Design Studio 2 - Part 2

Project	Design Studio 2 - Traffic Light Simulator
Class	IN4MATX 121 Software Design
Instructor	Alberto Krone-Martins
Date	Nov 11, 2021
Team	Group 25
Group Members	1. Joshua Hsin a. ID# 13651420 2. Wen Kang a. ID# 18746019 3. Chun Hong Chan a. ID# 19025282 4. Basma Bahia a. ID# 71434934

Table of Contents

highlighted in red are things removed from part 1, highlighted in yellow are things added in part 2

Reflection from Design Studio 2 - Part 1

When developing our traffic simulator design for Design Studio 2 Part 1, we ran into many logistical issues that affected our design such as deciding on the number of lanes a road section could have, how sensors and timer would work together and implementation options for our road sections and intersections in the simulation. For some of our ideas, we simplified our design due to feasibility and understandability for the user and us as designers, such as declining to implement left and right turn lanes and multiple lanes on one side of the road section, which would complicate the logic of our road sections and vehicles. It was also difficult to plan out how the implementation of the program would work, what classes we needed, instance variables classes needed, and how classes would interact with each other. For example, we first included the Route and Location attributes within the Vehicle class, but later abstracted them with their own classes so the program was easier to understand. Our Intersection class also needed a way to keep track of its location, so having a class that could be used by both Intersection and Vehicle to represent the location made our implementation smoother and more intuitive.

While thinking about our different approaches, we also ran into trouble deciding how to make our approaches uniquely distinct from each other while remaining viable on the surface. In fact, we had to delete one of our approaches due to it being redundant and infringing on the requirements. This approach provided the option for the user to implement vehicles with different speeds, however, other features in other approaches which were the same were already being explored, leaving little distinction between this approach and others to be worth exploring. For other approaches, we brainstormed general ideas of variables to change across approaches, but remained unsure of how to change implementations across variables, as we did not have a set idea of the overarching goals and results we wanted from each approach until explicitly stating and naming our approaches with easily understood directives such as User-friendliness, Feature Availability, and Specificity.

What did work from Design Studio 2 - Part 1 was our eventual focus on the essence of the project - education. We didn’t select an approach focusing on the efficiency of one vehicle, or a user-friendly approach lacking in features and functionality for teaching and demonstration, or an approach ripe with features but too complex for an introductory class (our assumption). Instead, we focused on providing an educational experience for the user which was intuitive, but provided enough realism and user control such that the user would be able to add vehicles, add routes, and change lights. Allowing the user the option of including sensors or not to traffic lights and changing traffic lights manually or automatically based on timers also gave users the decision, based on experience and overall goals of learning to be hands off or hands on in their simulation. These options allowed for progression of user learning of the program and traffic flow, removing the need for a complicated tutorial or other intensive learning process for the user before viewing, exploring, understanding, and modifying traffic states.

Proposed Changes for Design Studio - Part 2

We have modified existing features from and added necessary features that were specified in the Design Studio part 2 requirements to our selected approach from the Design Studio part 1, which was our approach 3 — Focus on Optimizing on All Traffic Present. Some uncertainties that arose from our Design Studio part 1 were also resolved with the additional and changed requirements from the Design Studio part 2.

There are a few existing specifications that we have decided to remove. We first decided to remove our goal of finding the optimal solution or combination of traffic light settings that makes all of the traffic present the most efficient, because it has been made clear that students should be able to see their own simulation and the instructor's simulation at the same time. The instructor's simulation will serve as the model answer for the students' learning experience, and thus there is no need for a generated optimal solution. The teacher will be able to create and modify a simulation, starting from scratch or a simple starting point. Adding additional complexity or modifying different factors of the simulation will teach students a lot more than simply providing an ideal solution, especially since traffic is rarely ever perfect (traffic is unpredictable). Traffic changes and evolves throughout time so improving and worsening traffic conditions can inform students about traffic interactions and relationships, which is vastly more important and feasible than discovering a magic bullet. Retaining the feature of finding an optimal solution will also reduce students' interaction with the software. This modified change allows us to keep the proposed software more feasible and more appropriate for the purposes of education. The second specification we removed is the constraint that all vehicles must move at the same speed, because of the added possibility of vehicle passing, and the addition of trucks which travel slower than cars. Instead, cars move at the same speed, while trucks move slower.

The additional specifications that came with part 2 of the Design Studio allowed us to solidify our design approach. A major change we made was adding trucks aside from cars for our vehicles. The difference between trucks and cars is that trucks travel at a slower speed than cars. We modified the statistics to include truck data as a result. For example, we added the average wait time of trucks and combined vehicles (cars and trucks) to the statistics of each intersection and have the overall average wait time statistics in the Statistics class to account for trucks on top of cars. Also, the overall count for vehicles in statistics would also include trucks now.

