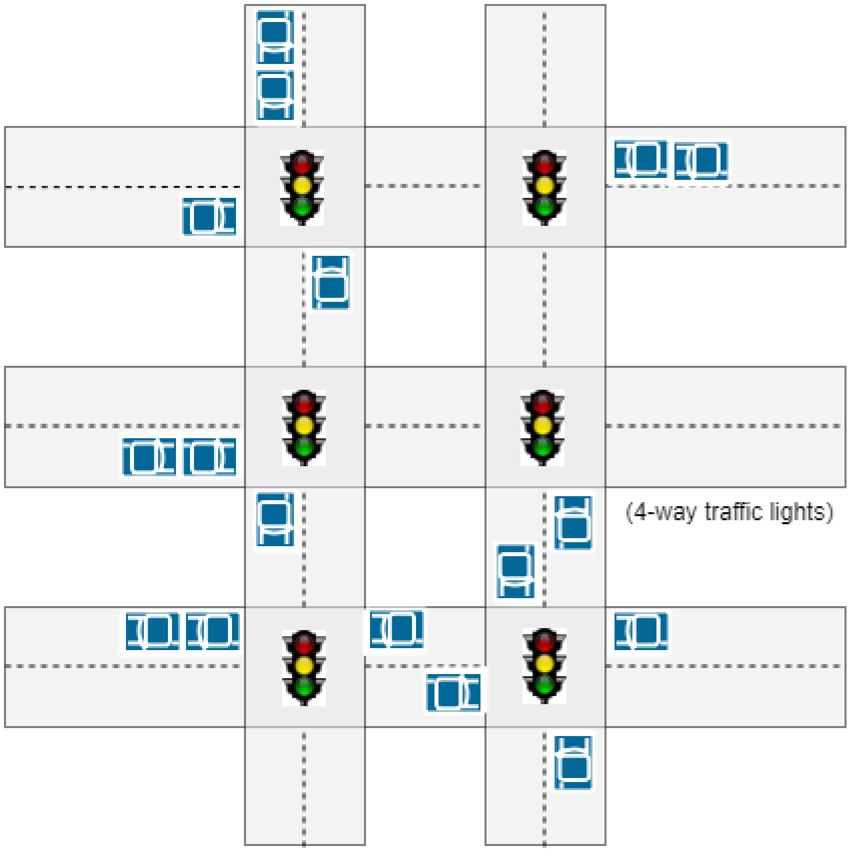


Design Studio 2

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

Constraints

- Cannot allow car crashes
- There must be at least 6 intersections and 5 roads
- 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

- Roads are single lane only
- We are able to use existing software packages providing relevant math functionality
- Cars come in and go out on the same road, never turning
- 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
- Cars are all the same in regards to type, size, speed, etc.
- There are two types of roads, major and minor, that users can define by direction (North-South, East-West)

