Week 8: Project 3

John Kucera

Prof. Didier Vergamini

CMSC 335 Object-Oriented and Concurrent Programming

12 December 2020

Week 8: Project 3 Documentation & Solution Description

# Traffic Simulation: Threads + GUI

**Assumptions, Limitations & Design decisions:**

- **The simulation begins with nothing but an empty control frame. When Start button is pressed, 3 Traffic Lights and 3 Cars at random speeds from 1 to 100 KPH are generated. The visual display is then made visible and has live updates.** When a Traffic Light is added, it goes 1000m after the furthest light. There can be a total of 5 Traffic Lights and 5 Cars present in the simulation. I would like to have allowed more, but I also wanted to keep the visual display simple.
- **Visual display is updated live for anything that happens in the control frame: car movement, light color changing, car/light additions, pause, continue, and stop.**
- Assumes the same details in the guidelines: ability to start, pause, stop, continue, add car, add light, 1000m between each light, cars travel in straight line, cars pass through green/yellow lights and stop instantly at red lights.
- I chose to have multiple roads, one for each car, just so I can more easily see each car separately. On a single road, I can barely tell where the 3 cars are at the start when they are all so close together, and they would technically be colliding. They all still travel in

straight lines, and the cars' positions are all Y=0 in relation to their own roads. The traffic lights still apply to each road.

- When a car reaches the end of the road (X=5000m), they loop back to the beginning at 0m and continue as normal.
- **Red lights stay for 8 seconds, Green lights for 10 seconds, and Yellow lights for 2 seconds.**
- Car speeds have a range of 1 to 100 KPH for the sake of realism, as this is a simulation. As such, they'll often take around a minute (on average) to reach the first traffic light.
- Converting Speed in Kilometers per Hour to Distance in Meters:
    - For 1 KM/1 HR = 5M/18000 Milliseconds = 1M/3600 Milliseconds
    - **Car thread sleeps for 10 milliseconds**. 1/360M traveled per 10 Milliseconds
    - **[any speed] * 1/360M = distance traveled in 10 MS for a car at [any speed]**
- Converting 5000m road to 1150-pixel line in the simulation:
    - Ratio: carXCoordinate (Meters)/5000 = carXCoordinate (Pixels)/1150
    - **carXCoordinate (Pixels) = 1150*(carXCoordinate (Meters)/5000)**
- When Stop button is pressed, all threads finish run and are terminated. The information stays in the GUI and visual display, but they no longer update.
- When pressing Pause and then pressing Continue, the light will ALWAYS finish sleeping for the remaining time the color had. For example, Green lights sleep for 10 seconds. Let's say the light is Green for 4 seconds, then Pause is pressed. Any amount of time can pass, and then when Continue is pressed, the light will stay Green for 6 seconds before switching to Yellow.
- Program ends if CLOSE is clicked on EITHER the control frame or the visual display.

Week 8: Project 3

**UML Class diagram:** SimulationGUI class contains inner classes TimeStamp, TrafficLight, Car, and DrawVisual. See Fig. 1.

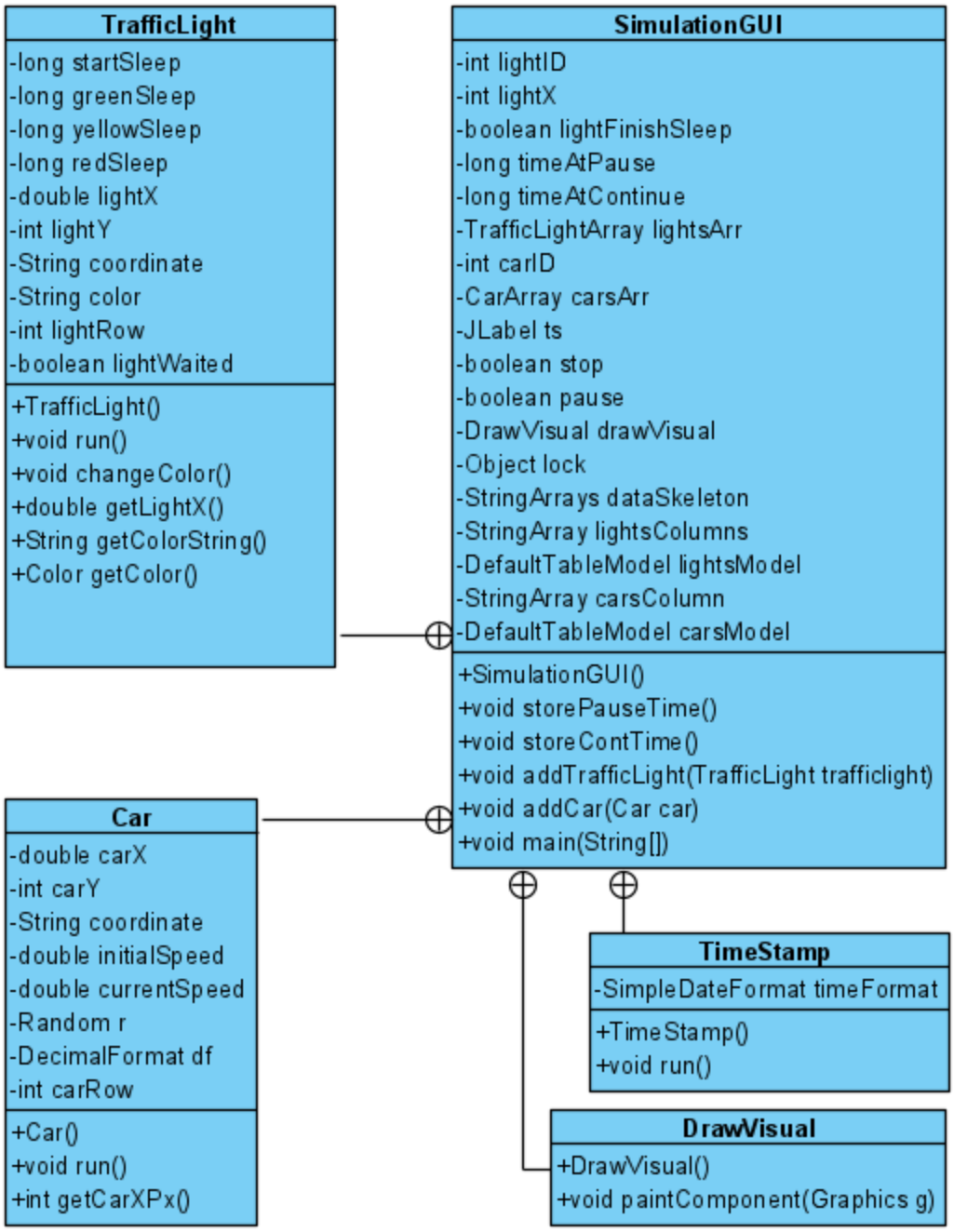



Figure 1: Project 3 UML Class diagram. (Kucera, 2020)

Week 8: Project 3

**User's Guide: How to set up and run this application**

1. With a software tool that can manage .zip and .rar files such as WinRAR, unzip my submitted zip file **JohnKuceraProject3.zip**. You can do this easily by right-clicking **JohnKuceraProject3.zip** and clicking **Extract Files**, then click **OK** (See Fig. 2). This gives you a readable folder with the application files inside (See Fig. 3).

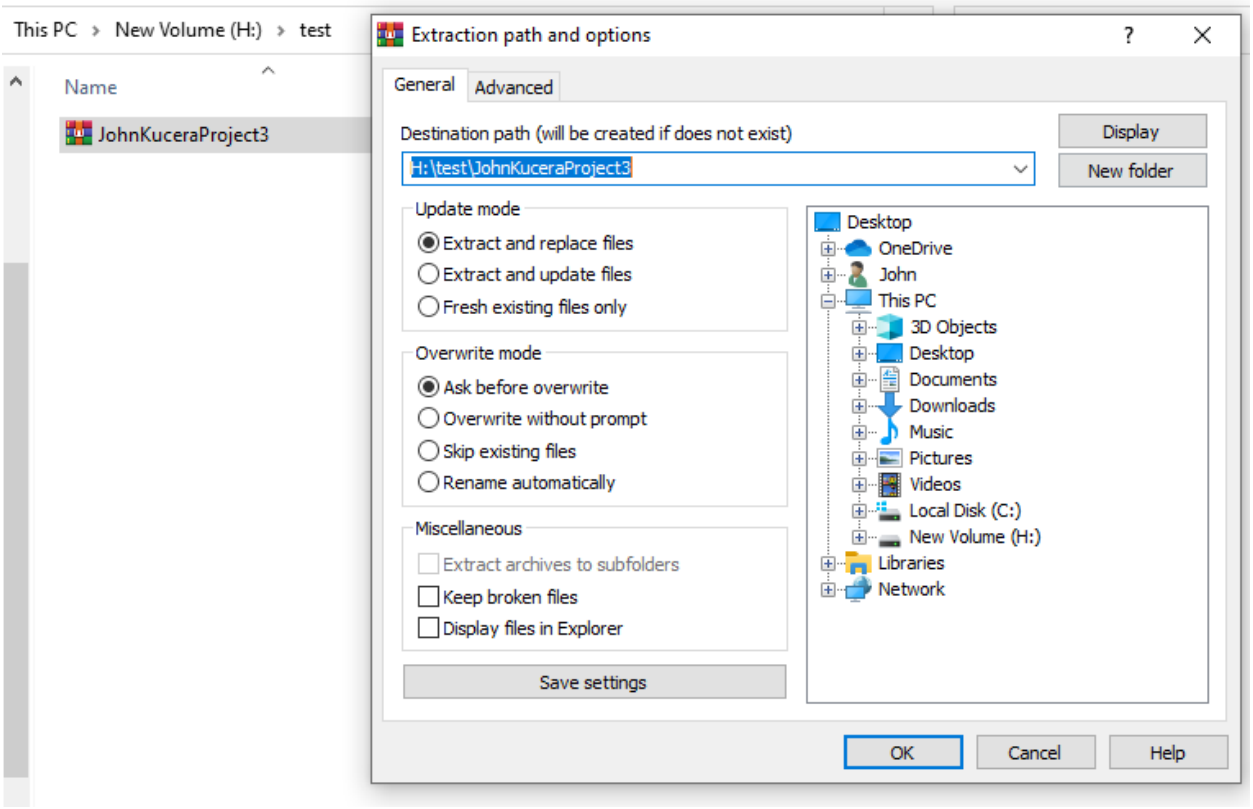



Figure 2: Unzipping a .zip file. (Kucera, 2020)

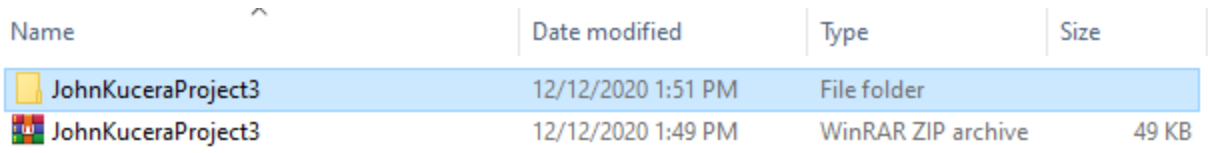



Figure 3: .zip file has been unzipped. (Kucera, 2020)

Week 8: Project 3

2. Open your IDE and create a new project (any IDE will work). Select Java Application (See Fig. 4 for example in Netbeans IDE).

