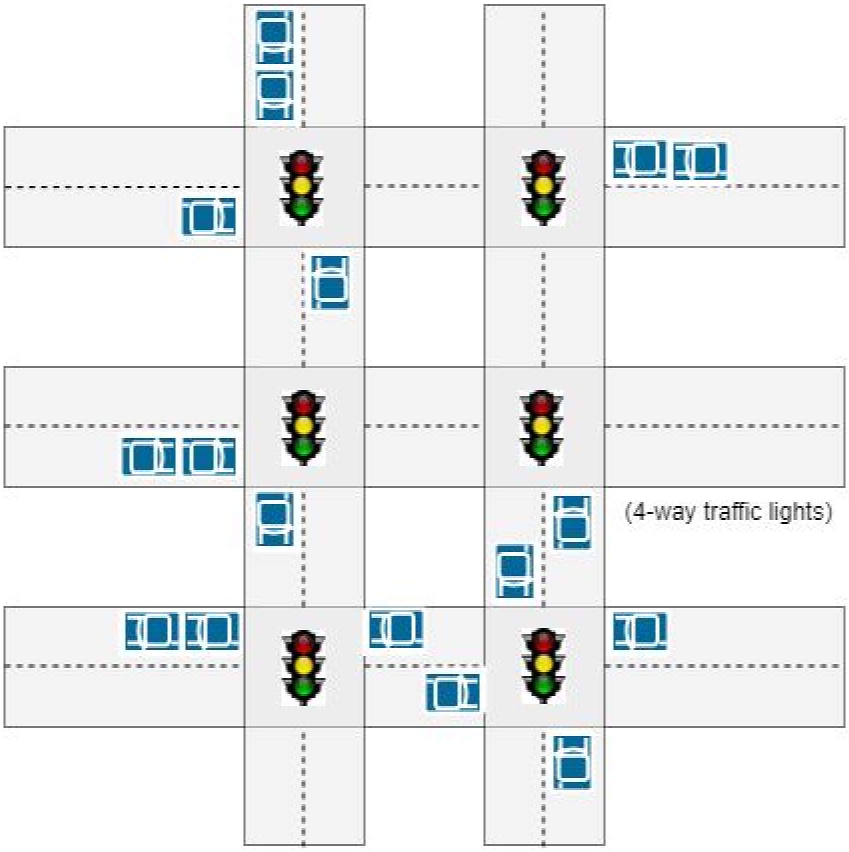


Design Studio 2

Allison Bennett #85571966
Ryan Fong #29212092
Akshita Nathani #93016780
Deone Peng #45954961
Yichen Wang #49271937

Essence

- Make it easy for students to learn about traffic signal timing by allowing them to “play” with different timing schemes in order for them to explore different traffic scenarios

Audience

- Professor E - would want to experiment with the simulator herself in order to explain to students how it works and how they would learn from it
- Professor E’s students (current as well as future) - would use the simulator to understand how traffic signal timing affects congestion
- Professor E’s TAs - would learn to use the software to answer students’ queries and understand best traffic signal timing to reduce traffic themselves
- Other Civil Engineering professors - would wish to adopt the traffic signal simulation software as a new learning tool for their students to learn about traffic signal timing
- Other Civil Engineering students - would use the same simulator to learn about traffic signal timing effects

Other Stakeholders

- City council members/planners - would use the traffic signal simulator to plan road and signal locations
- School faculty members - would approve the application to be used at the school
- UI designers - visual elements of the simulator would affect implementation details and vice versa

Goals

- Educate students on traffic signal timing and the factors that influence traffic congestion
- Enable students to experiment with various settings
- Simulate traffic conditions in a very controlled environment
- Enable students to observe traffic patterns given different signal timing
- Enable students to arrange a map with intersections and roads
- Enable students to change traffic signal timing
- Convey traffic levels in real time
- Enable students to change traffic density
- Enable students to choose for an intersection to have a sensor or not
- Advance cars at the same speed
- Allow for trucks that move slower than cars