Another addition we made to the program was the ability to save the simulation state to be able to close the simulation and return to it at another time. Saving the simulation would save the road section layout (including all road sections and intersections), traffic light state, which traffic lights have sensors, the time on all traffic light timers, statistics for the overall simulation, statistics for each intersection, and all vehicle metadata for each traffic light (to account for traffic intensity of a road section and traffic light behavior) and the overall simulation. Vehicle metadata would include the vehicle location, the type of vehicle, and the projected route of the vehicle. This would allow the user to return to different simulation scenarios in the future as needed if they wish to examine or modify the simulation rather than keeping tabs or the program running perpetually.

We also allowed the user to be able to create new tabs to run multiple simulations side by side in a split screen format. Allowing the user to run multiple simulations would give them the ability to compare different scenarios by changing different variables. The contrast between simulations and their results would allow the user to understand relationships between types of vehicles, traffic light timers, traffic light sensors, road section closures, and any other traffic variables.

Additionally, we also added names to cars because it will make the software better catered towards personas like professors, since they need to illustrate and present the simulation to students. Giving a name to a car can let students know which car the professor is talking about. Cars without a name would just have a null name.

Traffic light controls are also more consistent compared to part 1. In part 1, we mentioned that users or sensors can change traffic light state, but did not say how to do so. In this design, we made it clear how sensors can control the traffic lights. In part 1, we assumed sensors could only see if cars had arrived at the light, and that traffic lights would change randomly to another light with cars at the light. However, in part 2, we researched if traffic light sensors could detect the number of cars at a light and discovered that they could (**Appendix 3**). Given this new information, our new sensors in part 2 can detect the number of cars at a light in each lane and whether there are cars in each lane. Our new sensor capabilities allow our simulation to run smoother based on the new simulation, especially with the added complexity of multiple lanes. The light that automatically turns green on each light change is the light with most cars, or a random light given a tie. However, each intersection's traffic light will have a maximum wait time, which starts and continues to count down while the light is red, and resets once it turns green. When the maximum wait time counter turns 0 for any traffic light at an intersection, that light will be the next light that turns green no matter how many vehicles are present on any other road sections. This keeps road sections from having cars that wait too long or can never go due to having a lower number of cars than other sides of the intersection. Conversely, if a section of road or left turn lane controlled by a traffic light has no cars according to the corresponding sensor, the timer of the light will immediately change to two seconds (given the timer is currently above two seconds), allowing the light to turn from green immediately to yellow, then red. This interaction keeps other cars on other sides of the intersection from waiting without purpose, as the current green light can stop wasting time allowing no one to pass. It should also be mentioned that we also added the specification that users can manually change the state of any traffic lights, from green to first yellow then red, or from red to green. Changing a light from green to red should automatically trigger another light to turn green based on the aforementioned conditions. Changing a light from red to green, on the other hand, should produce a pause before the selected light is turned to green, allowing another light that is currently green to first turn yellow, and then red.

Furthermore, since there are two types of vehicles, cars and trucks, our new design handles the speed differences between them. To make the simulation simple, we assume that all cars or trucks will drive at their maximum speed when possible. In this design, if there is a truck directly ahead of a car, the car will wait behind the truck and pass the truck when it reaches a certain following distance given the car is not turning left or right in the upcoming intersection, there are no vehicles directly next to the car in the next lane, and there is a certain amount of space in front of the truck for the car to merge. This mimics the flow of traffic, as cars often try to pass slower trucks to get to their destination faster. We also gave details about how vehicles change lanes in order to turn at an intersection since our new design has multiple lanes. Cars attempt to change lanes to turn 1 intersection before the light in which they turn. The lane change is successful if there are no vehicles directly next to the car in the next lane over. However, when there is a vehicle immediately next to the merging car, the car will slow down and merge when there is space. If there is no vehicle that meets a certain distance threshold behind the vehicle immediately next to the merging car, the merging car will merge right after slowing down. This threshold keeps a vehicle from hitting the car while the car is merging. In all other cases, if there is a vehicle that meets the anti collision distance threshold behind the vehicle next to the merging car, that vehicle will slow down to a speed slower than the merging car, allowing the car space to merge into the lane. Trucks merge in the same way as cars (behaviors of other vehicles involved in the merge stay the same), but do not slow down before merging, and have a greater distance threshold (as they are larger). After vehicles merge, all vehicles that changed speed in relation to the merge will return to their original top speeds. These changes give more realism to the behaviors of cars and trucks in real world traffic and allow a more holistic view of traffic relations to students.