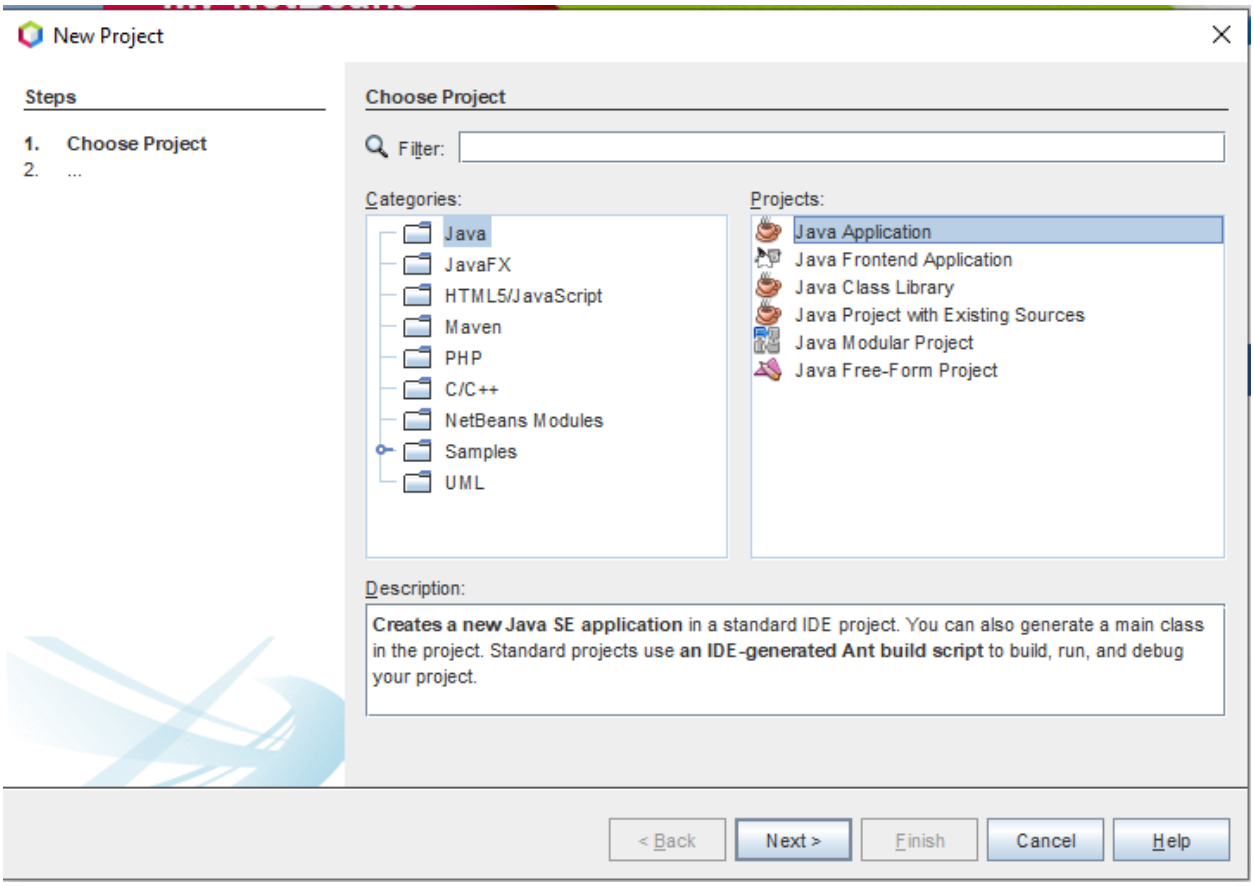



Figure 4: Creating Java Application Project in Netbeans. (Kucera, 2020)

Week 8: Project 3

3. (See Fig. 5) Name the project "**Test Kucera Project 3**". Identify the project location (which is where the application files will be saved). **DO NOT allow the IDE to automatically create main class**.

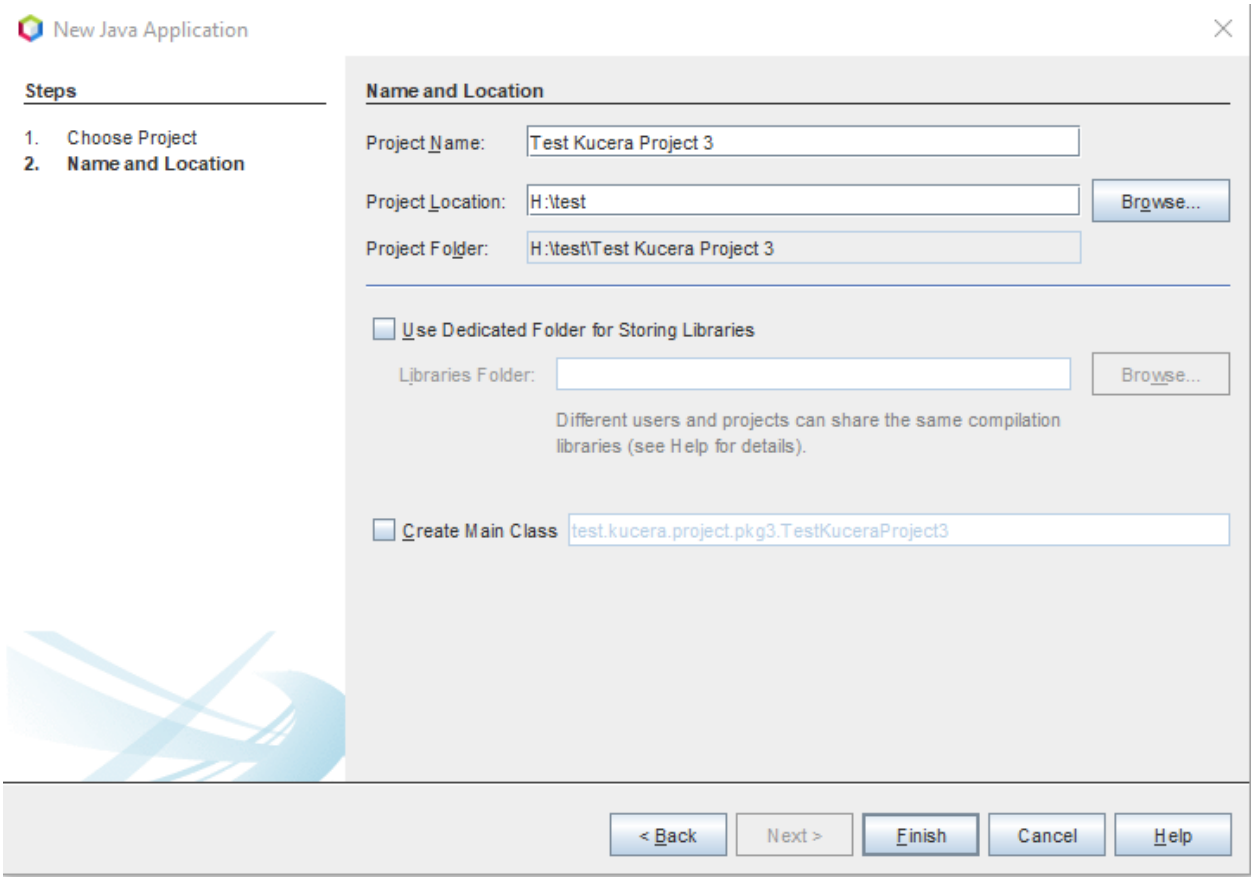



Figure 5: Creating Test Kucera Project 3. (Kucera, 2020)

4. In your File Explorer, go to the unzipped **JohnKuceraProject3 > src** folder and copy the .java source file **SimulationGUI.java** (See Fig. 6).

