Tom Dale

CS-273

FinalSpec.doc

7-24-2016

Traffic Junction Simulation

**Project Requirement:**

The people of 273*ville* are planning to construct new intersections within the city limits. At these intersections there will be either a roundabout, or a four-way traffic light constructed. To understand the efficiency of both of these intersection types, and to determine which should be used at a given junction a computer simulation was requested. The user will input the total cars using the intersection per hour as well as the percent of cars entering the four-way from the north, south, east, and west direction. The simulation will run a 4 hour block and give the user information about the time efficiency of both types of intersections and a recommendation for which intersection type to build.

**Requirement Specifications**

* Program needs to simulate traffic stops in the town of 273*ville*, population 2000
* The names of all the residents of 273*ville* are stored in our class folder.
* Every person in 273*ville* could drive through the intersection.
  + User sets average amount of vehicles using intersection per hour.
  + Vehicles can be cars or emergency vehicles.
  + On average 1 out of 400 cars will be an emergency vehicle.
  + Every resident is paired with one vehicle type, managed in a set
* Each intersection has four incoming streets labelled North, South, East, and West
  + User sets a percentage of vehicles that drive from each direction. (All totaling 100%)
* A car is 50% likely to drive straight, 25% likely to turn right, and 25% likely to turn left.
* U-turns are illegal.
* Time to go through intersection is based on intersections with speed limits of 30 mph.
  + Cars should take 2-10 seconds in the actual intersection
  + A car moves one space in intersection per second
  + Left turns should take longer than straight, and straight should be longer than right
* After the user enters the 5 inputs the simulation runs for a traffic light and roundabout.
  + Both intersection types have different rules (mentioned below).
  + When emergency vehicles approach all cars exit the intersection and no new cars enter until the emergency vehicle is through.
* The simulations are run for 4 hours
* Results are displayed for both simulation
  + Stats Displayed: Time spent in line, time spent in intersection, and average total time for a one car.
* Rules for Roundabouts:
  + 8 spaces in roundabouts (see intersection diagram)
  + A 0-2 second wait simulates hesitancy of a roundabout
  + Cars will not enter if portion of roundabout in front of them is taken
* Rules for Traffic Lights:
  + 4 spaces in intersection (see intersection diagram)
  + Lights can be GO for north/south or east/west, there are no special turning lights.
  + Light will change after 3 minutes if it is needed, or it will change if it’s lanes are empty.
  + Yellow lights are not shown, but new car entry is updated every 2 seconds to avoid collision
  + No car will enter the intersection if the space in front of them is taken
  + If space required to move into intersection is open:
    - Right turns are always allowed
    - Driving straight must have a green light
    - Turning left must have an empty que or car turning left at the opposite road
* The program will recommend that the intersection with the lowest wait times be constructed.

**Design**

The main function will have the user input the average amount of cars that go through the intersection, as well as the percent of those cars coming from each of the four directions. Main will also ask the user if he would like to be shown all the data of cars moving through traffic. The amount of cars driving through from each road as well as a Boolean show value to designate to show the exact traffic or not. Exception handlers will catch over 100% errors or inproper values. The main function will call both roundabout and trafficlight simulations to run. After this is finished main will call the .outputData() and the .rate() functions from both intersection simulations and compare them. Rate will be an integer number that tests the overall wait time of cars. The intersection type with the lower .rate() will be suggested to the user as the recommended traffic junction.

