementation was much easier with the deprecated methods, but they are deprecated for a reason. And while they were likely fine for this project, I felt it was better practice to not depend on them since who knows when they will go away. Booleans were used (likely in excess) for many facets of the program and to dictate much of the functionality. The most difficulty I encountered was likely configuring how to update the next intersection. I ended up using an array list with an int to check if the cars number of recognized intersections reflect that of the actual intersections (aka a new intersection was created). Once the car passes it's final intersection it will stop. One quality of life change I thought about was automatically reordering the intersection table based on distance. In other words, if an intersection at 500m was added, it would take the place of intersection of 1000m and move the 1000m and 1500m intersecions down one row. However, I deemed that project not worth the investment/complication so if an intersection is added in between intersections the table won't reflect that.

The time thread acts as the "master" thread. In other words, it toggles the Booleans (isActive and isPaused) in Project3Thread which affects CarThread, IntersectionThread, and itself. Each thread makes use of wait() to

some effect or another. Time uses it to only increment in seconds. Car uses it to determine how often to update current distance in its table. Intersection uses it to count down the light timers. Red and green light timers are randomly generated on each change (unique to each intersection) this means that the intersections will not always have the same light and adds some randomness.

User Guide:

Run Project3.java
Click start to begin the simulation.
Cars/Intersections can be added at any time (before starting or after).
Click pause to pause/resume to resume.
Once the simulation has stopped the GUI elements will change to reflect the fact.

Test cases:

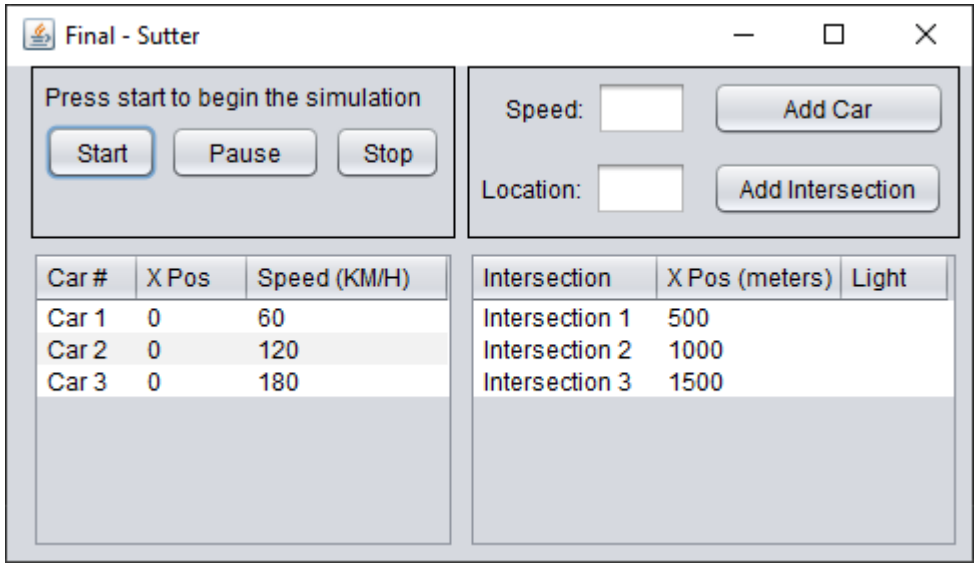

Figure 2. Showing the default state of the GUI

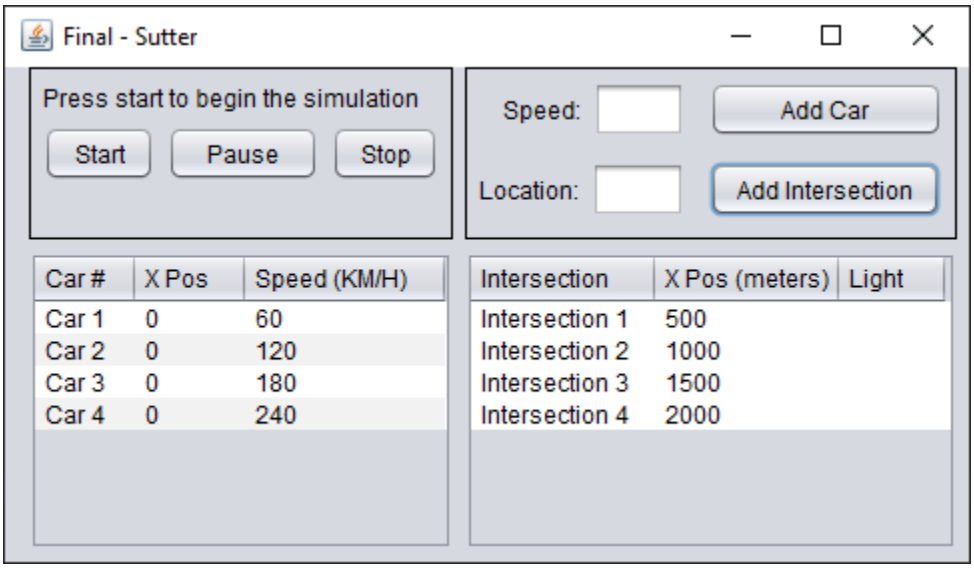

Figure 3. Test case #1. Adding cars and intersections before start button has been pressed.

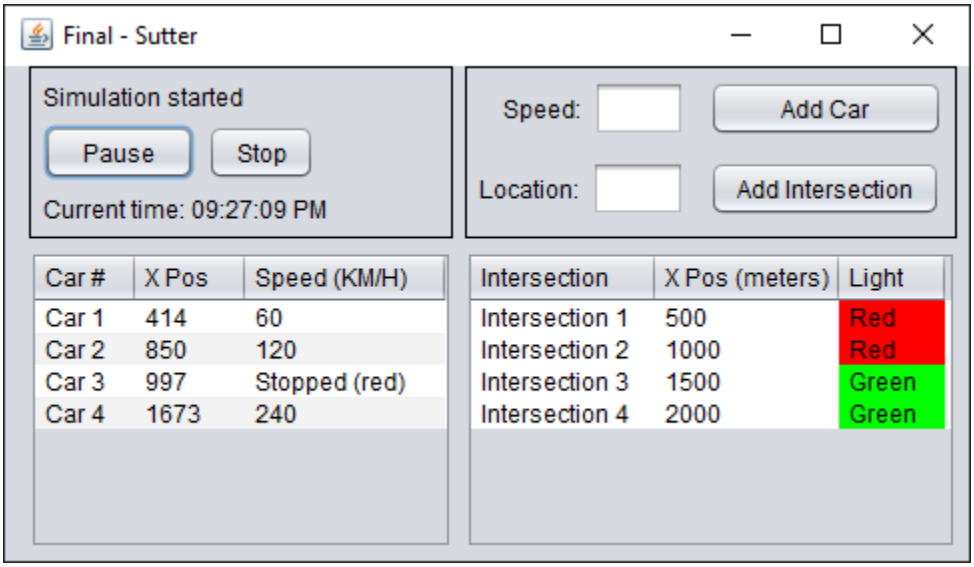

Figure 4. Test case #2. Start button pressed. Showing how lights have different random timers and a car stopped at a red light. Start button also disappears so user can't press it again.