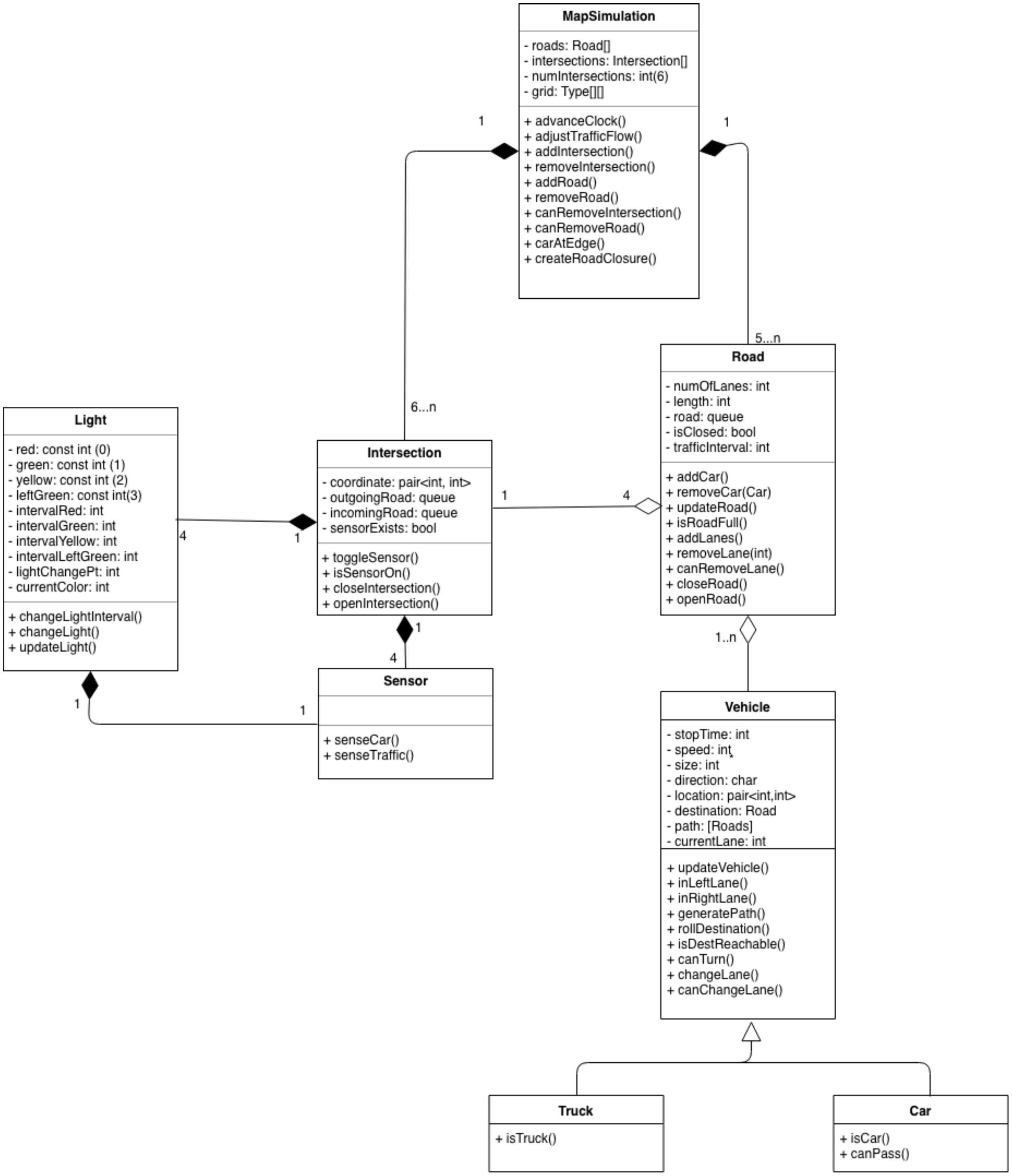
Constraints

- Cannot allow car crashes
- There must be at least 6 intersections
- Only one simulation can run at a time
- Every intersection must have a traffic light
- All intersections need to be 4-way
- Application must be simple - no road closures, bridges, etc.
- There must be an option to toggle sensors for cars at a light
- All roads must be horizontal or vertical