Differences Between Alternatives

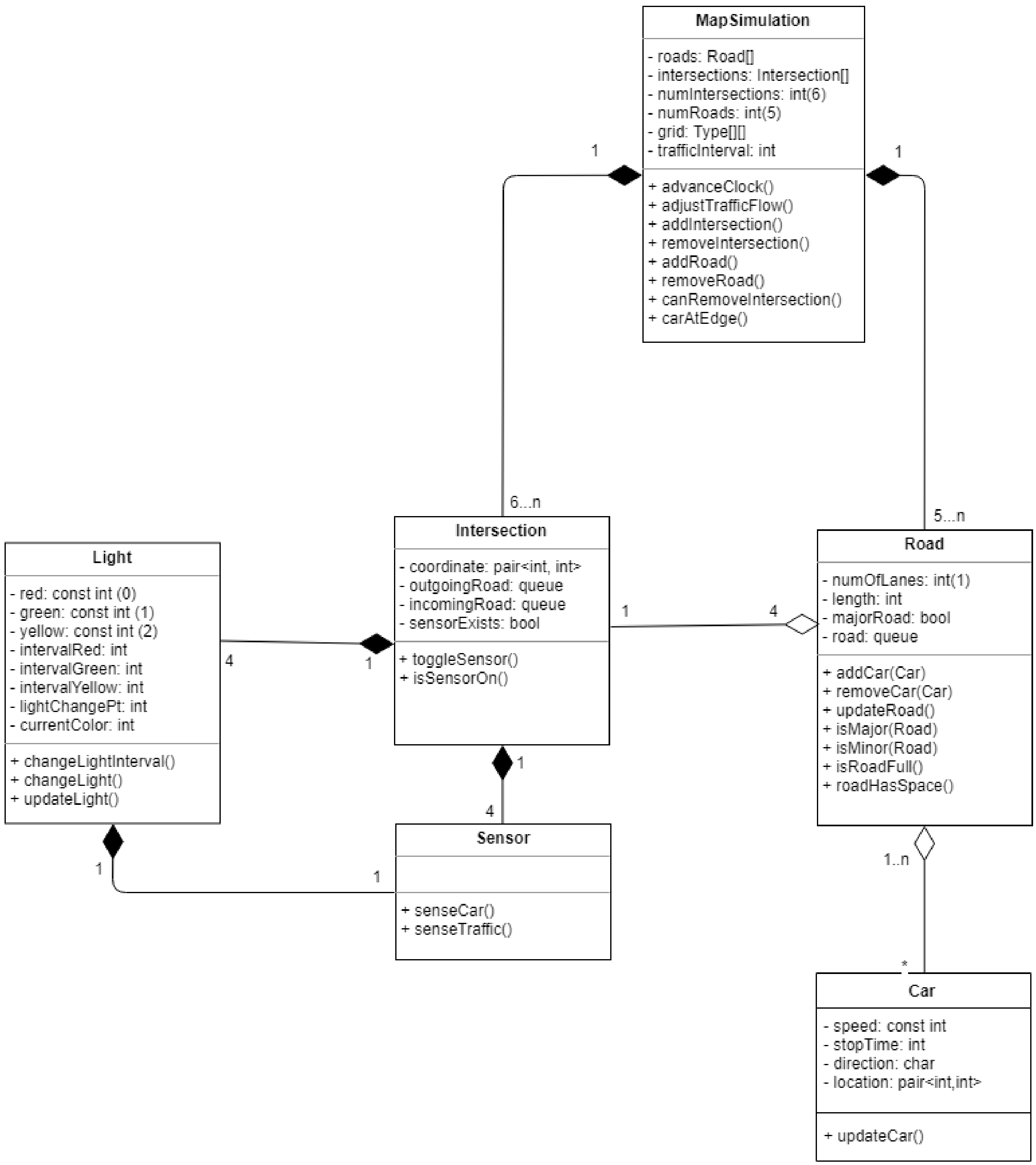
<table>
<tr><th></th><th>Chosen One</th><th>Alt #2</th><th>Alt #3</th></tr>
<tr><td>Light timing</td><td>Provide default light intervals to students, and also allow them to set their own values. When launching the program, for each light there will be 2 option buttons for light intervals: default values and entering values. If the users click on entering values, a box will promote users to input new time interval values.</td><td>Allow users to change timing of only one light (red, green or yellow); the program changes timing of the other two colors accordingly.</td><td>Allow users to change timing of all three lights by providing enough control.. There would be a minimum timing for each light color so users could not inadvertently cause crashes.</td></tr>
<tr><td>Light Scheme</td><td>Users choose what major and minor roads are; light timing changes accordingly</td><td>All lights for every road have same timing, set by user.</td><td>Individually set timing per light.</td></tr>
<tr><td>Light (Data Structure)</td><td>Light is a class. Each color is an attribute of the class.</td><td>Light is an array such as:

Light = [red, green, yellow]</td><td>Each light is a global variable.</td></tr>
<tr><td>Queue</td><td>The simulation has separate queues at each intersection.
→ this would be flexible to accommodate changes.
Eg: each red dash is a separate queue.
___|___|___|___
___|___|___|___
   | | |</td><td>The simulation has a continuous queue per row or column.
Eg: red and green are two separate continuous queues.
___|___|___|___
___|___|___|___
   | | |</td><td>(Same as alt #2)
The simulation has a continuous queue per row or column.
Eg: red and green are two separate continuous queues.
___|___|___|___
___|___|___|___
   | | |</td></tr>
</table>