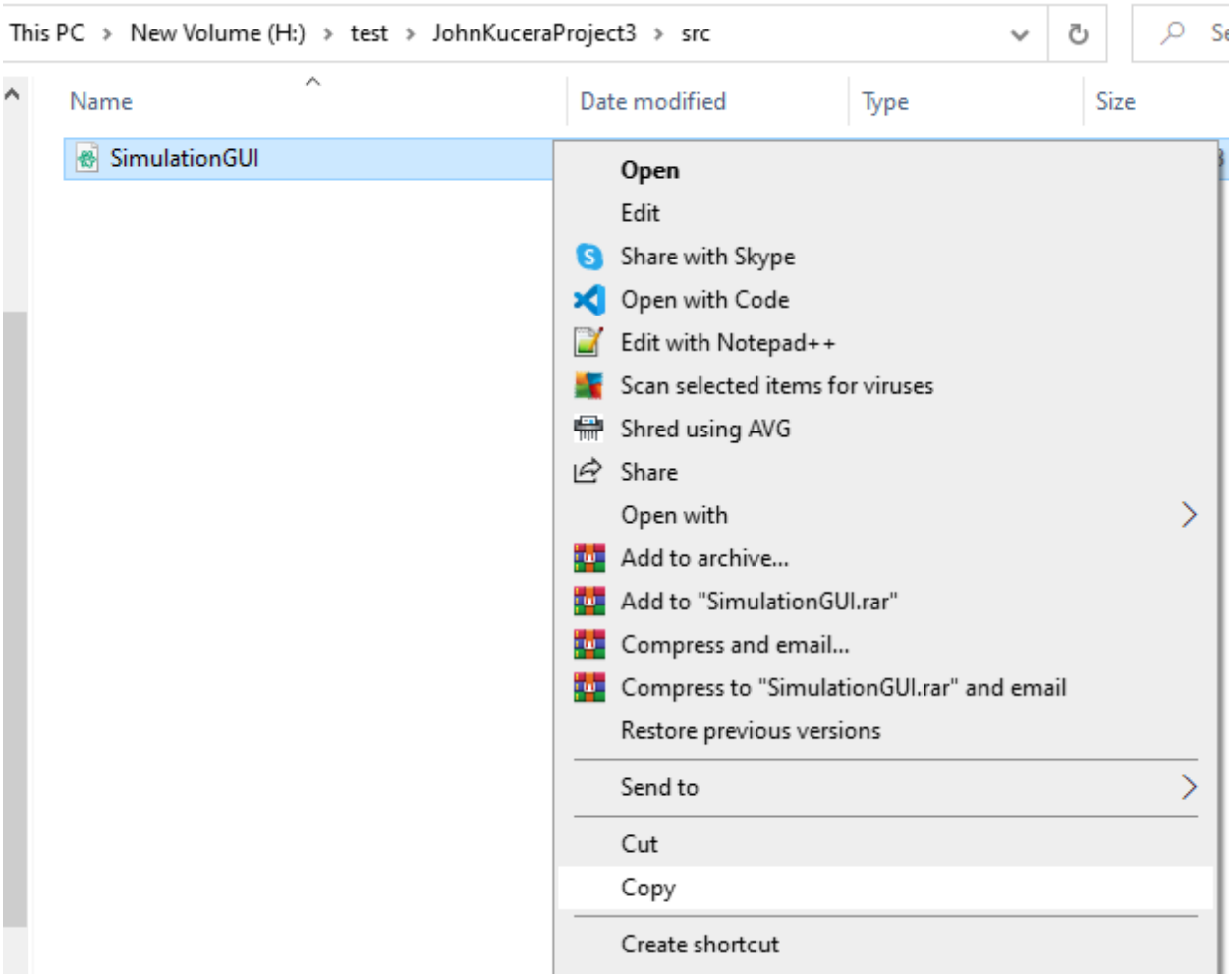



Figure 6: Copying Application Source Files. (Kucera, 2020)

5. Paste them into the **src** folder in YOUR project folder: **Test Kucera Project 3 > src** (See Fig. 7). They will appear in your IDE under the new project's Source Packages (See Fig. 8).

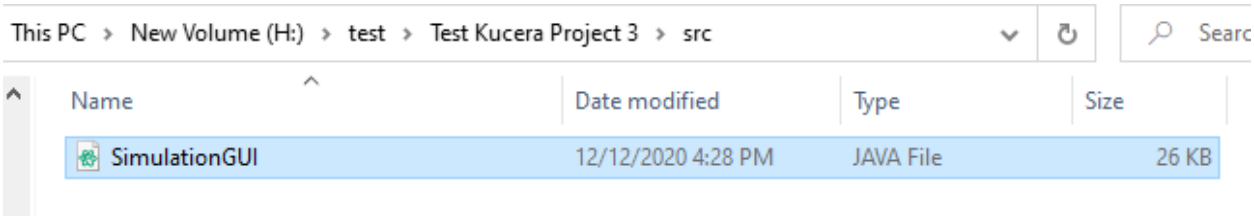



Figure 7: Pasting Application Source Files in Test Project. (Kucera, 2020)

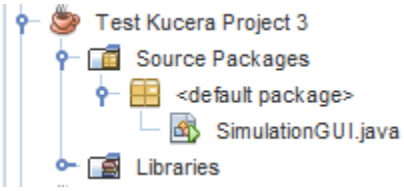



Figure 8: Pasted Source Files appear in IDE. (Kucera, 2020)

Week 8: Project 3

6. In the IDE, open the **Project Properties** of **Test Kucera Project 3**. You can do this by right-clicking **Test Kucera Project 3** and clicking **Properties**. In the "Run" options, change the Main Class to "**SimulationGUI**", since the Main Method for this application is in **Simulation.java** (See Fig. 9).

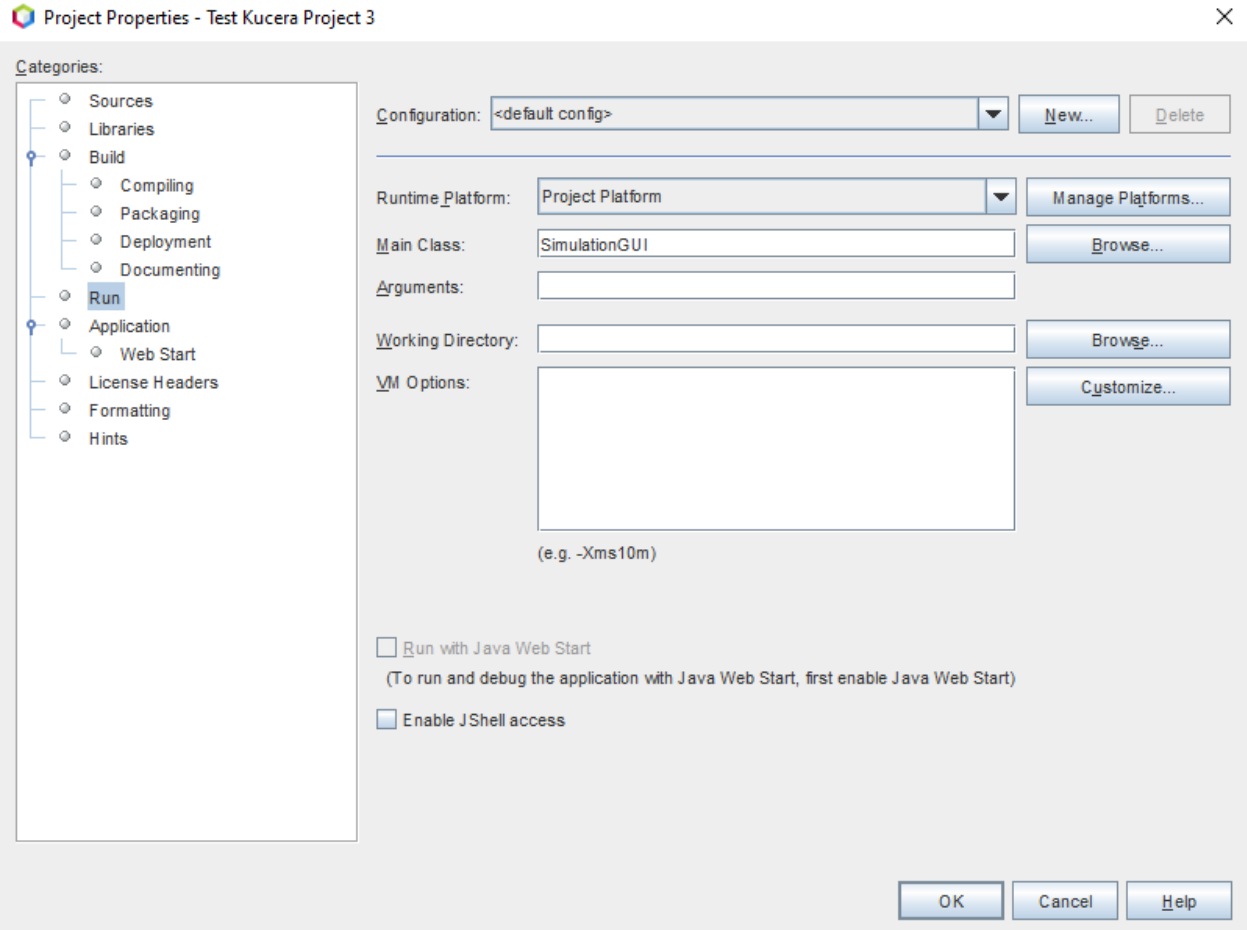
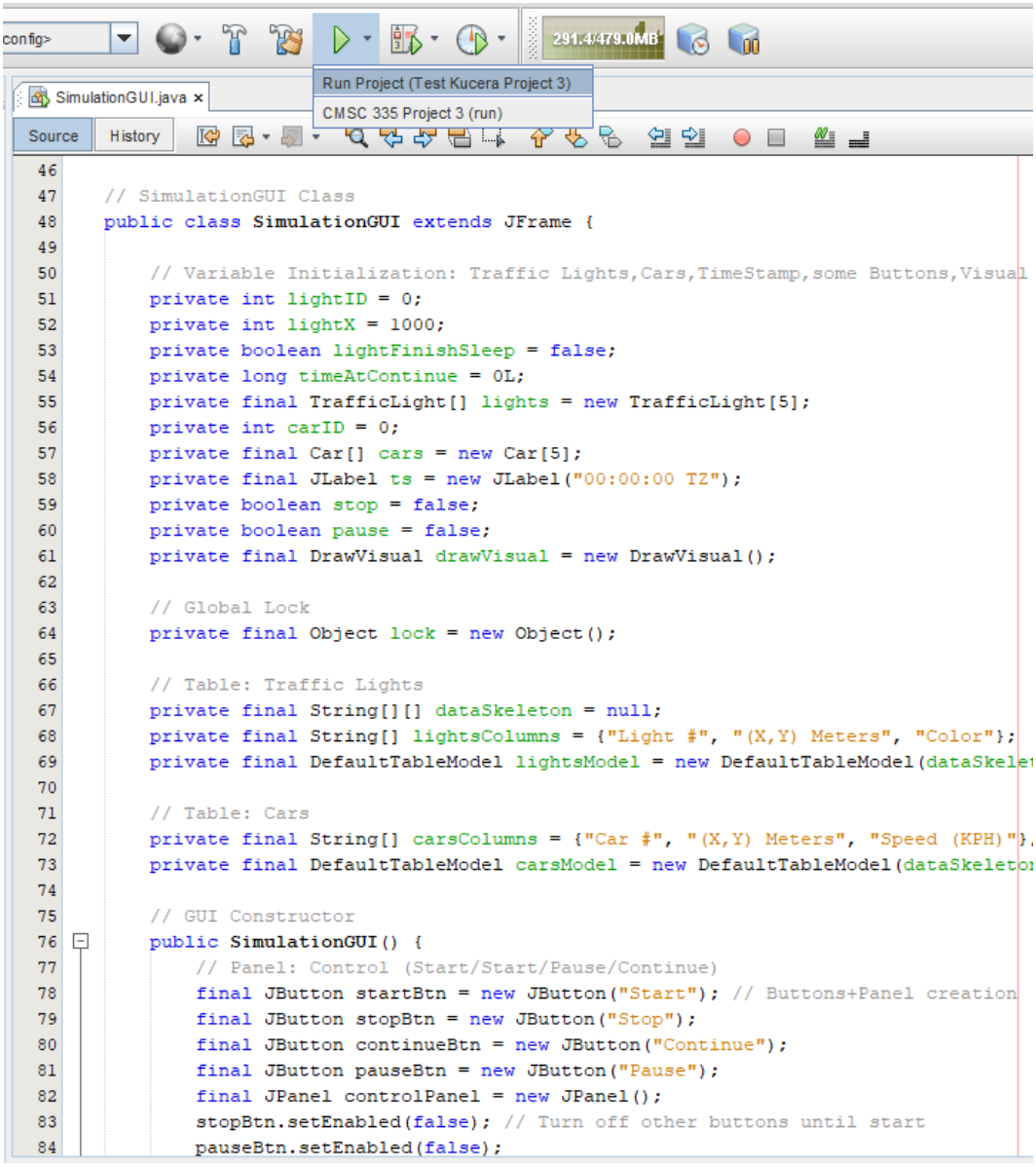