Assumptions

- We are able to use existing software packages providing relevant math functionality
- Our simulation will display individual cars on the roads
- This program will only be used for educational purposes
- The sensor waits for light cross-traffic and then changes light to green
- There are two types of roads, major and minor, that users can define by direction (North-South, East-West)
- Cars are not able to turn right on a red light

UML Class Diagram



UML Class Descriptions

Vehicle

A Vehicle is a parent class of Car and Truck, which will be defined later. All Vehicles have: stopTime, speed, size, direction, location, path, and currentLane. stopTime is an int that corresponds to how many clock ticks the car needs to stop when approaching a yellow Light. The direction is indicated by ‘L’, ‘R’, ‘U’, or ‘D’ (left, right, up, down). It is used to increment the x or y values of the location pair coordinate upon moving forward. The size refers to how many spots it occupies in the queue and speed is the amount of ticks it takes to move forwards.

When adding a Vehicle to the map, rollDestination() randomly chooses an end Road for the Vehicle’s destination. (Random is used from an external library) It then checks if the destination is reachable with the method isDestReachable(). A destination should not be reachable if there is a road closure blocking the route. If it is reachable, a Vehicle will use generatePath() to choose a random, valid path to the destination. Note that a Vehicle is added onto a random Road of the MapSimulation.

The path generated will dictate when a Vehicle will turn left, turn right, or go straight. If a Vehicle needs to turn left, the program must check all of these conditions:

- The Vehicle must be in the left lane (use currentLane())
- The Vehicle must be at a green left arrow Light
- If the Light is Yellow, the program must check if the Vehicle can pass
- There must be space in the destination Road if the Vehicle turns left

If a Vehicle needs to turn right, the program must check for all of these conditions:

- The Vehicle must be in the right lane (use currentLane())
- The Vehicle must be at a green Light or yellow Light
- If the Light is Yellow, the program must check if the Vehicle can pass
- There must be space in the destination Road if the Vehicle turns right

If a Vehicle needs to turn left or right but is not in its correlated turn lane and is approaching the end of the Road, it will call canChangeLane() which returns true if the Vehicle has space to change lanes and will change lanes by calling changeLane(). If a Vehicle cannot change lanes and misses its turn, it will generate a new path to its destination.

The method updateVehicle() utilizes the speed, stopTime, direction, location, and destination to update the Vehicle’s status at every clock tick. All Vehicles can either move forward, stop in traffic, or turn.

Car

A type of Vehicle and is faster than Truck, takes half the space in the queue and has a faster stopTime. If a Car is behind a Truck, it will call canPass(). The method will check if the Vehicle in front is of type Truck, call canChangeLane(), and if both are true then canPass() will return true and the Car will call changeLane().

Truck

A type of Vehicle is slower than Truck, takes double the space in the queue and has a slower stopTime. It has the method isTruck() that returns true if it exists. The method is used in the Car’s method canPass().

Road

A Road is initially represented by two queues, with only two lanes and a set integer length. The Car's addCar() and removeCar() methods are called by MapSimulation's advanceClock() method. The updateRoad() method is called by MapSimulation's advanceClock() to move the cars forward at each clock tick. isRoadFull() is used to check whether the road has space to add a car. If the Road is full, it's because of severe traffic backup caused by bad light timing.

The value of trafficInterval represents the amount of ticks before a Car is added to a Road. When users set a custom amount, if the input is, for example, less than 3 it will be considered heavy, in between 4 and 7 inclusively will be considered medium, and any more than 7 will be considered low. The program will have a default trafficInterval for all of the Roads upon execution but the user will be able to specify which Road will have heavy, medium, or low traffic density.

