User’s Guide:

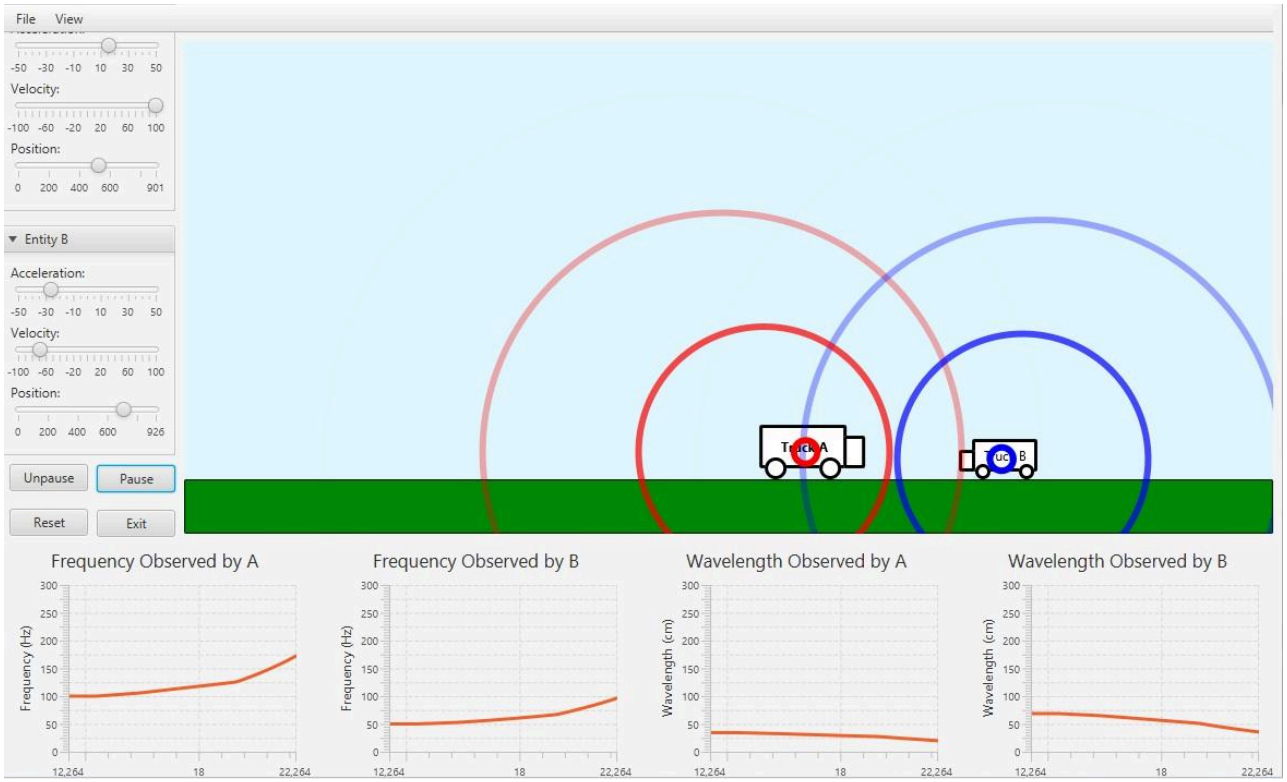
Our application simulates the Doppler effect, a method for calculating the frequency of an object. Using the acceleration, velocity and position, the user has the possibility to see in real time the frequency of each object, while looking at an animation. To run this project, the user must move one or more sliders of at least one of the two trucks. In fact, 2 trucks have three sliders each. Every slider gets the value of the acceleration, velocity or position of a truck. These values are used for the calculation of the Doppler effect. Furthermore, the user can use the taskbar to change the colour of the trucks and the intensity of the wave produced during the animation.

For more instruction, this is a step-by-step guide to follow:

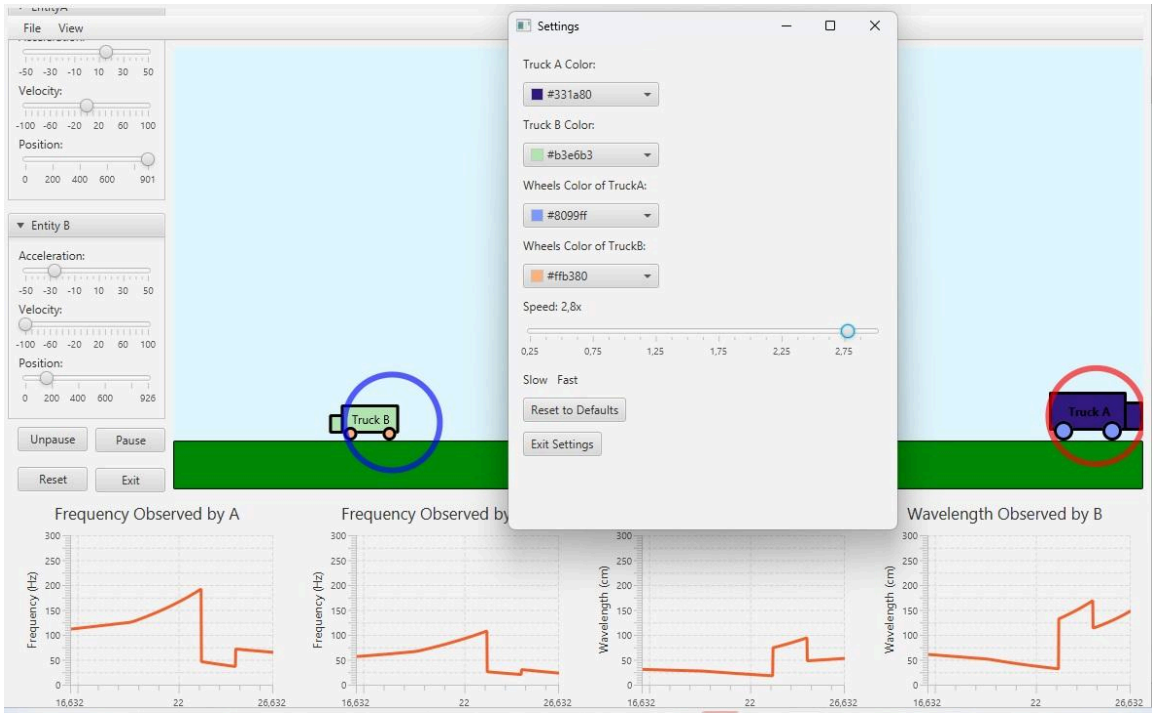
1. Move the acceleration, velocity or position slider of Truck A.
2. Move the acceleration, velocity or position slider of Truck B.
3. Watch the trucks move, and the frequency and wavelength graphs change.
4. Click on pause to pause the application, then click on unpause to unpause it.
5. Click on reset to reset the animation from the beginning.
6. Click on exit to exit the application.
7. On the taskbar, click on View, then Change Colours, and a window will appear.
8. Click on the buttons to change the colours of the trucks.
9. Move the slider to change the intensity of the waves.
10. Click on reset to reset the waves and trucks to their initial state.
11. Click on exit when the modifications are finished.

If something goes wrong, the user should check for a build or compilation error. Using another Java SDK lower than 22, the application might not run since this SDK might not have all the necessary properties. Moreover, the user needs to install a JavaFX library equal to or higher than 22. If not, the application might not work. Also, if the application runs for a really long time, the user might have a memory issue with the points created for the graphs. To avoid that, it’s possible to change the code to limit the number of points.