Figure 9: Project Properties > Run, type in Main Class. (Kucera, 2020)

Week 8: Project 3

7. You can click on each source file to open them and view the Java code in each. To run the application, make sure **SimulationGUI.java** is open and click **Run Project** (See Fig. 10).




```
46
47    // SimulationGUI Class
48    public class SimulationGUI extends JFrame {
49
50        // Variable Initialization: Traffic Lights,Cars,TimeStamp,some Buttons,Visual
51        private int lightID = 0;
52        private int lightX = 1000;
53        private boolean lightFinishSleep = false;
54        private long timeAtContinue = 0L;
55        private final TrafficLight[] lights = new TrafficLight[5];
56        private int carID = 0;
57        private final Car[] cars = new Car[5];
58        private final JLabel ts = new JLabel("00:00:00 TZ");
59        private boolean stop = false;
60        private boolean pause = false;
61        private final DrawVisual drawVisual = new DrawVisual();
62
63        // Global Lock
64        private final Object lock = new Object();
65
66        // Table: Traffic Lights
67        private final String[][] dataSkeleton = null;
68        private final String[] lightsColumns = {"Light #", "(X,Y) Meters", "Color"};
69        private final DefaultTableModel lightsModel = new DefaultTableModel(dataSkele
70
71        // Table: Cars
72        private final String[] carsColumns = {"Car #", "(X,Y) Meters", "Speed (KPH)"},
73        private final DefaultTableModel carsModel = new DefaultTableModel(dataSkeleto
74
75        // GUI Constructor
76        public SimulationGUI() {
77            // Panel: Control (Start/Start/Pause/Continue)
78            final JButton startBtn = new JButton("Start"); // Buttons+Panel creation
79            final JButton stopBtn = new JButton("Stop");
80            final JButton continueBtn = new JButton("Continue");
81            final JButton pauseBtn = new JButton("Pause");
82            final JPanel controlPanel = new JPanel();
83            stopBtn.setEnabled(false); // Turn off other buttons until start
84            pauseBtn.setEnabled(false);
```

Figure 10: Open SimulationGUI.java, click Run Project. (Kucera, 2020)

Week 8: Project 3

8. The program will be displayed in the form of a GUI JFrame. You can interact with this interface by clicking the buttons. (See Fig. 11).

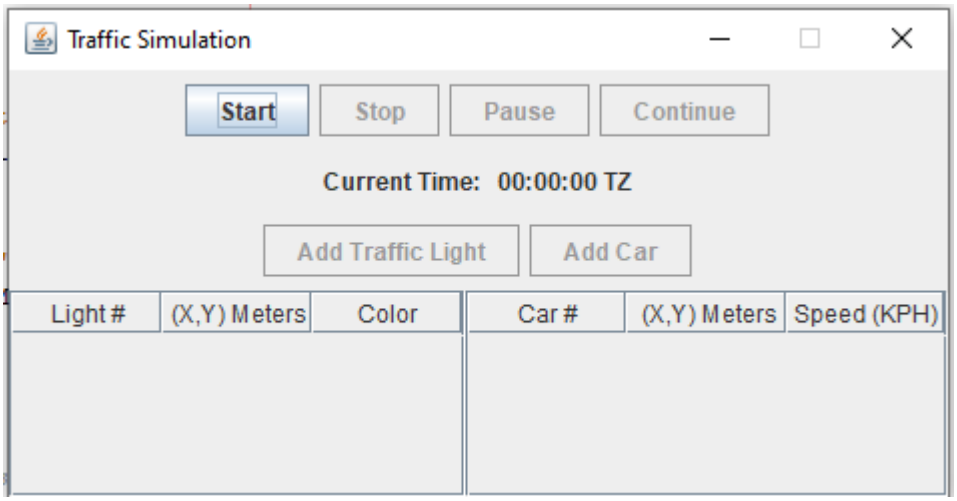



Figure 11: Run project, GUI displayed. (Kucera, 2020)

Week 8: Project 3

**9.** User input is allowed in the GUI as the application runs. Try testing the application by performing the Test Cases shown below.

**Test Cases (Summary)**

Test cases are numbered and are organized according to the component they test.

| **Aspect Tested** | **Test Case #** |
|---|---|
| "Start" button, cars/lights/timestamp are live updated and properly generated, visual display frame is live updated | 1 |
| Program exits when clicking close | 2, 3 |
| Timestamp | 4 |
| "Add Car", "Add Traffic Light" buttons | 5, 6 |
| "Pause", "Continue" buttons | 7-9 |
| Lights changing colors | 10-12 |
| Cars driving through Green/Yellow lights and stopping at Red lights | 13-16 |
| Car goes back to 0m after reaching 5000m | 17 |
| "Stop" button | 18 |
| Results: **18 out of 18 Test Cases PASSED. The program is successful.** | |

Week 8: Project 3

## Test Cases

1. **Aspect Tested:** Start button and initial data

   **Input:** Press start button

   **Expected Output:** Stop/Pause/Add Light/Add Car buttons become enabled. The Timestamp reflects the real-world time. 3 Traffic Lights and 3 Cars are started and displayed in the GUI. The 3 Traffic Lights are 1000m apart and the 3 cars have randomly generated speed 1 to 100 KPH. As time passes, the data in the table updates with car coordinates and light color changing. The Visual Display frame is made visible and reflects the information in the control frame.

   **Actual Output**: See Fig. 12.

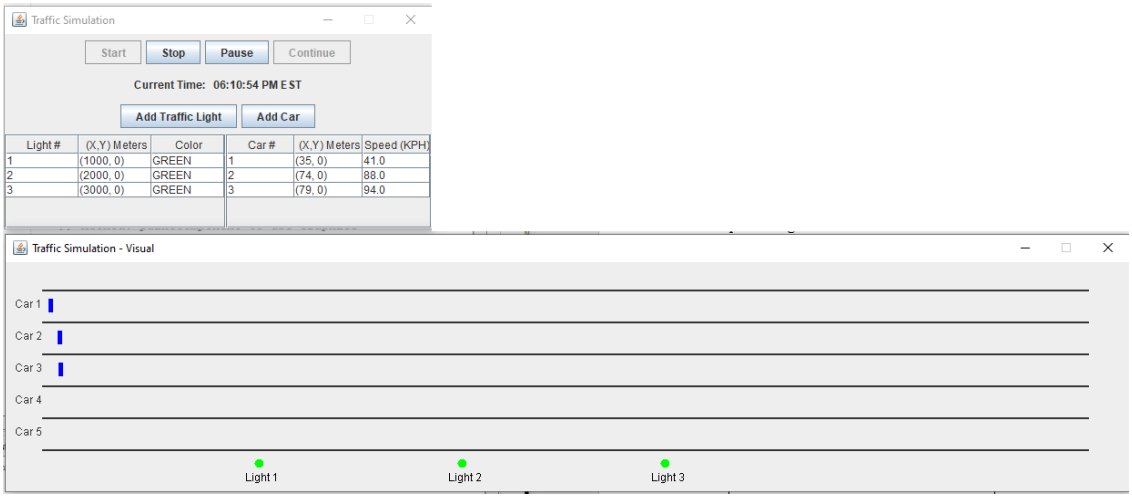



Figure 12: Test Case 1 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

2. **Aspect Tested:** Exit program on close (Control frame)

   **Input:** Click CLOSE button in top right corner of control J-frame

   **Expected Output:** Program exits

   **Actual Output**: See Fig. 13.

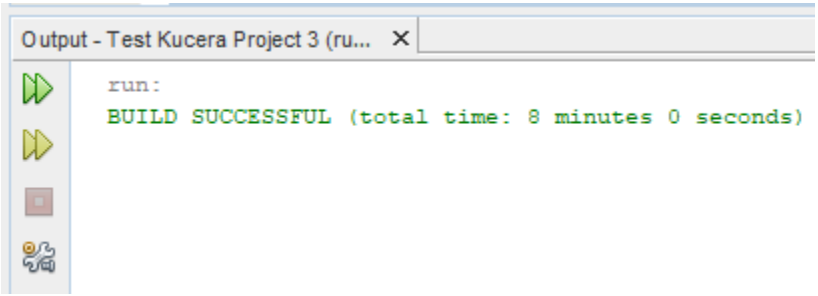



Figure 13: Test Case 2 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

3. **Aspect Tested:** Exit program on close (Visual Display frame)

   **Input:** Click CLOSE button in top right corner of visual display J-frame

   **Expected Output:** Program exits

   **Actual Output**: See Fig. 14.

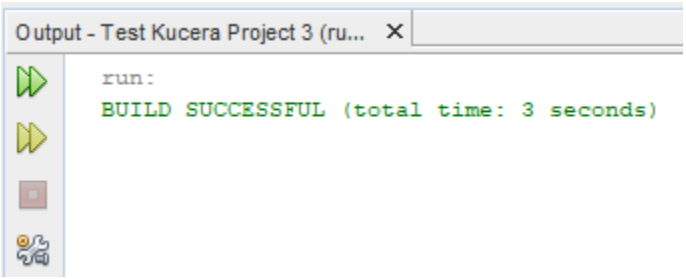



Figure 14: Test Case 3 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

4. **Aspect Tested:** Timestamp is updated every second with current time

   **Input:** N/A. Just watch the timestamp. (I used screen record for this)

   **Expected Output:** Timestamp label displays adds 1 second after 1 second passes in real time.

   **Actual Output**: See Fig. 15.


Figure 15: Test Case 4 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

5. **Aspect Tested:** “Add Traffic Light” button

   **Input:** Click “Add Traffic Light” button TWICE

   **Expected Output:** For each click on the button, 1 traffic light gets added to the simulation (I click twice so Lights 4 and 5 got added). They are immediately shown in both the control frame and the visual display frame. They are 1000m apart from each other. The button now reads “Max Traffic Lights Reached” since 5 is the limit, and the button is gets disabled for the rest of this program run.