Users are able to specify whether or not a Road can be closed. This will call the methods closeRoad() and openRoad(). Users are also able to add and remove lanes, but will call canRemoveLane() to ensure that the Road has at least two lanes. Closing and opening a Road can only be toggled by the user before running the program.

MapSimulation

MapSimulation is represented by a grid, a 2d array, that contains roads, an array of Road, and intersections, and array of Intersection. Both roads and intersections are arrays that contain its Type. The attribute numIntersections is initialized to 6 to meet the constraint acknowledged from the document. The attribute numRoads is initialized to 5, since the map will be defaulted upon execution. The class MapSimulation allows users to add or remove an Intersection or a Road. When attempting to remove an Intersection or a Road, MapSimulation will call the method canRemoveIntersection() or canRemoveRoad() which returns true if there are 7 intersections or roads and false otherwise.

This class is also responsible for adjusting traffic flow with the method adjustTrafficFlow(), which is specified by the user. Depending on the traffic flow (e.g. heavy = 3, medium = 7, low = 15, or a custom amount), the attribute trafficInterval will be set accordingly.

advanceClock() increments the clock tick by 1. When advancing the clock tick, MapSimulation will check if there are is a Car at the edge of the map with the method carAtEdge() to determine when to remove a Car.

Intersection

Intersection tracks all necessary information to display to users what is happening at each intersection. Coordinate represents the place on the map where the intersection is located so we may also keep track of which queues are incoming and outgoing at each intersection. Incoming roads are roads where cars are moving towards the intersection and outgoing roads are roads where cars are moving away from the intersection. Each intersection has a Boolean variable called sensorExists that indicates if the user has enabled sensors for that specific intersection. Function toggleSensor turns sensorExists to the opposite value allowing sensors to be turned on or off for the specific intersection and isSensorOn checks the value of the variable sensorExists.

User can close and open an Intersection. Closing an Intersection will affect the two perpendicular Roads. Closing and opening an Intersection can only be toggled by the user before running the program.

Light

The light class consists of a set of integer variables and a function used to update the status of the lights. The status of Lights are represented by integers 0(red), 1(green), 2(yellow), 3(left turn green) stored in currentColor variable. The duration of each light color is represented as integer variables named IntervalRed, IntervalGreen, IntervalLeftGreen, IntervalYellow. lightChangePoint is an integer variable that indicates what time the light should change to the next color. The time is based on the global smart clock. To determine whether the color should be changed, the changeLight() method looks at the currentColor variable. For every tick, this function will be called to decide to keep the light same or change into next color.

Sensor

A sensor has no member variables, only two functions. A sensor can senseCar to see if a car is activating the sensor and senseTraffic which checks if the cross-traffic is light enough to be able to change the light of the sensed car from red to green.