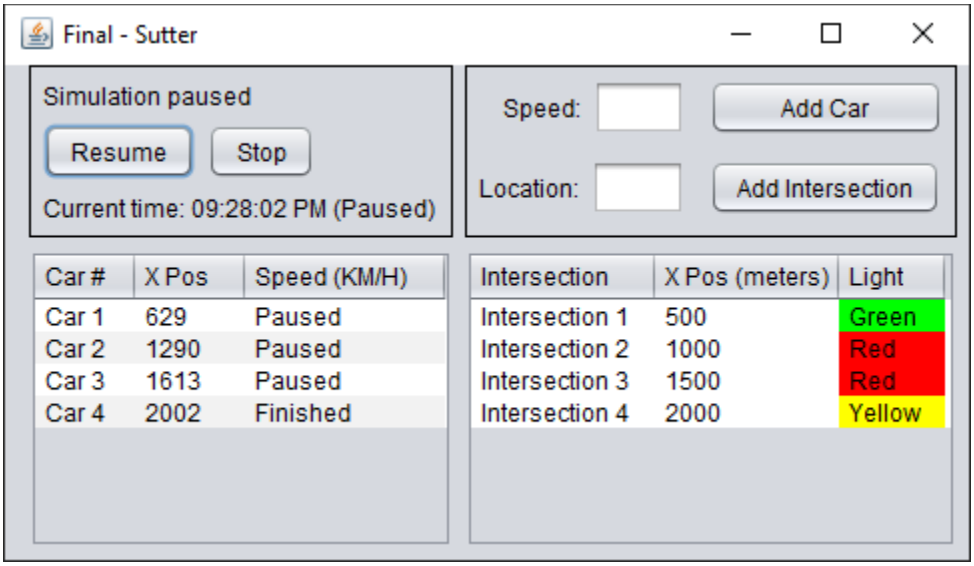

Figure 5. Test case #3. Pause button pressed. Car speed is overridden, but finished ones continue to say finished. Time also pauses and pause button changes to resume button. Also note "simulation paused".

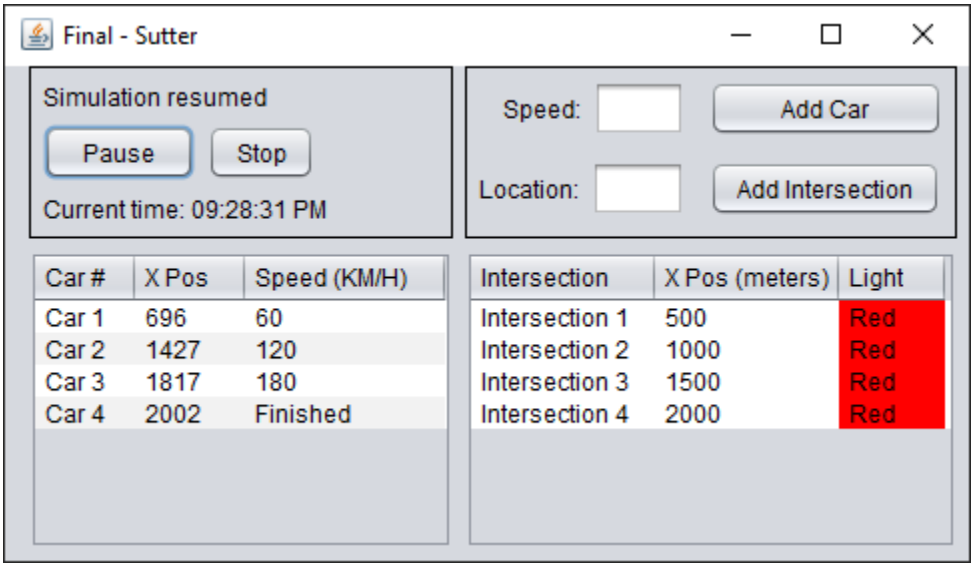

Figure 6. Test case #4. Resuming the simulation.