   **Actual Output**: See Fig. 16.

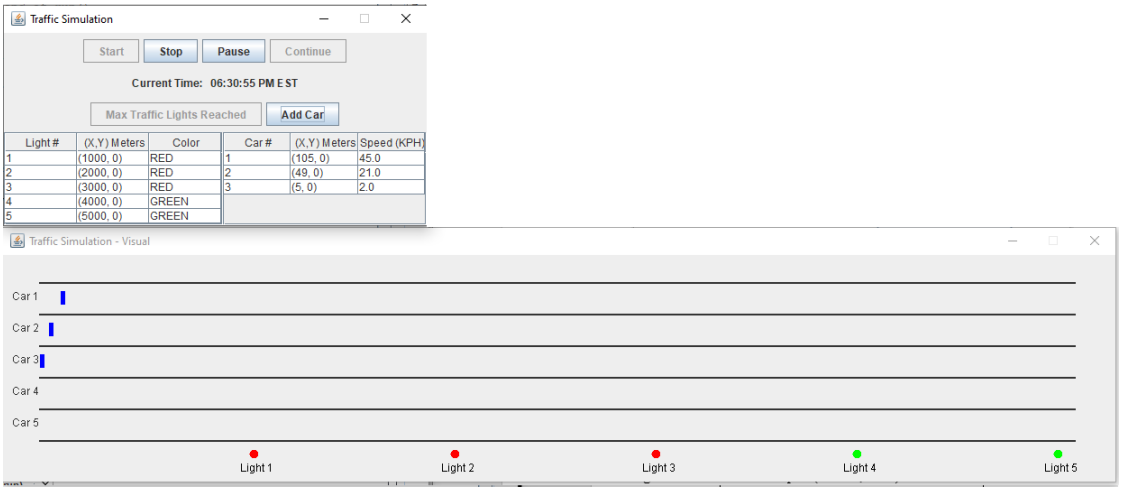



Figure 16: Test Case 5 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

6. **Aspect Tested:** “Add Car” button

   **Input:** Click “Add Car” button TWICE

   **Expected Output:** For each click on the button, 1 car gets added to the simulation (I click twice so Cars 4 and 5 get added). They are immediately shown in both the control frame and the visual display frame. They have randomly generated speeds 1 to 100 KPH. The button now reads “Max Cars Reached” since 5 is the limit, and the button is gets disabled for the rest of this program run.

   **Actual Output**: See Fig. 17.

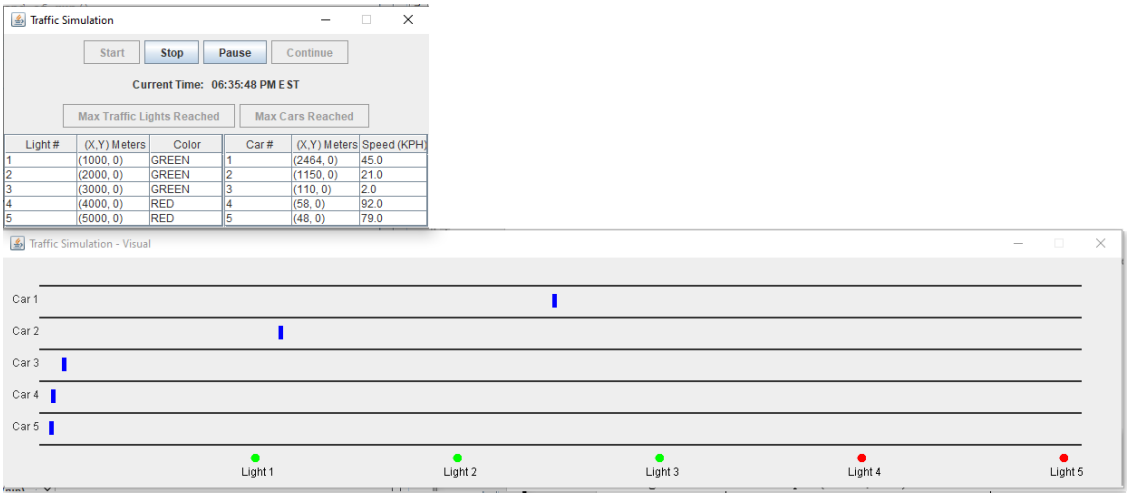



Figure 17: Test Case 6 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

7. **Aspect Tested:** Pause button

   **Input:** Click “Pause” button

   **Expected Output:** “Pause”, “Add Traffic Light”, “Add Car” buttons get disabled and “Continue” button is enabled. The lights stop changing colors, the cars stop moving, and both the control frame and visual display frame GUI reflect this. The TimeStamp also stops updating.

   **Actual Output**: See Fig. 18.

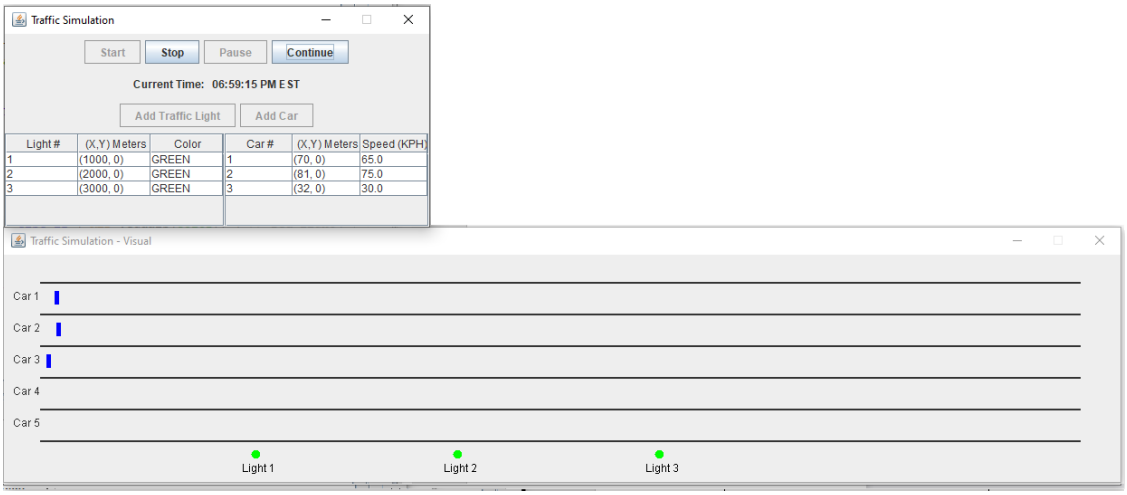



Figure 18: Test Case 7 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

8. **Aspect Tested:** Continue button

   **Input:** After 4 seconds of a Green light has passed (Lights #1-3), click “Pause” button. Then after 4 seconds of pause, click “Continue” button (I used screen recording for this)

   **Expected Output:** When “Continue” is clicked, the remaining 6 seconds of time passes before Green light switches to Yellow. “Continue” button is disabled. “Pause”, “Add Traffic Light”, “Add Car” buttons get enabled. The timestamp and lights and car information resume updating in both the control frame and in the visual display frame. The timestamp still updates the current time in real world.

(continued on next page)

Week 8: Project 3

**Actual Output**: See Fig. 19. First screencap is at Pause pressed, second screencap is when light has changed after Continue. Check timestamps to see 10 seconds between (4 seconds of pause + 6 seconds of resume sleeping).