Tradeoffs Among Alternatives

	Chosen One	Alternate #2	Alternative #3
Educational	More educational than the other alternatives since students have the flexibility to change the lights and see the aftermath.	Less educational as it gives the students less freedom with traffic lights, so they cannot see the full potential of traffic light changes	Can be educational with the amount of freedom given to students, but may be overwhelming where students cannot learn as well (learning ability)
Understandability	With more queues, it is easier to see how heavy traffic gets, if color coding traffic flow is implemented.	With fewer queues, it is harder to distinguish how bad traffic flow gets to color code the traffic.	May be more confusing for users. They may think they are changing all light timings by changing one.
Usability	Potentially easier to use because we give defaults to the user. If they want to further change light timing, they can.	This is easier for the user to change light timings, as they only have to change one, but is less customizable if students want to see how specific light timings change the traffic.	This option leaves the customizability open to the user, but could be tedious for them to change each light timing individually in the simulation.

UML Class Diagram



UML Class Descriptions

Car

Cars are all set to the same speed and same stopTime. stopTime 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 method updateCar() utilizes the speed, stopTime, direction, and location to update the car’s status at every clock tick. Cars can either move forward on the current road or stop in traffic.

Road

A Road is represented by a single queue, with only one lane and a set integer length. A Road is also classified as a minor or major road. Minor roads typically have shorter green lights than major roads, unless the user specifies the light times to be the same. 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. IsMajor() is a getter function that returns the value of majorRoad. RoadHasSpace() is used to check whether the road has space to add a car. If the Road doesn’t have space, it’s because of severe traffic backup caused by bad light timing.

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. The value of trafficInterval represents the amount of ticks before a Car is added to a Road from the edge of the map. 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.

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.

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), or 2(yellow) stored in currentColor variable. The duration of each light color is represented as integer variables named IntervalRED, IntervalGREEN, 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.