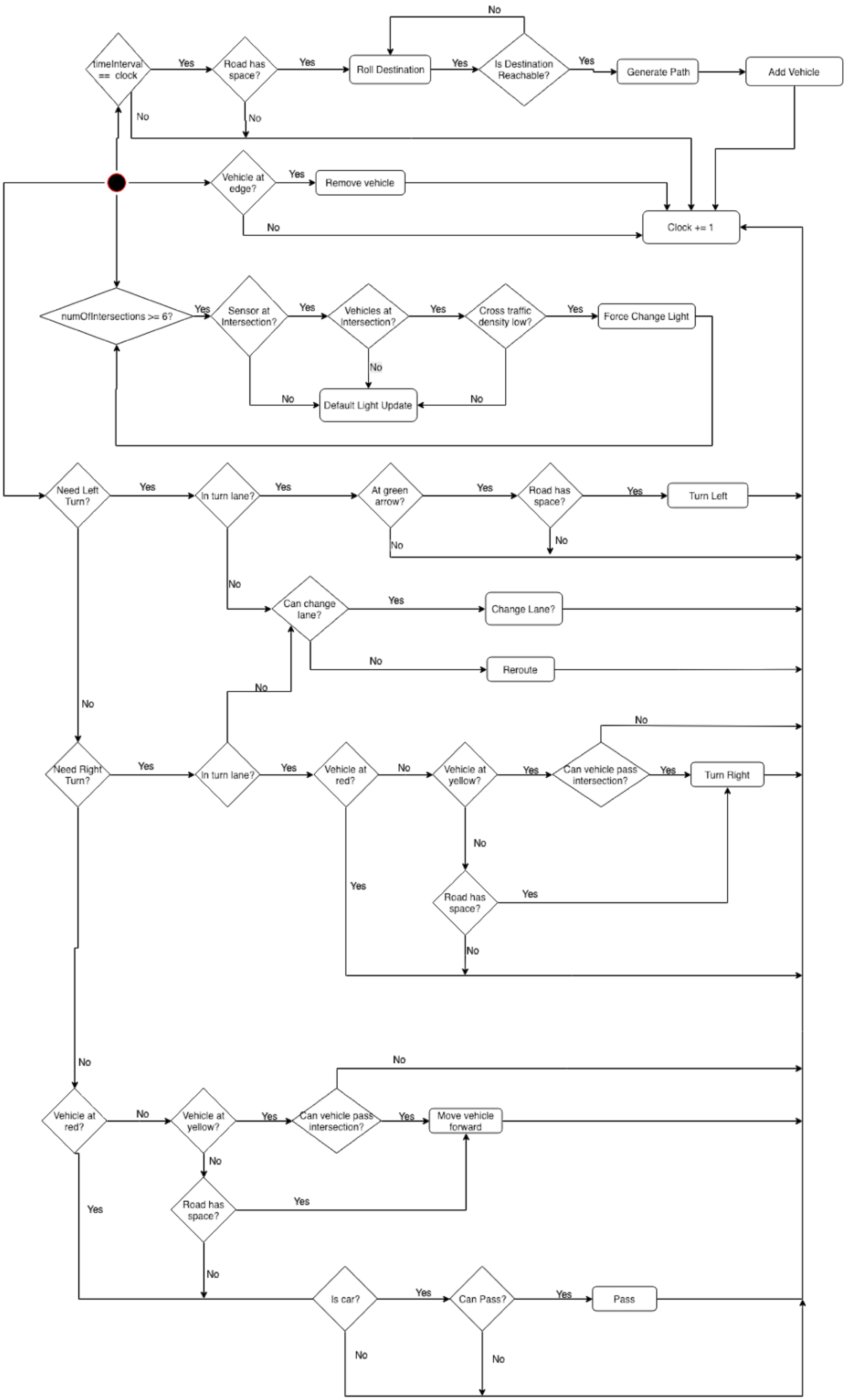
UML Class Relationships

- Intersection---◇Road
  - Intersection is a part of Road. An Intersection has four Roads, representing all four possible directions.
- Vehicle---◇Road
  - Vehicle is a part of Road. A Vehicle can exist without a Road. A Road can have 0 to many Vehicles.
- Vehicle is a parent of the classes Car and Truck
- Intersection---◆MapSimulation, Road---◆MapSimulation
  - MapSimulation is composed of both Intersection and Road. Intersection and Road consist and has the other classes in the UML Diagram. Intersection is composed of Sensor and Light. Road has Vehicles on it and can have an Intersection once two Roads cross. The MapSimulation must have at least six Intersections.
- Sensor--◆Intersection
  - Intersection is composed of a Sensor since a Sensor has no purpose without being at an Intersection, so it cannot exist without an Intersection. Each intersection has four sensors, one for each Road.
- Light--◆Intersection
  - Intersection is composed of Light since a Light has no purpose without an Intersection, so it cannot exist without an Intersection. Each intersection is composed of four Lights.

