Journal Report 6
10/7/19 - 10/13/19
Richard Zhan
Computer Systems Research Lab
Period 2, White

Daily Log

Monday October 7

In my input file, I added a parameter that changes the frequency of DTD cars being spawned. In my Car class, I changed the variable $events$ to be a vector of values instead of a vector of pointers. This should save memory for longer simulations. It also allows me to write cleaner code.

Tuesday October 8

I reorganized my $step\_simulation$ method and moved some parts of it to new file called $communications.cpp$. The new file contains the methods $createEvents$, $transferEvents$, and $exchangeEvents$. $createEvents$ stores the current speed vs. $speedLimit$ of the $Edge$ that each Car is on. $transferEvents$ is run after $createEvents$, and it simply checks each pair of $Cars$ to see if they are within the $COMMUNICATION_RANGE$. If so, then $exchangeEvents$ is run, in which all the $Events$ corresponding to each $Edge$ for both $Cars$ are compared. For each $Edge$, the more recent $Event$ is assigned to both $Cars$.

Thursday October 10

I altered my method $astar$ to try to support DTD navigation. For DTD cars, instead of using the $speedLimit$ of the $Edge$, we first check to see if there is an unexpired $Event$ that corresponds to that $Edge$. For some reason, the DTD cars currently are still choosing the same paths as the non-DTD cars. I specifically created cases where the DTD cars should've selected a different route from non-DTD cars, but they still choose the optimal non-DTD navigation route. I still have to fix this bug.

Timeline

Date	Goal	Met
9/23/19 - 9/29/19	Finish writing the simulation code and tweak variables to reach realistic settings.	Yes, I wrote the necessary code and found the correct input values to run a realistic simulation.
9/30/19 - 10/6/19	Began coding the naive (non-optimized) DTD scheme. Try to finish setting up the class $Event$ and the communication system between cars	Yes, I set up the class $Event$ and the DTD car communication system. I still need to incorporate these Events in the DTD navigation system
10/7/19 - 10/13/19	Finish the naive DTD scheme and begin looking into optimizations	No, I did not finish the naive DTD scheme this week, and I am currently stuck on a bug with it. As a result, I was also unable to start looking into optimizations (which aren't currently needed on these small-scale runs)
10/14/19 - 10/20/19	Fix the DTD navigation bug. I also want to try to create a GUI, which would be useful for debugging and overall visualization of this project.	
10/21/19 - 10/27/19	I would like to add functions to my GUI that would allow me to see the history of $Events$ and $Cars$. It would also allow me to see overall stats of the run as the program is executed in real-time.	

Reflection

This week, I cleaned up some parts of my code. I also worked on the implementation of the DTD navigation system. However, while coding the DTD navigation system, I created an issue where the DTD cars do not actually choose a different route from the non-DTD cars. I am still working on resolving this issue, but it has been difficult to parse through all of the logs. As a result, for this situation and potential future bugs, I would like to create a GUI for my program that allows me to see the map and the Cars in real-time. It would be useful for as a debugging and visualization tool. The most basic version would display the Cars, Vertices, and Edges. I also want to add more in-depth functions for the Cars that allow me to gain more information as I click on them. This information would include stored Events, the communication radius, which other Cars are within the communication radius, etc. Another feature would be clicking on Events to trace the history of how it was passed from Car to Car.

The code listed below contains my reorganized communication methods.

```
1  /**
2   * For each car, creates an event based on current speed
3   */
4  void createEvents() {
5    for (auto it = graphCars.begin(); it != graphCars.end(); it++) {
```

```
      int id = it->first;
      Car& car = it->second;
      if (car.roadIndex != -1) {
        car.events[car.roadIndex] = Event(EVENT_COUNT++, car.currentRoad,
            car.currentRoad->actualSpeed, car.currentRoad->speedLimit,
            CURRENT_TIME);
      }
    }
}

/**
 * Given two Cars, compare each of the stored Events for each road
 * Store the more recent Event for each road in both Cars
 */
void exchangeEvents(Car& c1, Car& c2) {
  for (int i = 0; i < EDGE_COUNT; i++) {
    if (c1.events[i].id == -1) {
      c1.events[i] = c2.events[i];
    } else if (c2.events[i].id == -1) {
      c2.events[i] = c1.events[i];
    } else {
      c1.events[i] = c2.events[i] = (c1.events[i].startTime <
          c2.events[i].startTime) ? c1.events[i] : c2.events[i];
    }
  }
}

/**
 * For each pair of Cars within 'COMMUNICATION_RANGE', exchange all Events
 */
void transferEvents() {
  for (auto it1 = graphCars.begin(); it1 != graphCars.end(); it1++) {
    Car& c1 = it1->second;
    auto it2 = it1;
    it2++;
    for (; it2 != graphCars.end(); it2++) {
      Car& c2 = it2->second;
      ld dist = distance(c1, c2);
      if (dist <= COMMUNICATION_RANGE) {
        exchangeEvents(c1, c2);
      }
    }
  }
}
```