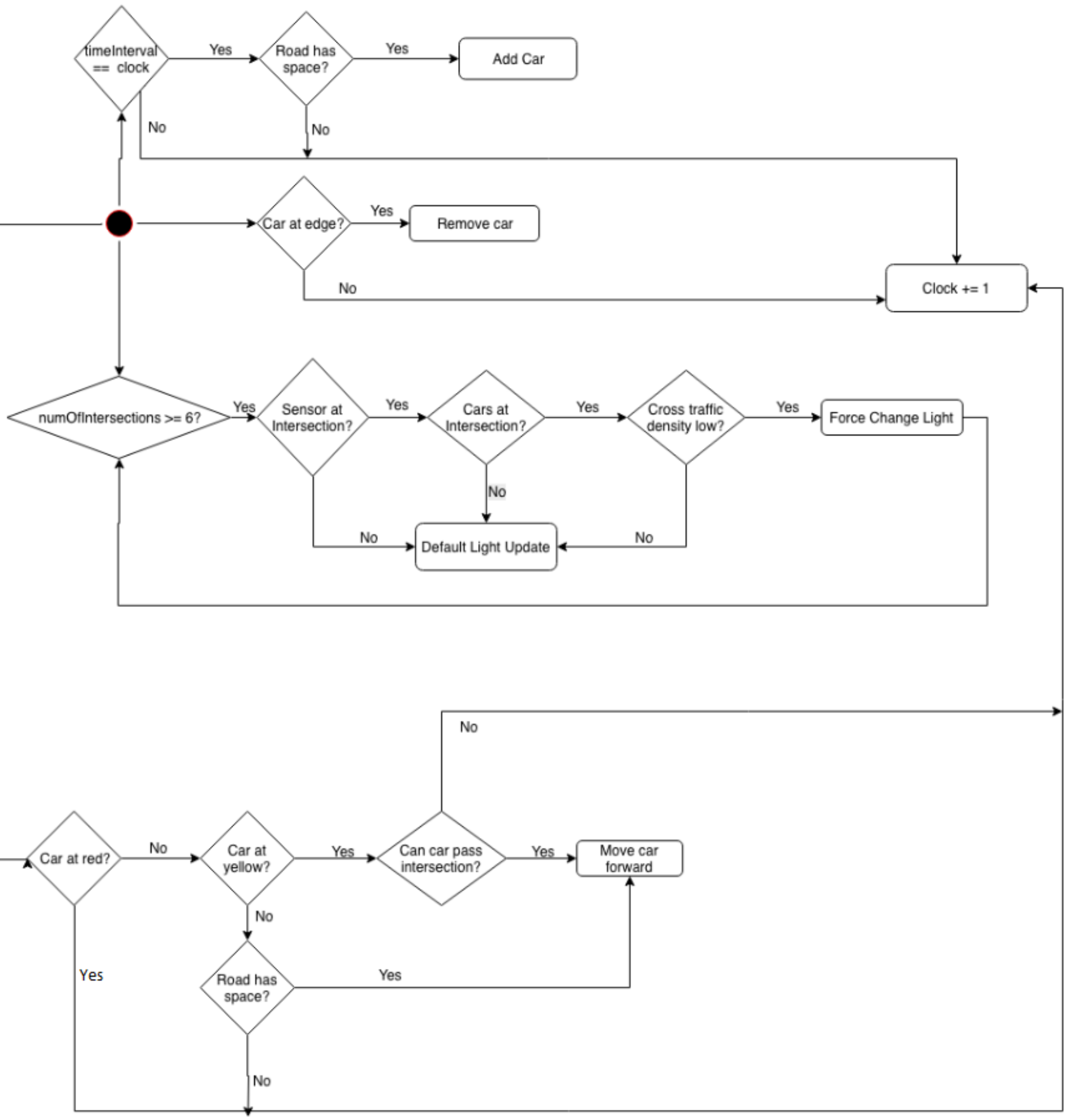
Sensor

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.

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)
    - ◆ major/minor Roads switch to next color
- ➔ Cars enter MapSimulation
    - ◆ Given traffic level (specified by user) and space to add a Car to the Road
- ➔ Cars leave MapSimulation
    - ◆ Use Road length, speed of Car, and traffic level to determine when Car leaves MapSimulation
- ➔ Update existing Cars
    - ◆ Car moves forward if not at red Light, and Road not backed up
    - ◆ Car stops if at red Light or not enough time to cross Intersection on a yellow Light
- ➔ Increment clock one tick

UML Activity Diagram - Advance Clock Tick



Pseudocode

global **<u>Clock</u>**

```
class Car
{
   int speed // how many ticks the car takes to move to next index
   int stopTime //how many ticks the car takes to stop
   char direction
   int location //index of the car in the queue road

   updateCar()
}

class Road
{
   queue<int> road

   int length //( amount of cars possible on this road)
   coordinate[x,y] start
   coordinate[x,y] end

   addCar()
}

class Intersection
{

   Lights[] lights
   //coordinate[x,y] location

   updateIntersection()

}
```

```
class Light
{
   int IntervalRED
   int IntervalGREEN
   int IntervalYELLOW
   int lightChangePoint
   int currentColor

   changeLight( currentColor)
   {
      if(currentColor = 0) // red
         lightChangePoint = IntervalRED
      else if(currentColor = 1) //green
         lightChangePoint = IntervalGREEN
      else //yellow
         lightChangePoint = IntervalYELLOW

       if(clockTime == lightChangePoint)
         {
            if(CurrentColor == red)
               CurrentColor = green
               lightChangePoint = lightChangePoint + IntervalGREEN

            else if(CurrentColor == green)
               CurrentColor = yellow
               lightChangePoint = lightChangePoint + IntervalYELLOW

            else
               CurrentColor = red
               lightChangePoint = lightChangePoint + IntervalRED
          }
        else do nothing

   }
}
```

```
class MapSimulation
{
    private bool SensorExists
    Car[] cars
    Intersection[] intersections
    [][] map

    advanceClock() //Clock += 1
    addCar() //Add car might fit better in the road class
    removeCar()
    addIntersection()
    removeIntersection()
    __canRemove() (at least 7 intersections)
    isSensorOn()          // public
    toggleSensor()

}
```