Just like part 1 of this Design Studio, incoming traffic is controlled by a timer by default and vehicles can be manually added by users at the same time, but for part 2, we decided to add presets to traffic density settings that users can choose from when they are adding cars and trucks to outer road sections with a timer. The presets include not used, seldom used, normal (default density), and busy, which are sorted from lowest to highest density of traffic. Giving the user these options and labels allows more useful comparison of traffic states and easier discussion of the traffic states rather than having arbitrary values. It should also be noted that there are two individual timers for traffic on each outer road section, one for cars and one for trucks. Additionally, trucks enter the outer road sections, by default, on the right lane. This lane is generally considered the slow lane and is usually reserved for trucks to keep from interfering with the general flow of traffic - faster cars, mimicking real life traffic interactions between cars and trucks.

We also added the constraint that there are 2 normal lanes in each direction of a road section and another left turn lane on the same road section. In part 1, we wanted to simplify the simulation and our architecture by having only one lane in each direction where cars would go straight, turn right and turn left. This way, we didn’t have to deal with lane changing logic or left turn lights. However, we added two lanes in Design Studio 2 Part 2, giving more realism to our simulation and a completely new dimension of traffic, whereas our one lane design ignored lane changing before intersections to turn, vehicle passing (albeit due to the addition of trucks), and separate traffic conditions for left turns. For example, vehicles can now enter the left turn lane from the left normal lane, and they can only turn left from the left turn lane. The left turn lane has a capacity of vehicles that is one-third of a normal lane from the same road section. Left turning is controlled by a special left turn traffic light signal that is distinct from the normal traffic light. All constraints about the normal traffic light can be assumed for the left turn traffic light, except that it controls left turning instead of the forward traffic flow. Vehicles can turn right on the right normal lane, when the normal traffic light is green or when no other vehicles are entering the same road section that the right-turning vehicle is trying to enter when the normal traffic light is red. Adding these new traffic interactions improves the representation of traffic nuances and special rules of traffic that can be applied to our simulation.

Road section closure may happen in the new design. Because the vehicles will not know at the time the road section is closed and route is calculated when the vehicles are added to the roads, they will drive along their preset routes. When they enter an intersection, they will check if the road section ahead is available, and if they find the road section is closed, their route will be recalculated at this time given the addition of road section closures during the simulation.

Road section closures can be added in two ways, manually by the user, or randomly during the simulation running (with a certain probability). In both cases, the section of road must be completely free of cars. Road section closures will always close all lanes of a section of road, so partial closures, such as closures of partial lengths of sections of roads and specific lane closures will not happen.

Changes to General Design and Approach 3 Design

- Audience
  - Students that take Professor E's civil engineering courses at UCI
  - Professor E
- Stakeholders
  - Actual stakeholders/investors
  - UCI
  - Tech Support at UCI
  - Government
  - Grants
  - Other Professors/Faculty
- Goals
  - Intuitive and easy to use - especially for people with less experience with technology
  - Should prevent input that creates unwanted behaviors (e.g. green light on all sides)
  - The simulation program should encourage users to explore different approaches or settings of traffic lights, by providing appropriate feedback to the existing setup such has problems with the setting or suggestions for improvement
  - Provides a broad view of traffic condition in simulation
  - Accurately records and simulates traffic movement and behavior for a large group of vehicles on the road
  - ~~Finds the optimal solution or combination of traffic light settings that makes all of the traffic present the most efficient~~
  - Tutorial to understand how the program works and the statistics
  - Users can give the name to a car for representation.
  - Shows detailed overall statistics
    - Average wait time of Cars
    - Average wait time of Trucks
    - Average wait time of combined vehicles
    - Number of Cars/Trucks
  - Shows intersection statistics
    - Average wait time of Cars
    - Average wait time of Trucks
    - Average wait time of combined vehicles
  - Be able to pause, play, fast forward, one step further the simulation
- Constraints

- Program is a webapp
- Each traffic light simulation program can only have up to 4 simulations running at the same time
- User can add vertical or horizontal road sections to the program
- Optional sensors may be installed on intersections to detect the presence of vehicles
- Traffic lights have a default timer
    - When traffic light timer runs out, turn another light green given sensors - sections of road with more cars go first, tiebreakers have random result
    - Users can change the timer of specific traffic lights.
    - Sensors can change the timer of its corresponding traffic lights
        - When there are no cars in the road section while the light is green, change the timer to 2 seconds (given timer is above two second), allowing the light to turn immediate yellow and then red
    - If cars on one section of road wait a maximum amount of time or more, the next light that turns green will automatically be the lights for that section of road
- User can manually change the state of the traffic lights
    - The ability of setting the light to yellow then red if it is green currently, and setting the light to green if it is red currently
- Program has to be able to run on an average computer or laptop
- No complications - accidents, pedestrians, bridges, road section closures, other incidents
- Program has to be able to run on major browsers
- ~~Cars move at the same speed~~
- Cars must be able to turn
- Road sections are horizontal or vertical
- Intersections are four way
- Every intersections must have traffic lights
- Students can control incoming traffic density
    - With timer by default - how many cars enters from edge of road section per time interval
    - With timer by default - how many trucks enters from edge of road section per time interval
    - Default timer preset is normal traffic density
    - Available traffic density presets (from least busy to most busy):
        - Not used → seldom used → normal → busy
    - Can also add cars manually
- All road sections and intersections have to fit within a rectangular area, of a size that can be determined by the user
- All movements or passing of time is associated to the system clock, which invokes the tick() method that updates map conditions and traffic at each tick of the clock