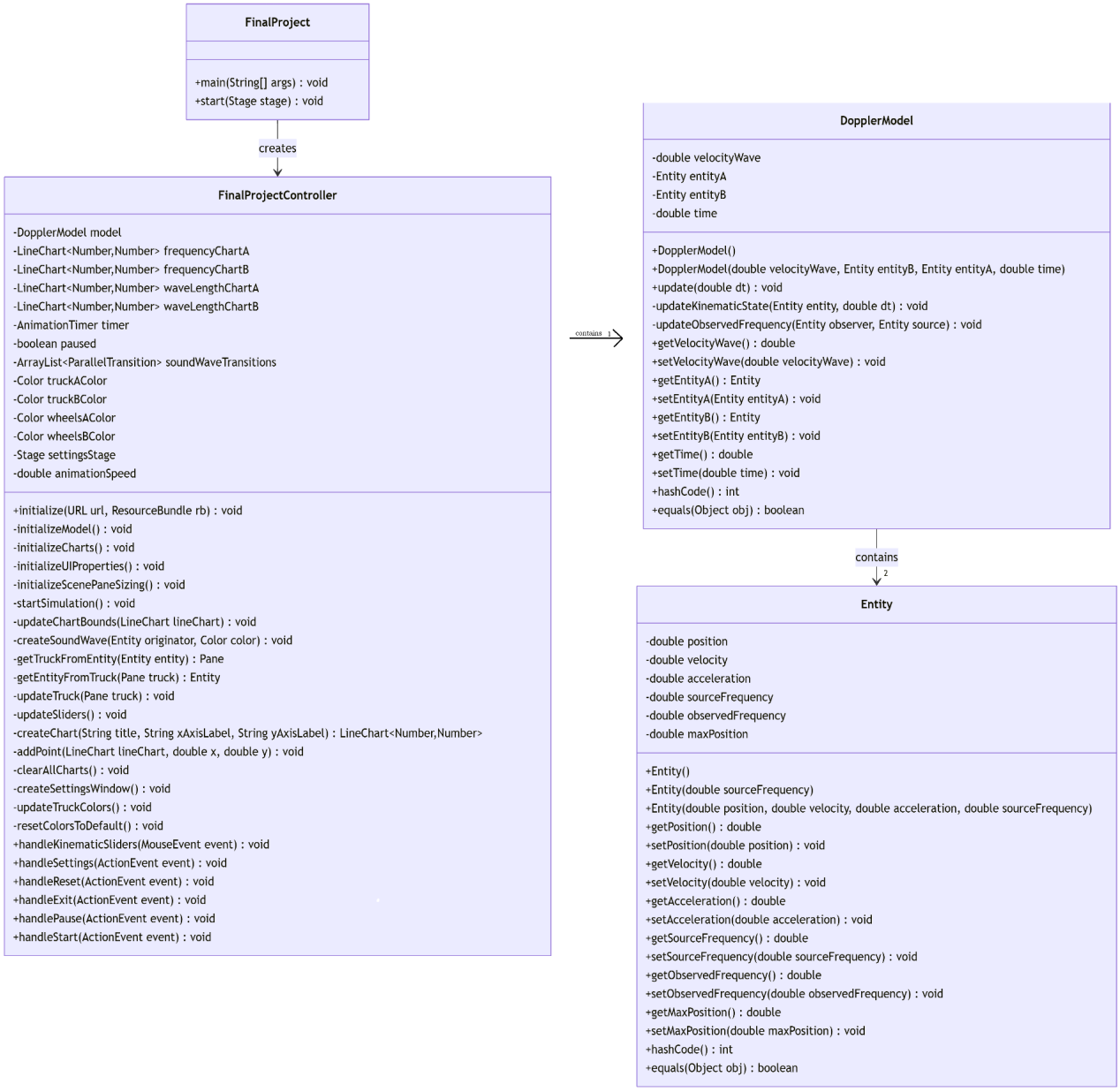
Screenshots of the application:
Simulation:



Settings:



Design Overview:



Role and Description of each class:

Entity Class: This class contains all the variables related to the calculation of the frequency of the trucks. Some of these variables are sourceFrequency, observedFrequency or even position, velocity and acceleration. It's used to contain all the necessary data for the calculation of the Doppler effect and the animation of the trucks.

DopplerModel class: It's used to do practically all the necessary calculations of this project. In fact, all the calculations related to the Doppler effect are in this class. Also, the position and velocity of the trucks are updated here.

FinalProject: This class contains the main. It's thanks to this class that our FXML is running.

FinalProjectController: This class is the connection between our calculation and our scene builder. In fact, the handlers of our FXML are here. The graphs are also created, as well as the animation. Moreover, all the labels, buttons, sliders and color pickers inside the settings window are created here.

FinalProject FXML: That's our scene builder. The trucks, sliders and buttons are there. We used it, since it's easier than coding everything.

Challenges and solutions:

Our first idea for the calculation of the Doppler model was to have one observer and one source. However, we realized that the calculation will be more complicated since we would have to always specify which truck is the source and which one is the observer. Instead, we decided not to classify our trucks this way, since they both can be an observer and a source. We changed our Entity class so that the trucks have all the same variables. Inside the DopplerModel class, the calculation of their frequency is done depending on their position or velocity. Facing this issue, we learned that there are many ways to calculate the Doppler effect and that programming can simplify some mathematical calculations.

The second challenge that we faced was time management. In fact, we had to coordinate our tasks to finish it on time. For instance, we started by doing the Entity and DopplerModel classes, which are used for the calculation of the Doppler effect. Since we both had to work on it, we decided that one would do his part before the other. Also, we had to manage our project with our other exams. Then, to be sure to meet the deadline of our application, we tried to start our project early and used to-do lists to make sure to respect all the requirements. This challenge helped us work on our time management since we tried to find new solutions to be more efficient, and also our teamwork skills, since we had to coordinate our commits.

The third challenge that we faced was dividing the work equally. Indeed, we wrote our project presentation before learning about Scene Builder. Also, we realized later that our calculation for the Doppler effect was easier than we thought, and we learned that we needed to add Junits. Thus, we had to change the way we divided our work. In our project presentation, we had more classes and more small tasks. Since we used Scene Builder, we had to work on the same classes and methods together. Thus, we often changed who was doing what during the project. This challenge helped us develop our communication skills since we always had to update each other on our work.