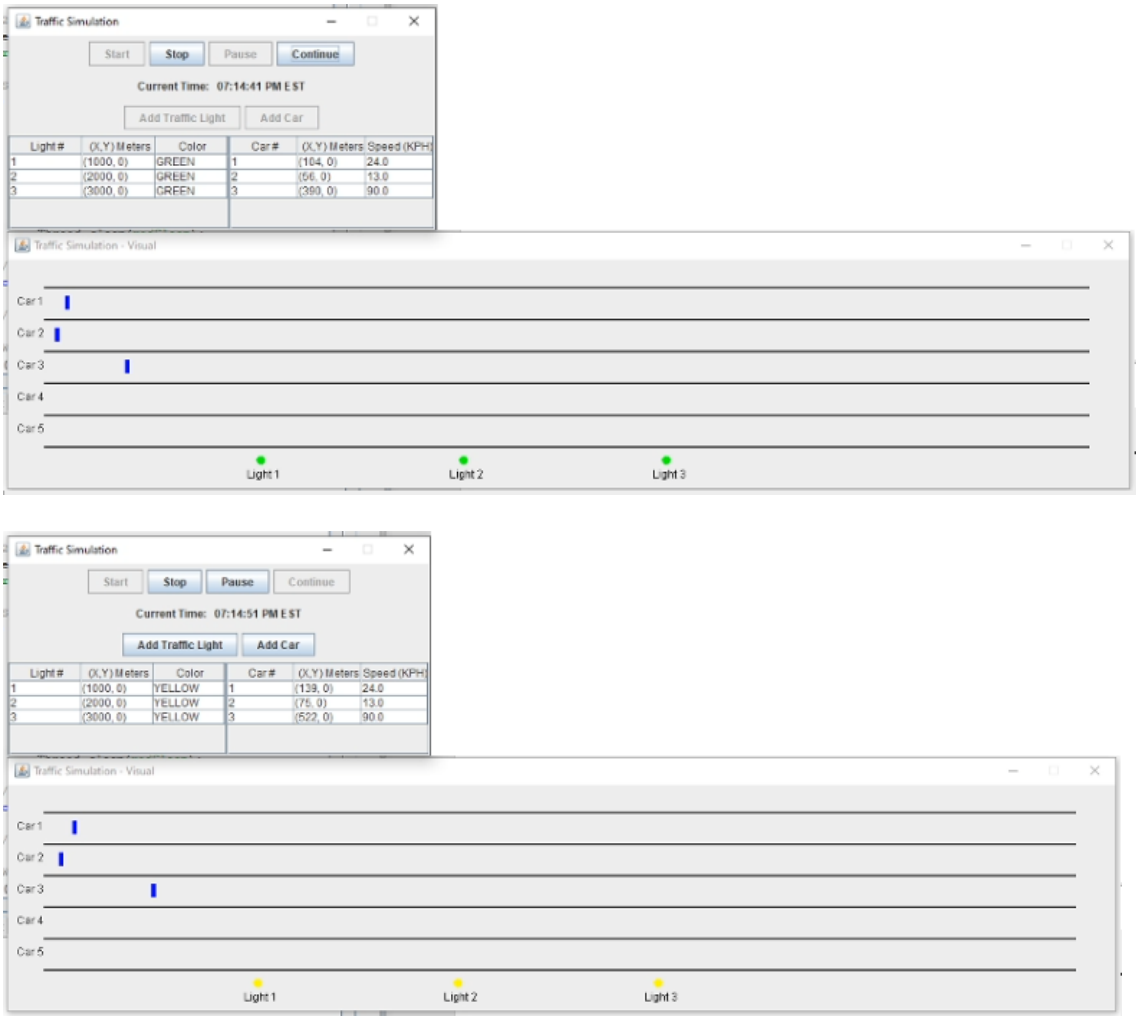



Figure 19: Test Case 8 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

9. **Aspect Tested:** Continue button

   **Input:** After 3 seconds of a Red light has passed (Lights #1-3), click “Pause” button. Then after 16 seconds of pause, click “Continue” button (I used screen recording for this)

   **Expected Output:** When “Continue” is clicked, the remaining 5 seconds of time passes before Red light switches to Green. “Continue” button is disabled. “Pause”, “Add Traffic Light”, “Add Car” buttons get enabled. The timestamp and lights and car information resume updating in both the control frame and in the visual display frame. The timestamp still updates the current time in real world.

   (continued on next page)

Week 8: Project 3

**Actual Output**: See Fig. 20. First screencap is at Pause pressed, second screencap is when light has changed after Continue. Check timestamps to see 21 seconds between (16 seconds of pause + 5 seconds of resume sleep).

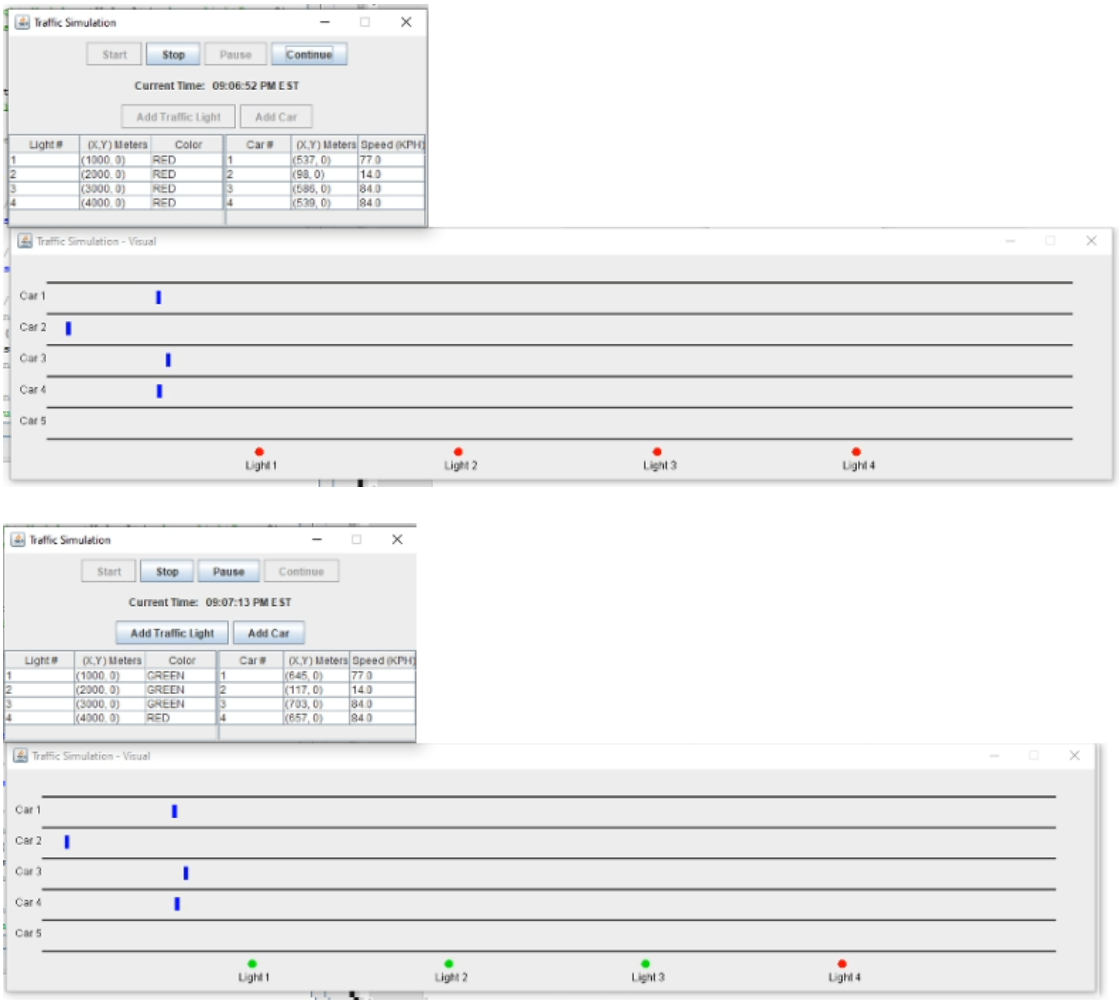



Figure 20: Test Case 9 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

10. **Aspect Tested:** Light change: Green to yellow after 10 seconds

    **Input:** N/A, just watching the colors change as time passes (Light #4)

    **Expected Output:** After 10 seconds, Light#4 has Green turn to Yellow

    **Actual Output**: See Fig. 21. First screencap is when light#4 turns green, second screencap is when light turns yellow. 10 seconds between the timestamps.


Figure 21: Test Case 10 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

11. **Aspect Tested:** Light change: yellow to red after 2 seconds

    **Input:** N/A, just watching the colors change as time passes (Light #5)

    **Expected Output:** After 2 seconds, Light#5 has Yellow turn to Red

    **Actual Output**: See Fig. 22. First screencap is when light#5 turns yellow, second screencap is when light turns red. 2 seconds between the timestamps.


Figure 22: Test Case 11 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

12. **Aspect Tested:** Light change: red to green after 8 seconds

**Input:** N/A, just watching the colors change as time passes (Light #2)

**Expected Output:** After 8 seconds, Light#2 has Red turn to Green

**Actual Output**: See Fig. 23. First screencap is when light#2 turns red, second screencap is when light turns green. 8 seconds between the timestamps.

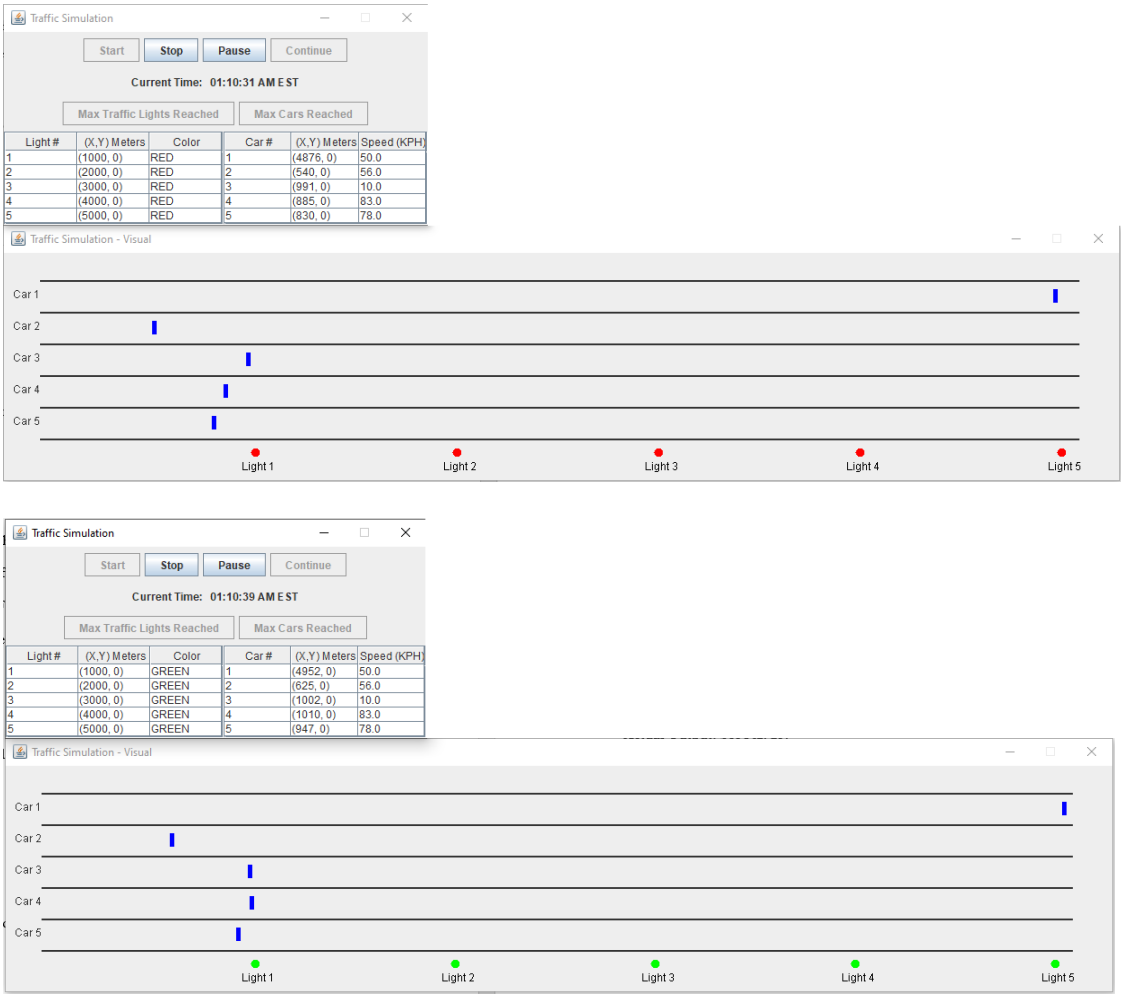



Figure 23: Test Case 12 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

13. **Aspect Tested:** Car: drive through green light

**Input:** N/A, watch car#2 drive past green light

**Expected Output:** Car#2 passes Light#1, which is green

**Actual Output**: See Fig. 24. First screencap is before Car#2 reaches light#1, and it is Green. Second screencap is after Car#2 passes light#1 because it stayed green.