- Conditions and statistics of all road sections and intersections/ traffic lights can be viewed, but each one can also be selected at a time to be viewed in detail
- The route must flow with no illegal movements
- Teacher will go through training for program and be able to teach students how to use the program
- Road sections will have three traffic lanes for each side
    - Two straight lanes and one left turn lanes
    - Six lanes total for both directions
- Road sections will have left turn lanes at intersections
    - Capacity of vehicles in a left turn lanes will be ⅓ of a normal lane of the same road sections
- Implements left turn traffic lights that control traffic for only a left turn lane
- Trucks enter on the right lane by default (random for cars)
- Cars will be able to change lanes, and there should be two types of lane changing that can happen concurrently:
    - Changing lanes to turn, with the following conditions:
        - For cars
            - Should begin attempting to switch lane 1 intersection away from turning
            - There are no vehicles immediately next to the car
                - The car merges immediately
            - If there is a vehicle immediately next to the car attempting the merge
                - Merging car will slow down to a speed below that of the vehicle immediately next to the car
                - If there is a vehicle within a distance threshold behind the car next to the merging car
                    - That vehicle will slow down to a lower speed than merging car (which is already slowing down) and the merging car will merge when there is space
        - For trucks
            - Same as cars, but
                - The merging truck will not slow down
                - The distance threshold is greater as trucks are bigger and need more space to merge
    - After the merging car successfully merges, all cars will revert to original top speeds
    - Passing a truck in front of the car, with the following conditions:
        - If it is is immediately behind a truck for a certain amount of distance
        - It is not going to turn right or left in the upcoming intersection
        - There are no vehicles immediately next to the car
        - There is a certain amount of space in front of the truck

- Road section closures are possible
    - Can be generated randomly or added manually by users
        - Randomly given section of road has no cars
    - Vehicles will recalculate its route once it is in front of a closure
        - Vehicles will check if section of road ahead (or the next section of road on its projected route) is closed
- Users can save the simulation and reproduce it.
    - Road layout
    - Traffic light state
    - Sensors
    - Traffic Light Timers
    - Statistics
    - Vehicle data
        - Type
        - Location
        - Route
- Users can run multiple simulations at once and split the screen to see simulations running simultaneously
- Vehicles can turn right on the right normal lane, when the normal traffic light is green

- Assumptions
    - Software is only used for simulation in a university class
        - Does not have an impact in real-life traffic
    - Students have prior experience with technology and basic technical expertise
    - Cars will not pass through intersections if the road section the car is supposed to turn onto is full
    - Assume map area is rectangular
    - Assume the simulation is run with more than one intersection and more than one road section
    - Assume user has basic knowledge of traffic and infrastructure
    - Assume user already has a layout plan for the road sections and intersections
        - Software allows for the construction of road sections and intersections that the simulation will run on, but no presets will be provided
    - Cars are always faster than trucks
    - Trucks will never pass cars or other trucks.
    - All cars and trucks have the same speed respectively. They always drive at this speed when possible.
    - Vehicles will always slow down to allow another vehicle ahead of it to merge into its own lane
    - Vehicles immediately behind other vehicles will go the same speed as the vehicle in front (if possible given top speed)
    - The sensor in the intersection can detect how many vehicles are in the left turn lane and in the straight lane.

        - Assume road section closures close all lanes on a section of a road in at least 1 direction
        - Trucks are larger than cars

Personas

- Learners
    - Students looking to learn about traffic flow in a more immersive manner
- Instructors
    - Instructors looking to teach traffic flow through a visual demonstration
    - The time in class is valuable, so the interaction needs to be simple to save time during class.
- Technology literacy
    - People who have a basic knowledge of technology but not necessarily technologically skilled
- School administrators
    - Education institutions who are looking to expand their civic engineering curriculum

Experiences

We hope that students will be able to use the simulation to learn about the theory of traffic flow and use it as an educational tool that aids in understanding the material. We also would like for students to be able to use the application with little experience required to adjust the timing schemes for each traffic light/intersection. The students should also be able to save previous simulations to be able to compare differences between separate simulations with different settings, therefore observing the effects each simulation has on traffic flow. Overall, the application should be easy to use and easily implemented in an educational curriculum for this subject.