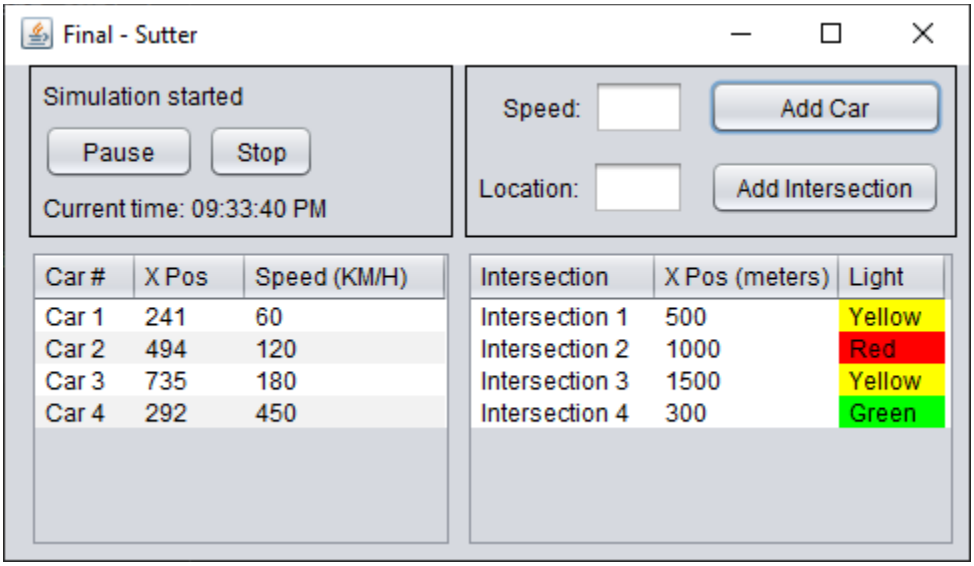

Figure 7. Test case #5. Adding a car mid simulation (note high speed and yet it is behind those that already started).

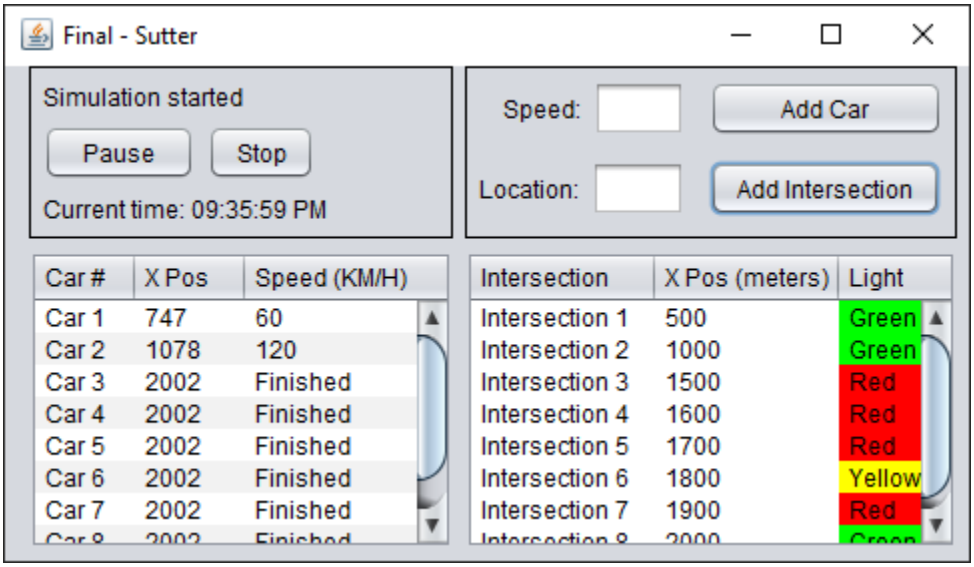

Figure 8. Test case #6. Adding tons of cars and intersections.

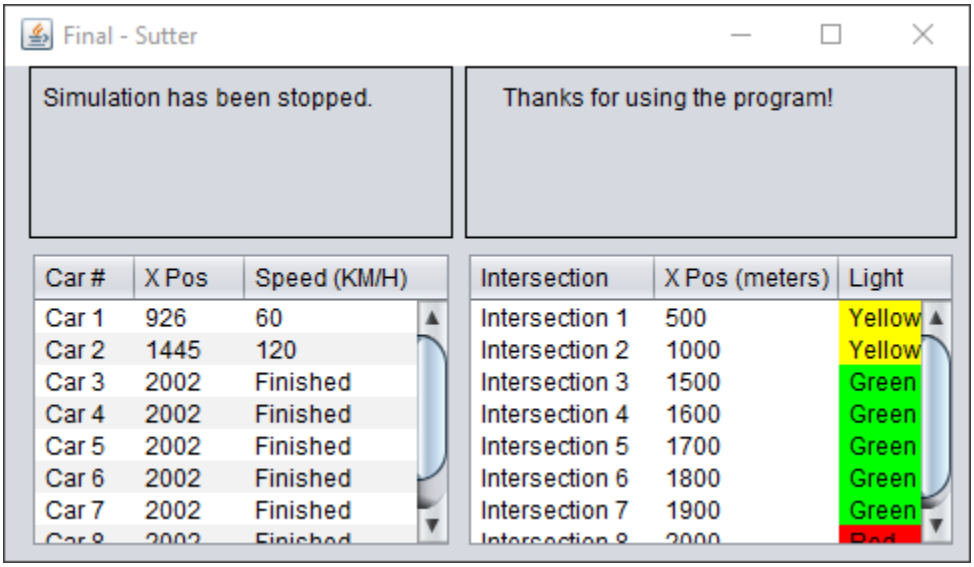

Figure 9. Test case #7. Stopping the simulation. Note text changes. Because this is a screenshot, it might not actually appear to have stopped but running the simulation will prove it does.

