Mechanics B

Project 1

Project Report

Tigran Minasyan

Here I will explain how I wrote my code and how each case works. First of all, in *stop* method, I used the formula **v12 – v02 = 2aΔx**, where **a** is the negative acceleration, then I assumed that the car is able to stop, so I took **v1 = 0** and found **Δx** from the formula, which is the distance that the car will pass before it stops. Then I checked if **Δx** is smaller than **x0**, which means that the distance that the car passes before stopping is smaller than the distance to the intersection; in other words, if the car stops, it will not reach the intersection, so it will successfully stop. If it is so, the method returns true. If not, then the car will stop after passing the intersection, so it will not be able to stop. In this case, false is returned. Since everything is given in meters and seconds, while the velocity is given in kilometers per hour, I divided the velocity by 3.6 in every place where I used it to turn it into meters per second.

The second method, *pass*, checks whether the car will be able to pass the intersection before the red light or not if it accelerates. Here I used the following formula: **s = at2/2 + v0t**, where **a** is the positive acceleration. The formula gives the distance that the car will pass during **t** time if it accelerates. If **s** is bigger than **x + Δs**, which means that the distance that the car will pass during **t** time is greater than the distance from the car to the intersection plus the intersection widths, in other words, the car will pass the intersection before the light turns red, then true is returned. Otherwise, no is returned.

In most of the cases, either the car is able to stop and not able to pass, or vice versa. But there are also some cases where the car is able to both pass and stop. There are also very few cases where the car is neither able to stop nor pass, which means in both cases, the car will end up being located in the middle of the intersection when the light turns red. This happens when the car has a slow speed, slow acceleration, the distance to the intersection is small, and the intersection is wide, and the time is short, too.

The third and the most complicated function, *passMaxSpeed*, checks if the car will be able to pass the intersection if the maximal speed it can reach is limited. I assumed that max speed is always greater or equal to the initial speed. First of all, I checked if the car will be able to pass without maximal speed boundary, because if it cannot, then there is no need to continue. If the car can pass, I calculated the time **t0** needed for the car to reach its maximal speed. Then, I used the same formula for checking the passing, but instead of **t**, I used **t0** first to calculate the distance the car will pass while accelerating to its maximal speed, then I calculated the distance it will pass while maintaining its maximal speed: **(t - t0) \* v1**. Again, if this distance is bigger than **x0 + s**, true is returned, false otherwise. The examples show that the car is able to pass without speed limit but not pass with speed limit if the speed limit is close to the initial speed.

The fourth method, *twoCars*, checks the pass and stop functionalities when there are 2 cars driving towards the intersection, with **d** distance between them. I simply called *stop* and *pass* methods inside this method for each car to check if they will be able to pass. But, if the car in the front is not able to pass, then the car in the behind will, obviously, not be able to pass too, even if it has very high speed, as it cannot go through the first car. I created another car object for the car in the behind as if I changed the **x0** of the car behind, the value for the object would be changed forever. It was simpler to have another object to check the functionality for than change the original object then change it back.

**Graph**

Since I was unable to write some code that would graph the distance-time and distance-speed graphs, I used the online tool Desmos: <https://www.desmos.com/calculator/fsuy0koyue>

For graphing, I used the formula **s = at2/2 + v0t**. In both of the graphs, *y* axis shows the distance (**s**). In the first graph (red line), *x* axis shows the time (**t**), while in the second graph (black line), *x* is the speed (**v**). In the first graph (red line), **a** and **v** are constant and can be changed with sliders to see the distance-time graph with different acceleration and speed values. In the second graph (black line), **t** and **a** are constants, to see speed-time graph with different acceleration and time values, as I could not construct the graphs by neglecting **t** and **a**.

The acceptable values of *y* (distance, **s**) are 15-170, as the boundaries are: **x0 = 10, Δs = 5; x0 = 150, Δs = 20; s = x0 + Δs**. The acceptable values of *y* can be seen with a blue field on the graph. The acceptable values of *x* on the first graph (red line, *x* is time, **t**) are 2-5, which can be seen with a green field. The intersection of blue and green fields shows the range of all possible inputs. For certain **a** and **v** in the first graph, the *y* value shows the distance that the car can pass in the given *x* time with constant **a** and **v**. The acceptable values of x on the second graph (black line, *x* is speed, **v**) are 5.5-22.5 m/s, which is almost equal to the range 20-80 km/h. In the second graph, for certain **a** and **t** values, the *y* value shows the distance that the car can pass with the given *x* speed in constant **t** and **a**.