For the instructor, we hope the simulation will have the same features as the students in order for them to use it live while in class. Since we assume the simulation is a web application, it should be simple to stream it to a projector and make changes for students to observe during the lesson.

For the Administrators, we hope the simulation will be seen as an excellent educational tool that they will approve for use in the classroom. This program can be implemented into the current curriculum rather easily since no prior setup or knowledge is required, therefore administrators will be able to leave the use of it up to the professors. Overall, the traffic simulation is an additional strategy for teaching the theory of traffic flow, and should be seen as such.

Pseudocode

In each tick, the simulation will generate road section closure randomly. If a road section is closed, it does not allow vehicles entering it, but vehicles can still go out from it. After that, vehicles will be added to the outer road sections depending on the traffic density presets. Then, loop through all the vehicles in the simulation. For each vehicle, check if the vehicle needs to turn in the next intersection and it is in the wrong lane (left lane for right turns and right lane for left turns). If so, start the lane changing behavior. If there are no vehicles immediately next to the merging vehicle, then the vehicle immediately merges. However, if there is a vehicle immediately next to the merging vehicle, check if the merging vehicle is a car. If the merging vehicle is a car, then the merging car slows down to a speed below the vehicle next to it. Then, check the distance of the vehicle immediately behind the vehicle next to the merging car. If the distance of the vehicle immediately behind the vehicle next to the merging car meets a certain distance threshold predicting car collision upon a lane change, then that vehicle slows down to a speed below that of the merging car. If the merging vehicle is a truck, however, the truck remains the same speed and the program checks if the vehicle behind the vehicle next the merging truck meets a distance threshold predicting truck collision upon a lange change. If the distance threshold is met, then the vehicle behind the vehicle immediately next to the merging truck slows down to a speed below that of the merging truck. After all speed changes happen for merging cars and trucks and vehicles involved in the merging process, when there is space available for the merging vehicle, the merging vehicle changes lanes and all vehicles involved in the merging process revert to their original top speeds. Next, if the vehicle is a car, also check if the car is eligible for truck passing. If the car is immediately behind a truck and meets a certain distance threshold behind the truck, the car is not going to turn within the next intersection, there are no vehicles immediately next to the car, and there is a certain amount of space in front of the car for the car to merge, then pass the truck. Then, loop through all intersections and find the lights that ~~is~~ are currently running (yellow or green). ~~Then~~ For each light, decrement the timer. If the timer hits 0, turn the running light red, then check all other lights in the intersection to check if the maximum waiting time is hit on any of them. If so, change that light to green, reset it's maximum waiting time and reset the current traffic light's timer. If there are no lights that have hit the maximum waiting time, check the sensors on other intersections. Following this, turn another traffic light where ~~cars~~ vehicles are waiting green based on the car count for lanes controlled by traffic lights and reset the current traffic light's timer. If the timer hits 1, turn the running light yellow. Within the intersection loop, get the running lights again, since the running lights may have changed when a previous running traffic light's timer turned red and it's timer hit 0 - turning another traffic light green. ~~Within this traffic queue (which holds the cars queue for the section of road controlled by the traffic light), find the first car.~~ For each light, loop through the lanes controlled by that light to get the traffic queue (which holds the vehicle queue for the road section in the lane). Then, loop through the vehicles of the traffic queue. For the first ~~car~~ vehicle of each lane, check it's route to make sure the road section it is navigating to after passing through the intersection is open. If it is not, recalculate the car's route to avoid the road section closure. Then, check it's route and whether it moves right, left, or straight. ~~Cars~~ Vehicles without a set route will have an automatic route of going straight their whole route unless their routes are recalculated due to a road section closure. If the road section ahead in which the first ~~car~~ vehicle is heading is clear, then the ~~car~~ vehicle moves into that road section. If the road section is not clear, the first ~~car~~ vehicle stays. For other cars in the traffic queue (looping through from front to back), check how far the ~~car~~ vehicle in front of each respective ~~car~~ vehicle is. If there is space in front of that ~~car~~ vehicle, move the ~~car~~ vehicle forward by a one second increment based on the speed of the vehicle. If there is no space, do not move the ~~car~~ vehicle forward. After this, for every ~~car~~ vehicle in the traffic queue (including the first), check if the ~~car~~ vehicle is out of the Map range (the size of the whole simulation including all intersections and road sections). If so, then delete the ~~car~~ vehicle from the simulation. Next, update the ~~average~~ statistics of the average car wait time for every intersection by getting all cars ~~(from the traffic queues)~~ and their waiting times, the average truck wait time for every intersection by getting all trucks and their waiting times, and the average vehicle wait time for every intersection by getting all vehicles and their waiting times. This is possible by looping through all lanes controlled by the traffic light and getting their vehicle queues, which allows you to check vehicle types and wait times for individual vehicles. ~~At the end of each tick, update the average waiting time of all cars and car waiting times through the car array in the program interface.~~ At the end of each tick, update the overall statistics for the whole simulation for average car waiting time by getting all cars and their waiting times, truck waiting time by getting all trucks and their waiting times, and average vehicle waiting time by getting all vehicles and their waiting times. This is possible by looping through the vehicle array within the statistics class, contained within the simulation state (a running simulation instance), and getting all vehicle types and waiting times in the simulation. Finally, save data to a file in case the user wants to return to the simulation.