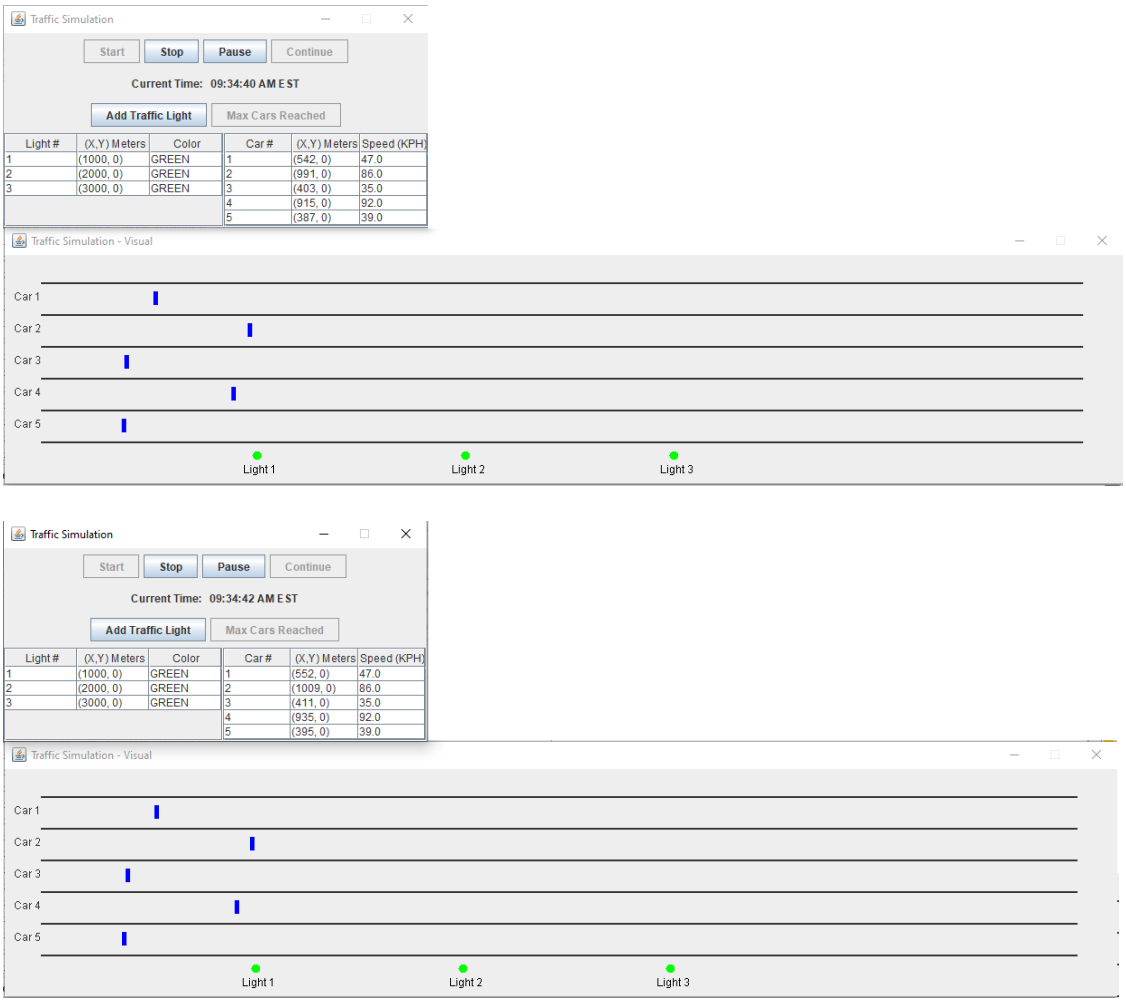



Figure 24: Test Case 13 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

14. **Aspect Tested:** Car: drive through yellow light

**Input:** N/A, watch car#2 drive past yellow light

**Expected Output:** Car#2 passes Light#1, which is yellow

**Actual Output**: See Fig. 25. First screencap is when light#1 has just turned yellow and Car#2 is about to reach it. Second screencap is 2 seconds later when light#1 is still yellow and car#2 has passed it.


Figure 25: Test Case 14 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

15. **Aspect Tested:** Car: stop at red light

**Input:** N/A, watch car#3 stop at red light #2

**Expected Output:** Light#2 is red. Car#3 stops at Light#2 with speed = 0 if light is red.

**Actual Output**: See Fig. 26. First screencap is when Light #2 is red, and Car#3 has just stopped at it with speed = 0. In second screencap, a little under 2 seconds have passed (check timestamps), and car#3 has not moved since the light#2 is still red. Speed = 0 still. The other cars have moved since they are not at a red light.


Figure 26: Test Case 15 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

16. **Aspect Tested:** Car: resume driving after red light turns green

**Input:** N/A, watching cars#4 and #5 stop at red light and then resume when it turns green

**Expected Output:** Light#1 is red, and Car#4 and #5 reach it and stop before it, speed = 0. When Light#1 turns green, car#4 and car#5 resume driving at their normal speeds.

**Actual Output**: See Fig. 27. First screencap is when light#1 is red and car#4 and #5 have stopped before it. 6 seconds later, light#1 has turned green, and car#4 and #5 have driven past it.

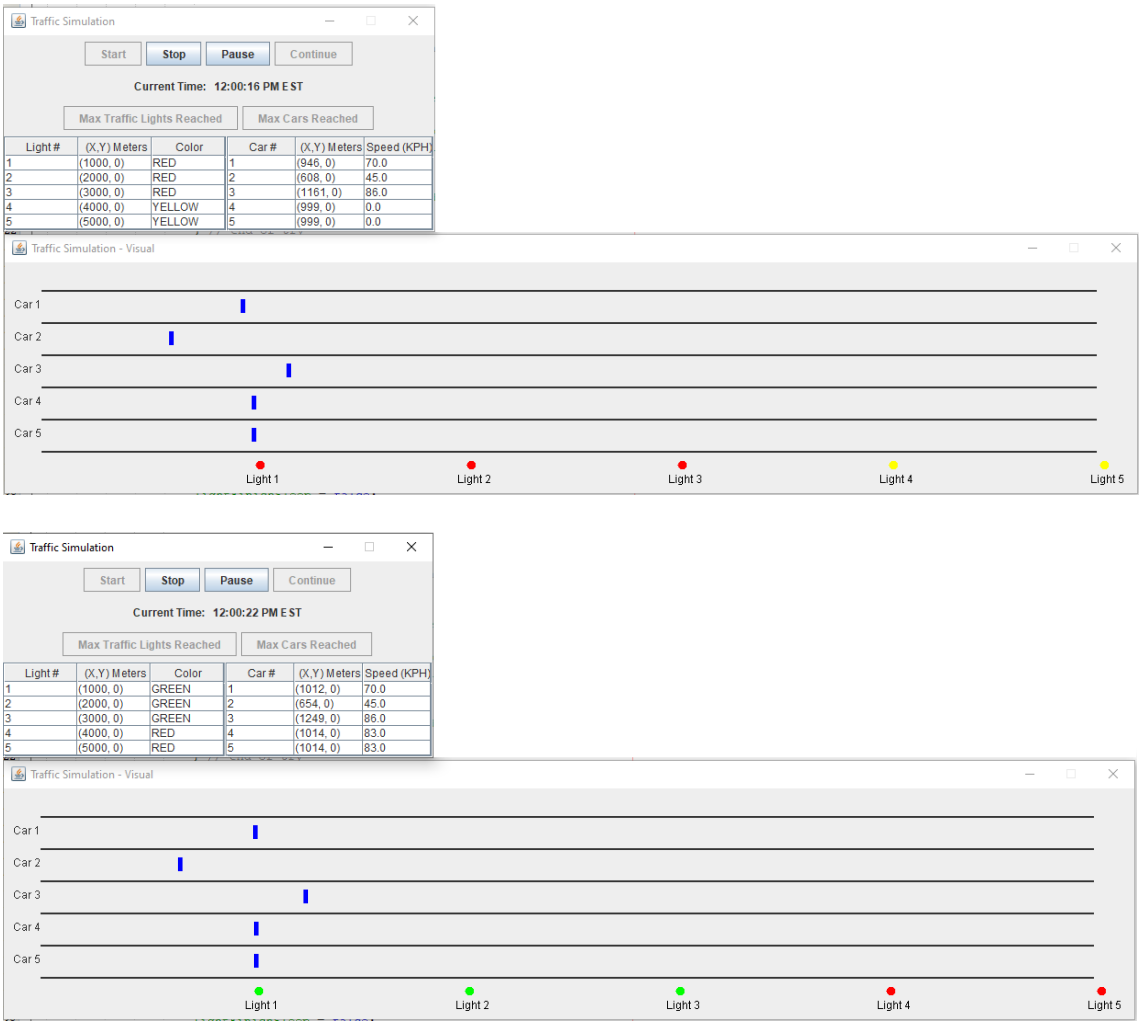



Figure 27: Test Case 16 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

17. **Aspect Tested:** Car: loop back to beginning after 5000m

    **Input:** N/A, watch car#4 return to 0m after passes 5000m

    **Expected Output:** Car#4 returns to 0m after passing 5000m, which is also light#5

    **Actual Output**: See Fig. 28. First screencap is car#4 stopping at 4999m before the red light #5. Second screencap is car#4 having looped back and now at 2m a few seconds later.