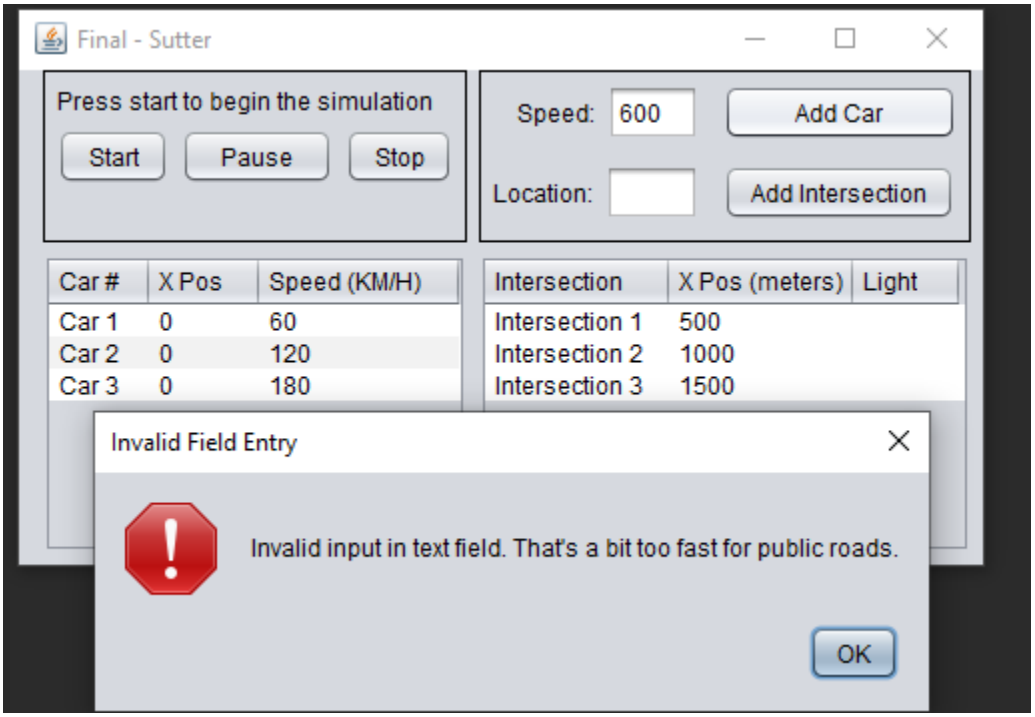

Figure 10. Test case #8. Trying to add a very fast car (the fastest a car can be in the simulation is 500km/h).