```
tick() {
    Generate road section closure randomly given no cars on the road section
    Add vehicles to the simulation by traffic density timer randomly to outer road sections
    For each vehicle {
        If the vehicle needs to change lane to turn in the next intersection and it is in the
        wrong lane {
            If (There are no vehicles immediately next to it) {
                Merge immediately
            }
            Else {
                If merging vehicle is a car {
                    Merging vehicle slows down to a speed below the vehicle
                    that immediately next to it
                    If there is a vehicle behind the vehicle next to the merging
                    car that meets a certain distance threshold for car
                    collision{
                        The vehicle behind the vehicle next to the merging
                        car slows down to a speed below that of the
                        merging car to make a gap for the merging
                        car to change lanes
                    }
                }
                Else {
                    If there is a vehicle behind the vehicle next to the merging
                    truck that meets a certain distance threshold for truck
                    collision{
                        The vehicle behind the vehicle next to the merging
                        truck slows down to a speed below that of the
                        merging truck to make a gap for the merging
                        truck to change lanes
                    }
                }
                When space is available to merge, merging car changes lane
                All  the vehicles revert to original speed
            }
        }
        If (The vehicle is a car &&
        It is immediately behind a truck within a certain distance threshold &&
        It is not going to make turn in the next intersection &&
        There is no vehicles immediately next to this car &&
        There is a certain amount of space in front of the truck for the car to merge){
            Pass this truck
        }
    }
    For each intersection {
        Get runningLights
        For each light in runningLights {
            Check timer on ~~runningLight~~light
            Decrement timer
            If timer hits 0 {
                Traffic light turns red
                Bool maximum_waiting_time_hit = false
                Check all traffic lights in the intersection {
                    If maximum waiting time is hit on a traffic light {
                        Turn that traffic light green
                        Reset maximum waiting time on that traffic light
                        Reset timer on this traffic light
                        break
                    }
                }
                If maximum_waiting_time_hit == false {
                    Check sensors and turn another light to green based on
                    car count for lanes
                    Reset time on this traffic light
                }
            }
            Else {
                If timer hits 1 {
                    Traffic light turns yellow
                }
                Else {
```

```
                Stay green
            }
        }
    }
    Get runningLights
    For each light in runningLights {
        For each lane in light {
            For each vehicle in ~~runningLight~~lane.TrafficQueue {
                If first vehicle {
                    Check Route
                    If road section is closed {
                        Recalculate route
                    }
                    If vehicle in front of the light is turning right {
                        If road section to the right is open {
                            Turn onto the road section
                            Reset vehicle wait time
                        }
                        Otherwise {
                            Don't move
                        }
                    }
                    If vehicle in front of the light is turning left {
                        If road section to the left is open {
                            Turn onto the road section
                            Reset vehicle wait time
                        }
                        Otherwise {
                            Don't move
                        }
                    }
                    If vehicle in front of the light is going straight {
                        If road section forward is open {
                            Go forward
                            Reset vehicle wait time
                        }
                        Otherwise {
                            Don't move
                        }
                    }
                }
                Else {
                    Check vehicle in front of this vehicle in queue
                    If there is space {
                        *Move vehicle and update location*
                    }
                }
                *If vehicle location is out of map range {*
                    *Find and delete vehicle from statistics vehicle array*
                }
            }
        }
    }
    *Update Intersection CarAverageWaitTime for all cars*
    *Update Intersection TruckAverageWaitTime for all trucks*
    *Update Intersection AverageWaitTime for all vehicles*
  }
  *Update Intersection CarAverageWaitTime for all cars*
  *Update Intersection TruckAverageWaitTime for all trucks*
  *Update AverageWaitTime for all vehicles*
  *Save data to file*
}
```

We are assuming that no cars are added, and there are already intersections, road sections, and sensors for each traffic light, and cars in the intersection. Cars can have a set or default route.