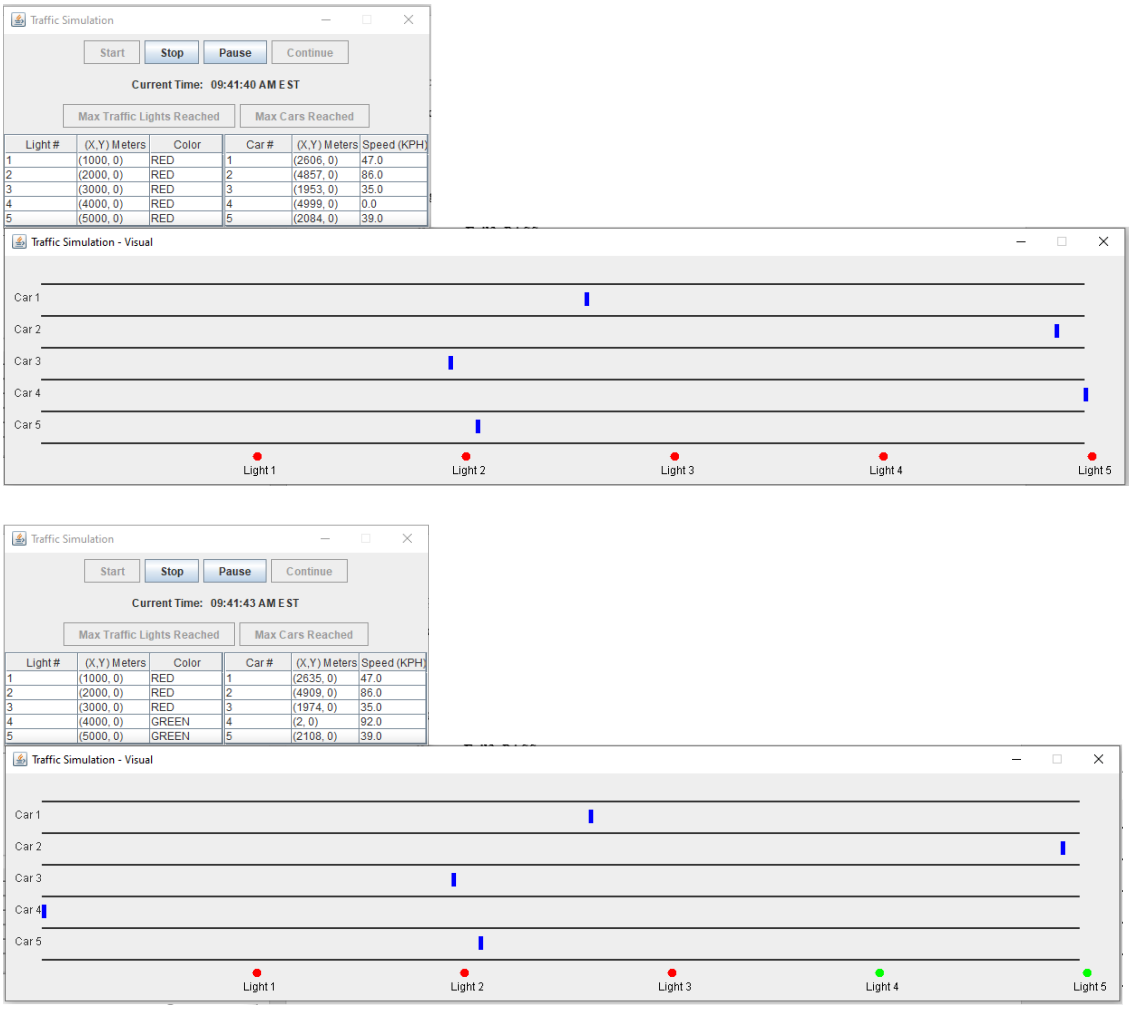



Figure 28: Test Case 17 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

18. **Aspect Tested:** Stop button

**Input:** Click “Stop” button

**Expected Output:** All threads stop running and terminate, causing information in both control frame and visual display frame to stop updating. The cars no longer move and the lights no longer change lights, now completely stopped. The timestamp no longer updates. All buttons become disabled.

**Actual Output**: See Fig. 29.

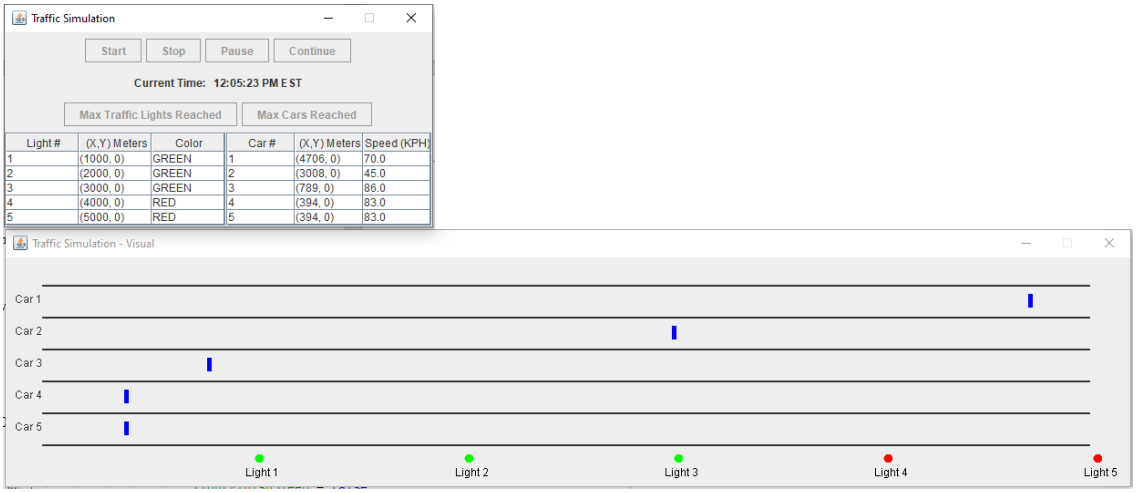



Figure 29: Test Case 18 Output. (Kucera, 2020)

**Pass or Fail?**: PASS

Week 8: Project 3

**Lessons learned**

Being my first major project that involves multithreading (and just managing threads as a whole), I took me too long to grasp the syntax of thread methods and synchronized solutions such as wait(), notify(), and sleep(). During the project I even realized I had been misunderstanding locks the whole time and had to backtrack a lot. In the end, I was still able to figure those out thoroughly and now feel more comfortable working with threads.

For me, the biggest challenge is managing the timing of traffic light threads when the pause/continue buttons are involved. The reason I have several blocks of code for sleeping in my TrafficLight inner class is because of the different situations that arise when the program is paused. Initially, my TrafficLight sleeps for a specified amount of time, then if the pause button is pressed, it enters a block where the thread will wait(). After being notified, the program sleeps again for the remaining time the light color has to stay. But what if the continue button is pressed before the wait() block is reached? The light changed colors too fast since it did not wait(), and therefore did not resume sleeping. So, I had to make another block that specifies “if the thread has not reached wait() and pause was pressed, then resume sleeping for THIS amount of time”. They are two different situations, and I’m even finding it difficult to explain it words here. I’m glad I figured it out though.

The biggest lesson I learned in this project was to make it a habit to check the resource/CPU usage of my programs. At one point, I tried to put an action listener inside a thread (not in the GUI constructor) and the simulation simply slowed down to a crawl. The car distances were hardly updating and the timestamp would skip around. After some time of debugging, I checked my processes in Task Manager and saw that the simulation was taking up 90% of my CPU utilization (normally it’s around 10-20%). I was actually getting scared - I had

no choice but to take the listener out of the thread and it went back to normal. Of course, I now know that listeners normally don't belong in run() methods, and it could have had severe consequences if I was working with a computer with bad specs. From now on, I'm going to be cautious about trying unconventional code and researching things first. I don't want to crash any system just from trying some random code out.

Overall, I am satisfied with the end result of my program and enjoyed the challenge. Everything works the way it should, and I started really early so I am not running out of time. The traffic light threads run smoothly when concurrent with the car threads and even the timestamp thread. Using invokeLater took some time to figure out the syntax for, but watching tutorials on Youtube helped me with that and now I can interact with the GUI from my run() methods. If I could change something, I would find a way to pause and continue threads more smoothly without needing the variables and methods for storing time and making separate code blocks for manually resuming sleep() (which is what I did). I'd like to continue working with threads to figure out a convenient way to do that.

On a final note, the biggest lesson I learned from this entire course was paying attention to code duplication. I've never had a professor who cared about it before, and so neither did I. Now, I tried my best to simplify and factor out code for both readability and lessening the amount of work the system has to do. I'm grateful for this good habit I've picked up now.

Week 8: Project 3

## References

Kucera, 2020. .zip file has been unzipped.

Kucera, 2020. Creating Java Application Project in Netbeans.

Kucera, 2020. Creating Test Kucera Project 3.

Kucera, 2020. Copying Application Source Files.

Kucera, 2020. Open SimulationGUI.java, click Run Project.

Kucera, 2020. Pasted Source Files appear in IDE.

Kucera, 2020. Pasting Application Source Files in Test Project.

Kucera, 2020. Project 3 UML Class Diagram.

Kucera, 2020. Project Properties > Run, type in Main Class.

Kucera, 2020. Run project, GUI displayed.

Kucera, 2020. Test Case 1 Output.

Kucera, 2020. Test Case 2 Output.

Kucera, 2020. Test Case 3 Output.

Kucera, 2020. Test Case 4 Output.

Kucera, 2020. Test Case 5 Output.

Kucera, 2020. Test Case 6 Output.

Kucera, 2020. Test Case 7 Output.

Week 8: Project 3

Kucera, 2020. Test Case 8 Output.

Kucera, 2020. Test Case 9 Output.

Kucera, 2020. Test Case 10 Output.

Kucera, 2020. Test Case 11 Output.

Kucera, 2020. Test Case 12 Output.

Kucera, 2020. Test Case 13 Output.

Kucera, 2020. Test Case 14 Output.

Kucera, 2020. Test Case 15 Output.

Kucera, 2020. Test Case 16 Output.

Kucera, 2020. Test Case 17 Output.

Kucera, 2020. Test Case 18 Output.

Kucera, 2020. Unzipping a .zip file.