Assignment 6
COMP 86

In this assignment we will add mouse picking, collision detection, and a few more features to the simulation/game of the previous assignments. As before, you need not be restricted by what you did in your previous assignments, but you should still provide all the features required by the previous assignments.

Mouse Picking

Add some widgets for commands that apply to an individual vehicle already on the screen, such as changing its speed, direction, or color. For this, the user will first click on the vehicle in the canvas to identify which vehicle the command applies to (since the individual vehicles are not Java widgets). Your program must therefore figure out *which* vehicle was clicked, to determine which vehicle the user's next command will apply to. You should also highlight the selected vehicle so the user will know which one it is.

To do this, you have to be able to translate *from* screen coordinates back into vehicle locations (your **draw()** routines translate the opposite way). Since we want to keep information about each vehicle's location encapsulated in the **Vehicle** object, the way to do this is to add another member function to **Vehicle** to do this job (which is called *pick correlation*.) It accepts a mouse location and returns a boolean to indicate whether that mouse location is on this vehicle. You can use a simple bounding rectangle, or you could use some of the `java.awt.Shape` methods for checking whether a point is inside a shape. You can install a callback routine for the canvas as a whole, and whenever it receives a mouse button down it can call this new function for every vehicle in your program until it gets back **true** for one of the vehicles (if it doesn't, then the user clicked somewhere else on the background).

Collisions

As each vehicle moves, check to see whether it has collided with with one of your other vehicles, and take some action that you choose. For example, the vehicles could be deleted, or perhaps when two vehicles collide you provide rules for how to choose one that crashes and one that remains, or perhaps they could lose speed or altitude but not necessarily crash. You will need code for comparing the locations of two objects to determine whether they are touching or not. You can use a simple bounding rectangle, or you could use some of the `java.awt.Shape` methods for checking intersection. When two vehicles appear to collide in x and y, you may need to compare their altitudes or depths to decide whether they should indeed collide or just pass over one another safely.

Game Rules

Add your own logic or rules to turn your simulation into a game that a user could play. You can make up any rules you like. For example the user could get points if a particular type of vehicle crashes, but lose points if another type does. Or perhaps gain points by visiting certain geographic locations. Or you could provide increasing levels of difficulty as the game proceeds.

Among other things, this will probably require an object that keeps track of the overall state of the game and maintains state information that applies to the game as a whole, i.e., beyond the states of the individual vehicles, which are already stored within the vehicle objects.

Make sure to include simulation/game instruction for *users* (not for programmers) in an appropriately titled section of your `readme` file. The instructions should state how to use/play and the goal of your simulation/game for *users.*

State Panel

Provide an area on your screen that displays the status of the game. It could show scores, number of vehicles remaining, or other relevant general information, depending on the rules of your game.

Program Design and Practices

(The rest of this still applies from previous assignments)

Your program design should exploit the features of object-oriented programming (encapsulation of code and data, support for abstract data types, polymorphism/overloading, inheritance).In particular, object-oriented programming provides us a good way to handle the various data needed in callback routines. You should use objects to encapsulate each interactive widget with the routines and data you need to use it or pointers to other objects containing such.

You should provide an object for each interactive widget or small group of widgets you create. That object should hold anything you need to remember about the widget from one callback to another, all the data pertinent only to the command for that widget or that you need to operate this control, (including, in most cases, a pointer to the model or other outside object needed to perform the actual action the user requested), and the widget's own listener callback routines.

If you have several widgets that share some behavior or properties, you should organize your objects into an appropriate inheritance hierarchy.

You will have other data that must be accessed by several widgets, particularly shared information about the state of the program or global information about the state of the user interface. Provide additional classes and objects for holding this kind of information.

Remember to trigger your drawing to repaint itself explicitly whenever one of your commands causes a change that should be reflected on the screen. And remember that the way to change the screen is first to change the data stored your classes and then to trigger the repaint.

You should follow these general Java programming practices:

- Make all instance variables of your classes **protected** or **private**. If you need access outside of the class, provide it with *set* and/or *get* methods; don't access protected the variables from outside the class (even though Java -- unfortunately -- allows us to).
- Avoid most global variables or widely-accessible public variables; pass the data you need explicitly.
- Put each class in a separate .java file.

Design Documentation

In addition to your program, submit documentation about the design of your system in these forms:

- An outline showing the inheritance hierarchy
- An outline showing the aggregation hierarchy (which objects contain or "own" which other objects)
- A list showing "uses" or collaboration relationships (which objects use which other objects to perform functions)
- The information hiding "secrets" of each of your classes (i.e., what design decisions are entirely encapsulated within that class).

Submit this documentation electronically in text form, along with brief instructions for how to compile and run your program. Include it as part of the `readme` file that you submit with your assignment.