UML Diagram

As shown in **Appendix 1**, our UML diagram contains the necessary classes for the Traffic Signal Simulator program. The following is a description of the classes and relationships between these classes:

- User Interface class contains an array of every simulation state instance, and methods to retrieve or remove simulation states from the array. This class essentially contains all of the data needed to display and run the traffic light simulations. Each User Interface Object can have 1 to 4 simulation state objects in its array (as specified in our restrictions).
- ~~Program Interface~~ Simulation state class contains some global variables that are needed by other classes such as the system clock and the visual interface, as well as methods that control behaviors of the entire simulation, like run() and tick(). It also contains methods that can alter attributes and behavior of the Map and Statistics class objects. It is the center of the program that holds essential objects to the program. The incoming traffic densities of cars and trucks are also specified in this class. The specified densities will be true for all road sections within the simulation state objects. Each instance or object of the simulation state class represents a stand-alone simulation. Each Simulation State object contains exactly 1 statistics object and exactly 1 map object, but 1 or more Simulation State objects can be stored in the same database.

- Map class maintains most of the objects within a map, which includes intersections, road sections, traffic lights, and sensors. The ~~two~~ main array~~s~~ it has ~~are~~ the Intersection array ~~and the Road array~~, which is an exhaustive array of all intersections ~~and roads~~ within the map object~~, respectively~~. It also specifies a map range which describes the dimensions of the rectangular area of the map. Each Map object contains 1 or more Intersection objects and 1 or more Road Section objects, each Map object is also contained in a Simulation State object.
- Intersection class models traffic intersections, which is why it contains traffic light and sensor objects associated with the intersection itself. A mapping is used to associate traffic lights with sensors, in which the sensor field will be NULL if the sensor option is not enabled by users. Other attributes within the intersection class include its location on the map, the average wait time associated with the intersection, and an ~~attribute~~ array that keeps track of the running traffic lights (straight or left turn traffic light with green signal at the moment) out of its ~~four~~ eight traffic lights. All of its methods are related to updating or retrieving information from the class object attributes. Each Intersection object contains exactly 1 Location object, 0 or more Sensor objects, exactly 8 traffic light objects. Each intersection is also contained in a Map object and can be associated with vehicle objects.
- Traffic light class specifies the possible light signal colors (red, yellow, and green) ~~and maintains a queue that contains vehicles waiting at the traffic light or arriving at the traffic light on the same road.~~ ~~The order this queue stores vehicle class objects is relative to the order vehicles are lined up on the road.~~ This class also contains a timer attribute which is a counter that signifies the number of seconds left until the light should turn red, and counts down while the light is green. A traffic light will only turn red from green (and yellow) when the timer counter reaches 0, unless the user interferes with it. The timer counter will return to a preset value whenever the light turns green. The maxWaitTime attribute, on the other hand, is a counter that signifies the number of seconds left until the light should turn green, and counts down while the light is red. It will be reset to a preset value whenever the light turns green, and will only start counting down again when the light is red. The maxWaitTime attribute does not have to be 0 for a traffic light to turn green, it can turn green when another light turns red from green, but it should be noted that factors like the amount of traffic on a road section can affect the priority to which light turns green first. The maxWaitTime counter is made so that no road sections will have to wait for an unreasonably long time or infinitely long before turning green because of traffic conditions. The TrafficLightDirection attribute identifies which direction it is located relative to an intersection, so “N” would mean the traffic light is on the northern side of an intersection, “S” would mean the traffic light is on the southern side of an intersection, and so on. The TrafficLightType attribute specifies whether the traffic light is a left turn light or a straight traffic light. The LanesControlled array holds all of the lane objects associated to the light, and because of our restriction of 2 normal lanes and 1 left turn lane per road section, the LanesControlled array will have 2 lanes if the light is a straight traffic light, and 1 lane if the light is a left turn light. All of its methods are related to updating or retrieving information from the aforementioned light signal color enumeration and vehicle queue. Each traffic light object contains 1 or 2 Lane objects, it

is also contained in exactly 1 Intersection object and can be associated with vehicle objects.