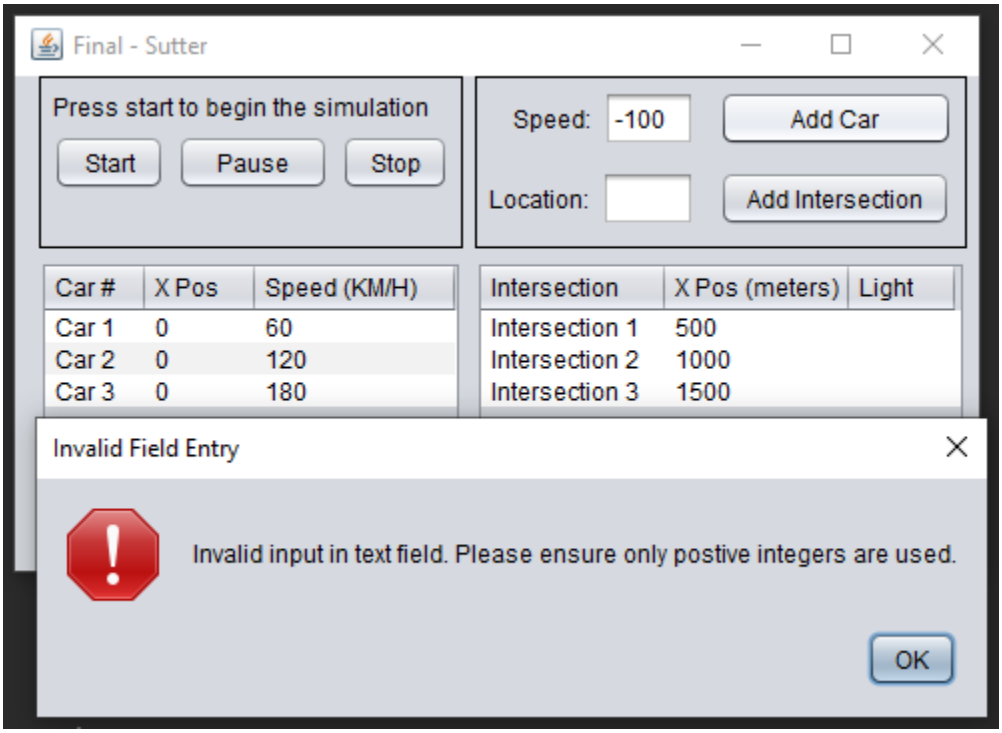

Figure 11. Test case #9. More invalid input, negative numbers. Neither text field will allow them.

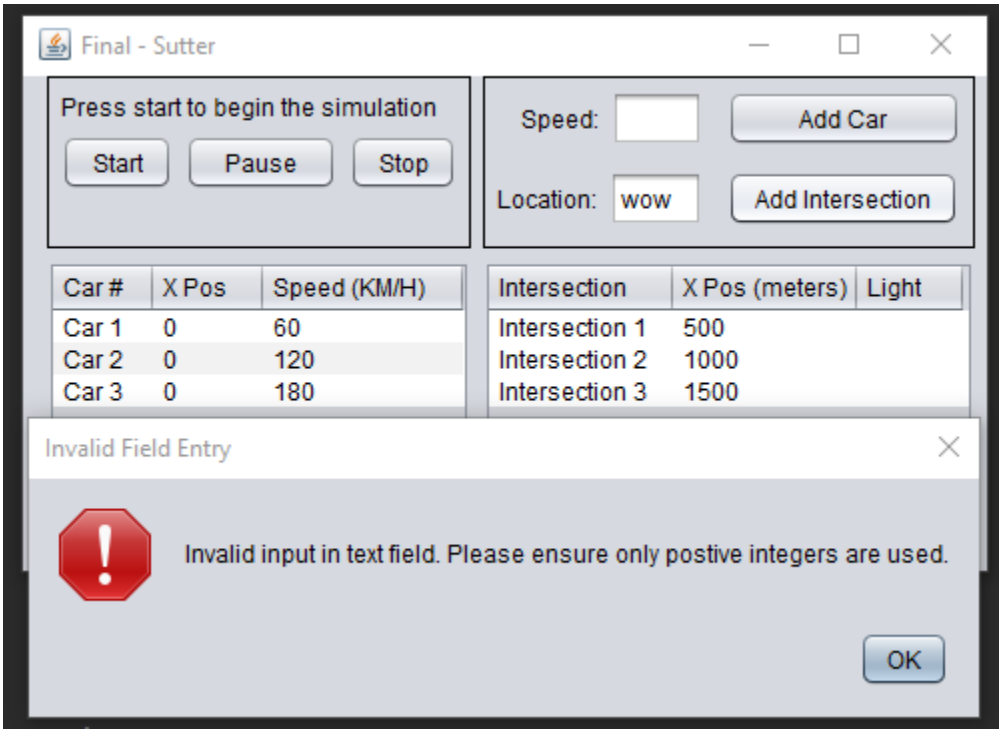

Figure 12. Test case #10. More invalid input, strings. Neither text field will allow them.

Given the nature of the program, it is difficult to represent in screenshots. But I believe I did my best to adequately cover all aspects of the program. One small thing I would like to point out is what occurs in figures 5 and 6. It may appear that after resuming the simulation all of the lights jumped to red. However, it was poor timing. Each light has its own timer that is broken up and counted down by half seconds.