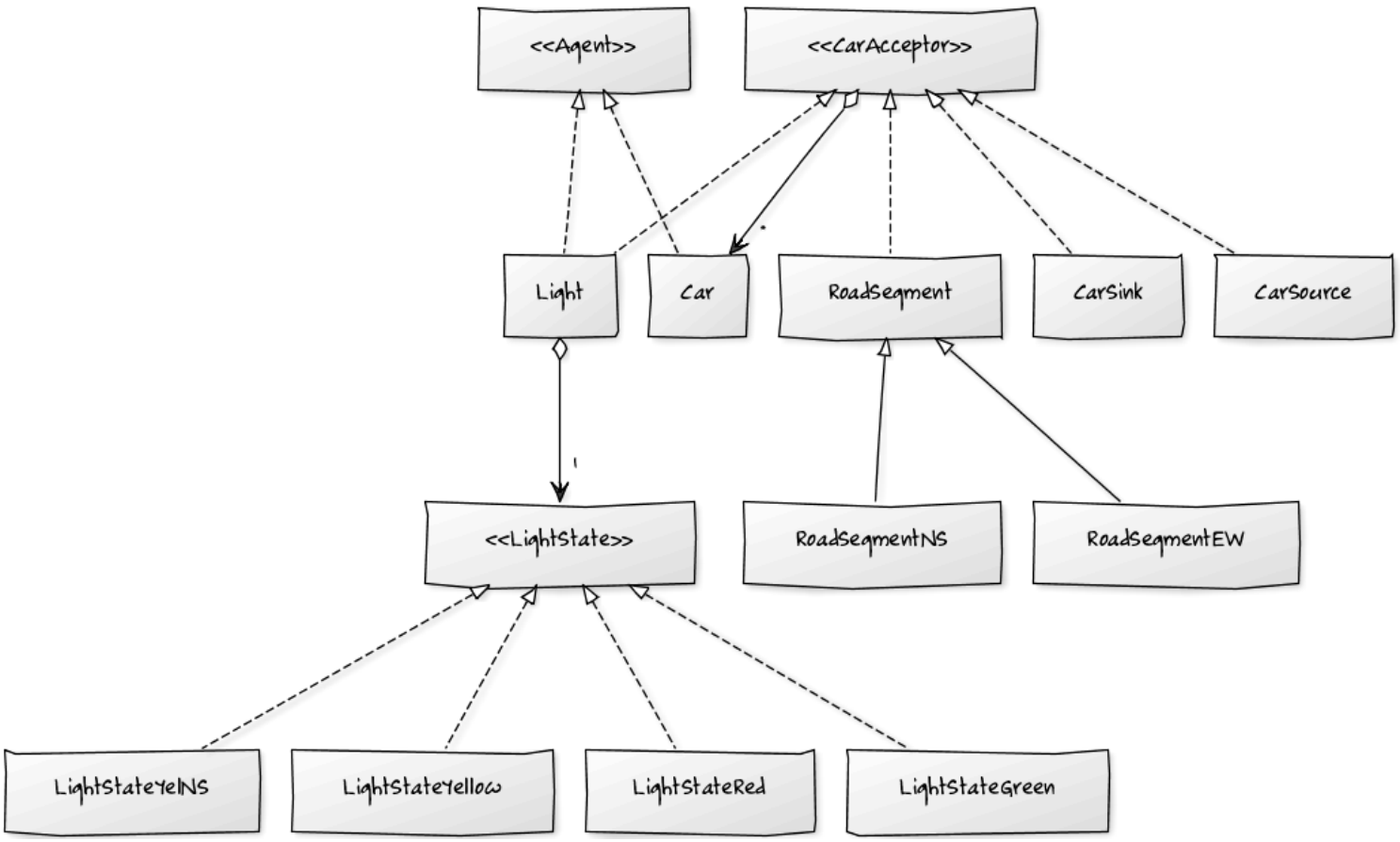
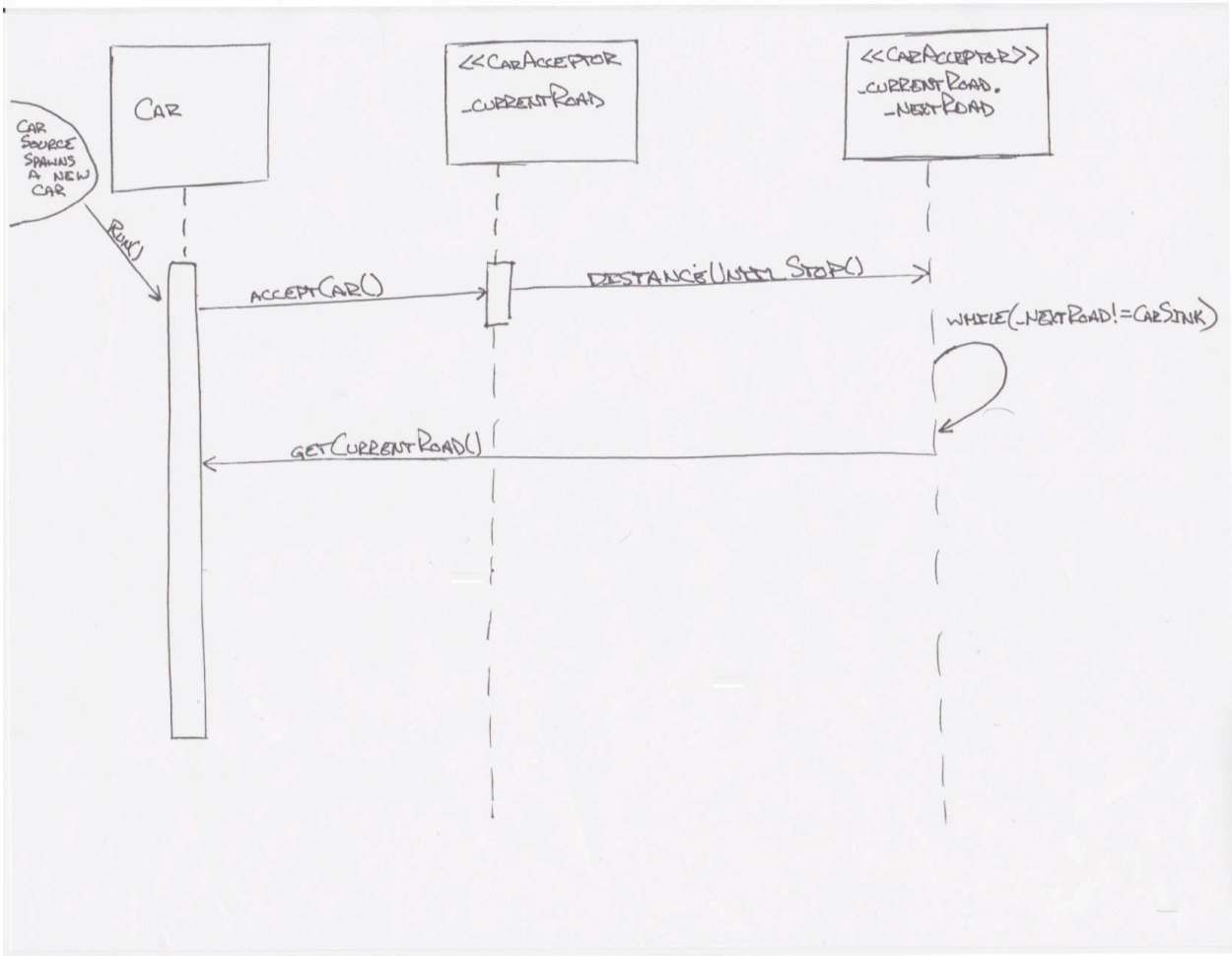
Raymond Elward
SE450
Prof. Jagadeesan
3/7/2011