- Sensor class is an optional class that models the behavior of a sensor at an intersection. ~~It specifies which intersection the sensor object is located at and has an boolean attribute that describes whether cars are present or not on a road.~~ The vehicleCount attribute stores the number of vehicles it is able to sense (on a road). All of its methods are related to updating and retrieving information from ~~the intersection and cars_present attributes.~~ the vehicleCount attribute. Each sensor object, if initialized, would be contained in an Intersection object.
- ~~Road~~ Road Section class models roads on the map of this simulation. It keeps track of the number of vehicles present on the road section and specifies the direction or orientation of the road section relative to an intersection. The LanesContained array holds all of the lanes that are within the Road Section object. The RoadClosed boolean attribute indicates that there is a road section closure at the road section when the attribute is true. RoadClosed is checked against when any vehicles try to enter or are being added to the lanes on this road section, and such behaviors are only allowed when there is no closure on this road section. It also contains a method that adds vehicles at the edge of a road section, a method that deletes vehicles from a road section, and methods that update its attributes. Methods that calculate the average speeds of cars, trucks, and vehicles overall can be found here as well. Each Road Section object contains 6 Lane objects (3 Lanes going in each direction), and is contained in an Intersection Object.
- Statistics class maintains information about the overall simulation, such as the exhaustive array of all vehicle class objects present in the simulation map, and the average wait time of all intersections. All of its methods are related to updating and retrieving its attributes, or computing specific values (~~minimum and maximum wait times~~ average wait time for cars and trucks) using its attributes. Each Statistics class contains 0 or more vehicles, and is contained in a Simulation State object.
- Vehicle class models 2 kinds of vehicles present in the map, cars and trucks, which can be added into the map by a timer or manually by users at edges of road sections. Each vehicle object specifies its location, route, total or accumulated wait time, and temp wait time (wait time at the current intersection). Vehicles will go straight until it reaches the end of a road section if a route is not specified. The speed attribute is the speed the vehicle is traveling at for the current tick, while the MaxSpeed attribute is a constant that specifies the maximum speed a vehicle can travel at. A name can be specified when a vehicle is created so it can be referred to by users who want to look at the behavior of a certain vehicle closely. The name attribute will be NULL if not specified. Each vehicle also notes its relationships with other classes, including the traffic light of the lane the vehicle is in, what type of lane the vehicle is in, other traffic lights within the same intersection (stored in a traffic light array), and the intersection the vehicle is currently in. Its methods are related to updating and retrieving values of its attributes. Each Vehicle object contains exactly 1 Location object and exactly 1 Route object. Each Vehicle object is also associated with exactly 1 Intersection object, exactly 1 Lane object, and exactly 1 Traffic Light object.

- Route class is used by the vehicle class and specifies the route that a vehicle will take on the map. It contains an array of directions, and each direction can only be one of the items in the enumeration {“forward”, “left”, “right”}. The method recalculateRoute recalculates the array of directions a route object has based on the current condition of the map, accounting for things like road section closures. Its other methods are related to updating and retrieving its attributes. Each Route class contains no other class objects. There are no outward associations from Route, but each Route object must be contained in a Vehicle object.
- Location class is used by several classes such as the Intersection class and the Vehicle class. It contains an array of integers that is of size 2. This array is the coordinates that describes the location of the object this location is associated with on a grid representation of the map. Its methods are related to updating and retrieving its attributes. Each Location object contains no other class objects. There are no outward associations from Location, but 1 Location object is contained in each of Vehicle objects and Intersection objects.
- Lane class is contained in the Road Section class as an array of Lanes taking the entire length of the road section along each direction and the Traffic Light class as an array of Lanes that the Traffic Light controls (either two straight lanes or one left turn lane). Lane is also contained in Vehicle to track the current lane that the vehicle is in, either straight lane on the right, straight lane on the left, or left turn lane. Lane contains an enumeration of {“Straight Left”, “Straight Right”, “Left”} which keeps track of the type of lane. Lane also contains an integer capacity, which tracks the total vehicle capacity of the lane. Left turn lanes will always have a capacity of ⅓ of straight lanes in the same section of road. TrafficMap is a mapping of vehicles of type Vehicle to Floats tracking the distance of the vehicle behind the vehicle in the key. TrafficMap keeps track of the cars in a lane and their distance from each other to track expected vehicle behavior like vehicle speeds, cars passing trucks, and lane changing. Methods of the Lane class include getters such as GetLaneType() and GetCapacity() which return the type of lane as an enumeration and the capacity of the lane as an Integer. AddVehicleToTrafficMap() adds a vehicle to the lane as a vehicle enters the lane through the previous intersection onto the section of road, and thus the lane represented by the lane object. It is also used when a car changes lane into this lane. RemoveVehicleFromTrafficMap() removes a vehicle from the lane when the car moves past the section of road through the next intersection or a car changes lane from this lane. Finally, ModifyTrafficMap changes the Float distance of vehicles behind each vehicle in the traffic map when any vehicles move in the lane or when any vehicles are added or removed from the lane. Each Lane class contains 1 or more vehicles.

Appendix

I. UML Diagram (Nov 22, 2021)
II. Design Studio 2 - Part 1 (Nov 15, 2021)
III. Traffic Light Sensor Behavior (Nov 20, 2021)
    A. Link to research about the available technology for traffic sensors