Advance Clock Tick Algorithm

- Check Sensors for traffic at each Intersection (if user determines that Sensor exists at Intersection)
  - Forcibly change Light if appropriate
    - Light change for Sensor is appropriate if Car is sensed and there is *light cross-traffic*, therefore safe to change the Light where Car is sensed (see changeLight pseudocode in Light class)
- Default Light update (see changeLight pseudocode in Light class)
- Vehicles enter MapSimulation
  - User specifies traffic level for a particular Road
    - If there is space, add a Vehicle to the specified Road
  - Before adding the Vehicle, the program will choose a destination randomly, check if it is a reachable destination, and then generate a path to get to the destination.
    - If the randomly chosen destination is not reachable, loop until a valid destination is chosen.
- Vehicles leave MapSimulation
  - Use Road length, speed of Vehicle, and traffic level to determine when Vehicle leaves MapSimulation
- Update existing Vehicles
  - Vehicle moves to another queue when crossing an Intersection by going forward, turning left, or turning right
    - Only move if Road is not backed up and if not at a red Light
    - Vehicles cannot turn left or right at a red Light
  - Vehicle stops if at red Light or not enough time to cross Intersection on a yellow Light
  - If the Vehicle is a Car and is behind a Truck, then it will change lanes and pass the Truck if it is safe to pass.
    - Safe meaning there is enough space adjacent and behind the adjacent spot.
  - All Vehicles can change lanes if they need to turn soon, but only if it is safe
- Increment clock one tick

UML Activity Diagram - Advance Clock Tick



Changes and Comparison

Key	**Strengths** **Weaknesses**

	Part 1	Part 2
Cars & Trucks, passing behavior	One lane in each direction, no passing. Car class has *speed* and *stopTime* variables, to account for different speeds. Didn't consider vehicles of different types. Didn't consider passing lanes because different vehicle types were not allowed, and all cars went the same speed.	(difficulty: easy) Added second lane for passing. Created parent Vehicle class with Car and Truck subclasses. Added *size* to Vehicle. *speed* and *stopTime* now useful to specify differences between Car and Truck
Left/Right Turns	No turn capabilities. Split road into multiple chained small queues instead of one long queue. Allows for easy pops and adds if turning becomes required.	(difficulty: hard) Maintained same *Road* queue layout, but needed to add second lane for designated left turns. Right turn on red is prohibited. Right turn only permitted on green and yellow if road has space. Previous design choice of separate queues allowed for easy implementation of turning. Pop *Vehicle* off one *Road* queue at intersection, then add it to any of the other three *Road* queues going in the direction of the vehicle.
Road Closures	No consideration for road closures	(difficulty: medium) Road closures have to be created before simulation begins. Create route to destination upon adding car. If road closure makes destination not reachable, create new destination and route.
Multiple simulations side-by-side	MapSimulation class instance to encompass simulation objects (roads, cars, intersections, sensors). Allows for easy modularization.	(difficulty: easy) Same global clock for all MapSimulation. Single Stop and Start buttons control all MapSimulation. When user attempts to modify one, they all pause. MapSimulation instances have their own separate roads, vehicles, intersections. Each simulation's advanceClock() call *waits* for the other simulations to finish before changing all states.

Save/retrieve simulations	No consideration for save and retrieve capabilies	(difficulty: easy) This application should allow users to save the current traffic status and retrieve it at another running. This is to say, the values of attributes of all classes can be saved(i.g. Car’s position, speed, traffic light status, intersection status,etc). When users want to recover the simulation to the saved status, all data will be loaded from the save.

Reflection

Cars, Trucks, and passing behavior

Our part 1 design only had one lane in each direction. This made passing behavior impossible. However, since all cars were initially going at the same speed, passing wasn't desirable. To account for the possibility of cars moving at different speeds in part 2, we included variables speed and stopTime. In part 1, we initialized all cars to have the same speed and stopTime. Our initial design choices made part 2 in this category fairly easy to implement. Part 1 only had a Car class to represent moving objects. In part 2, we altered our approach and created a parent Vehicle class with two subclasses, Truck and Car. Now our speed and stopTime variables won't be the same for all moving objects. Cars have faster speeds than Trucks. Cars also have shorter stopTimes. If new vehicles will be added in the future, such as Motorcycle, we can create additional subclasses to the Vehicle parent class. We added a second lane to enable passing behavior. Cars going straight can pass Trucks turning left. Trucks can't pass at all, but they are allowed to change lanes. Vehicles can only change lanes if the neighboring lane has space (empty space to the right/left and behind the right/left space).