Final Project Traffic Simulation:

UML for classes made by me in the project.model package:



UML Sequence Diagram for how car updates its position:




Time Summary:

| | Week 1 | Week 2 | Week 3 | Week 4 | Total |
|---|---|---|---|---|---|
| Design | 5 | 0 | 3 | 4 | 12 |
| Coding | 3 | 10 | 18 | 8 | 39 |
| Debugging | 0 | 3 | 10 | 5 | 18 |

Notes on patterns:

The first pattern I implemented off the bat was the static factory. This served my needs for a single place to have the creation of objects to occur. This solved the problem of giving access outside the package to instantiating instances of my class, as well as giving me a central place to instantiate my own classes.

The strangest implementation I have in this project is the Light class. This class serves as the representation for intersections and traffic lights in the simulation. The class implements the state pattern. Instead of having a crazy amount of case statements for a lot of the Light's behavior methods these are called dynamically on the current state of the Light. The light owns a LightState object. LightState is an interface that is implemented by the 4 different states of the light. Why are there four light state, I thought a light was only red, yellow or green? Well since I implemented the Light class to act as the whole intersection its fourth state is for a yellow (or slow down) light for north/south facing cars before it turns to green for the east/west cars. Anyway, these states all dynamically handle key functions for the light class, which solves the problem of having a ton of cluttered if-else statements in my light code. Now instead of these complicated case statements I pass the work dynamically to the java virtual machine by having each individual state solve its own problems.

Another big pattern I implemented was the Template pattern. The template pattern helped me solve the problem of differentiating between the north / south roads and the east / west roads. I made a RoadSegment class that is abstract and implements the CarAcceptor interface. I them made two final concrete classes RoadSegmentEW and RoadSegmentNS that could uniquely handle the differences in the two types of road.

Successes and Failures:

I think my biggest failure was trying to incorporate all the code that was given. For instance I spent a lot of time merging the example models with the timeserver demonstrated in class. I think realizing that I should just pick one choice or the other and implementing it would have served me a lot better in the long run.

Implementing the light to be working was probably the most difficult thing I found in the program. I was going off of the provided code for the simple model and swing classes. This basis left me attempting many iterations of the light class. Finally I gave up on the having a light controller that holds two lights. If I had time to toss the whole program out and start again the thing I probably would have concentrated the most on would be this aspect. It really bothers me that the visual of the light on my simulation only shows the east west light state. I would need a light controller with two lights to have it where I could fully visualize the other north south light. Spending a ridiculous amount of time on this aspect was a failing on my part.

I implemented the intersections so that they only had one light; this led to an interesting design decision. I needed the CarAcceptor interface to implement new methods now to serve the light so it could function to pass along north south cars in the correct direction and east west cars also in its correct direction.

I had a lot of trouble working with the swing graphics, at least the way it was implemented in the code given to us. But eventually I was able to crack the cipher and control the content pane in ways I never have before. That was a great success.

One last problem has to be with setting the road length. I feel like I spent too much time fiddling with the swing tools provided. But one of the ultimate things I wanted to have in the final piece was getting the road lengths to normalize on screen. The way I have it now road length is the one major parameter that the users cannot change. I kept running into visual problems where cars would be way past where a road ended or skip to the next road segment before the road ended. If I had time to familiarize myself completely with swing I probably could have fixed this, but since this assignment is more about the patterns I decided to set the road length in stone and set this issue on the back burner.