Left/Right Turns

In part 1, we didn’t support turning. We only allowed Cars to go straight through intersections. Although turning wasn't allowed, we chose a Road design that would make turning easier to implement. Instead of choosing a single queue to represent the entire road, we split the road into multiple queues connected by intersections. When a car passed through the intersection, it would pop off one queue and get added onto the queue directly across the intersection. When part 2 came along, instead of only adding the Vehicle to the road across, it could be added to any of the three roads. The Vehicle could turn right and get added to the right queue, or turn left and get added to the left queue. In part 2, we added a lane for protected left turns so traffic going straight doesn't slow down too much. Right turns on red are prohibited. They're only permitted on green lights and yellow lights if there's space. Although our initial multiple queue design made the actual turning action easy to code up (pop and add), deciding when turns were permitted was very difficult.

To be able to turn, a vehicle needs to be in the correct lane to make the turn needed, and the correct light needs to be displayed before the vehicle can turn. For example, to make a protected left turn, the vehicle needs to be in the leftmost lane and the green arrow light needs to be displayed for the vehicle. To make a right turn, the vehicle would need to be in the rightmost lane and a green light would need to be displayed for the vehicle. There are also many conditions that need to be met for the vehicle to be in the correct lane for the next “step” in the vehicle’s path (either moving forward, turning left or right) that need to be considered when allowing turning.

We considered this to be the most difficult task to include in part two because of all the conditions that need to be in place to allow for a vehicle to turn. We also needed to consider how and when a vehicle could change lanes to accommodate turning and what would happen if the vehicle was blocked from doing so. We decided to adjust a vehicle’s path if a vehicle was blocked from turning the way its original path needed it to.

Road Closures

In part one, we did not consider road closures because the specifications said no further complications should be introduced including road closures. In part two, we needed to work them into our design. At first, we tried to allow for road closures to be applied while the simulation is running, but we did not want the simulation to crash if the user closed all roads. Instead, we decided to allow for roads to be closed only before the simulation starts so we have more control as designers and warn users of improper closures. We decided to threshold the number of road closures so that there is at least one entry and one different exit point and that the entry and exit points are reachable to one another. If the user does not meet this threshold, we will then throw an exception and not allow the simulation to run until the error is fixed.

The second implication of road closures is when a closure affects a vehicle's destination and route to its destination. Before a vehicle is added to the simulation, we roll a random destination point for the vehicle (not including entry point of vehicle). Based on the vehicle's entry and exit points, we determine if the destination is reachable from the entry point. Then, we generate one of potentially many paths the vehicle can take from entry to exit point. If a road closure makes the vehicle's exit point unreachable, we roll another random destination point and generate a new path.

The way we designed part one did not prepare us at all for road closures, but we had no choice as designers about including them. Adding road closures to our design was a medium-level difficulty task only because we had to consider potential user behavior. Once we decided to only allow road closures to be added while a simulation is stopped did the task become easy.

Save/retrieve simulations, and Multiple simulations side-by-side

We didn't consider the Save/retrieve function for simulations in part 1 because we wanted this app to focus on its educational purpose. In part 2, we needed to give this new feature to this program. The newly developed function will allow users to save the current traffic status and retrieve it when they want. This is to say, the values of attributes of all classes can be saved in local disk (e.g. Car's position, speed, traffic light status, intersection status, etc). When users want to recover the simulation from the saved status, the system will provide two options to do that: either keep the current simulation and load data in the new window, or shut down the current simulation and load data in the same window. If the user wants to retrieve in the same window, a warning will pop up telling the user that the current simulation will be lost. After the user confirms the selection, the data will be loaded from the save state.

Users are also allowed to open multiple windows at the same time to run the simulations. This gives users the possibility to compare and to visualize the traffic simulation with different setting, which is more educational to students.