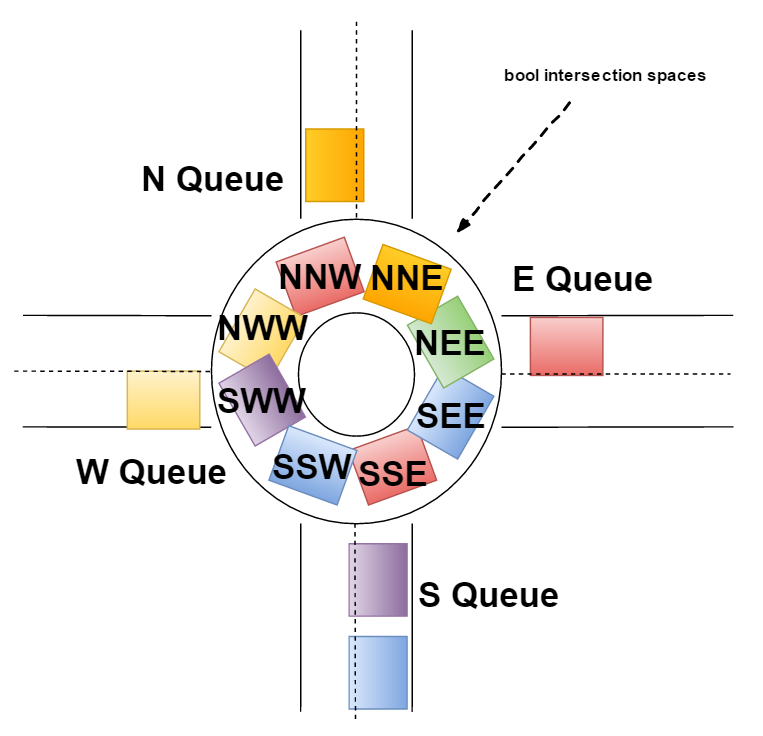
The traffic simulation will be run mainly within the two classes Roundabout and TrafficLight each inheriting from the intersection class. The intersection .run() will call add cars, and then call moveCarsIN, moveCarsOut and moveTraffic. These functions are abstract and will be implemented differently be both child classes because of the different nature of roundabouts and traffic lights. Both intersections have for queues N,S,E,W that hold cars waiting to enter junction. The roundabout has eight bool values NNE, NEE,SEE, SSE,SWW,SSW,NNW, NWW that represent the space required for a new car to enter the roundabout. The lighted intersection works differently, there are only four spaces NE,SE,SW,NW that cars move though, but there is also a NStrafficlight bool that represents a north-south traffic light when true and east-west light when false. The run function is a while loop that increments the clock with each loop representing one second until 14,400 second (four hours) have been run. It calls a function input cars into the queues if a new car arrives. If bool showCars is true then call each car upon leaving the intersection to .print\_data(). Next the new car is checked to be an emergency vehicle, if it is then set emergency count to the vehicle .get\_time\_through(). Now check to see if all spaces in intersection are empty, and if so decrement emergency count once per loop. If the new vehicle was not an emergency vehicle then call a moveCarsIn function to move any car that is able out of the front of the queue and into a space in the intersection.

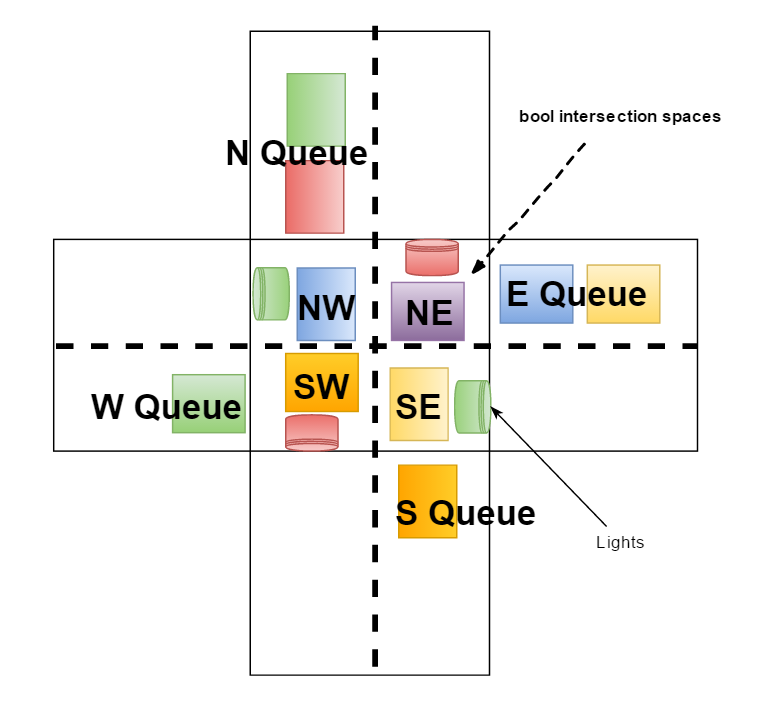
The random file is used to execute any random function. It contains a function to seed the rand() function with current time. It contains a bool rand(int x) function that returns true x/3600 % of the time (the amount of seconds in one hour). This is used by the addCars function to designate when a new car approaches. It also has a string randDirection() function that returns left 25%, right %25, and straight 50% of the time to be used by a car object to decide which way it is turning. A similar function string randType() returns vehicle types at differing percent chances. It also uses an int rand\_int(int max) function to return a random integer between 0 and max for getting random cars from the set.

The set\_of\_cars class is used to initialize a set of human + car matches. Using its getCar() function any of the 2000 residents can be randomly called driving their individual car. Using a set, one person who enters the roundabout more than once will be driving the same car.

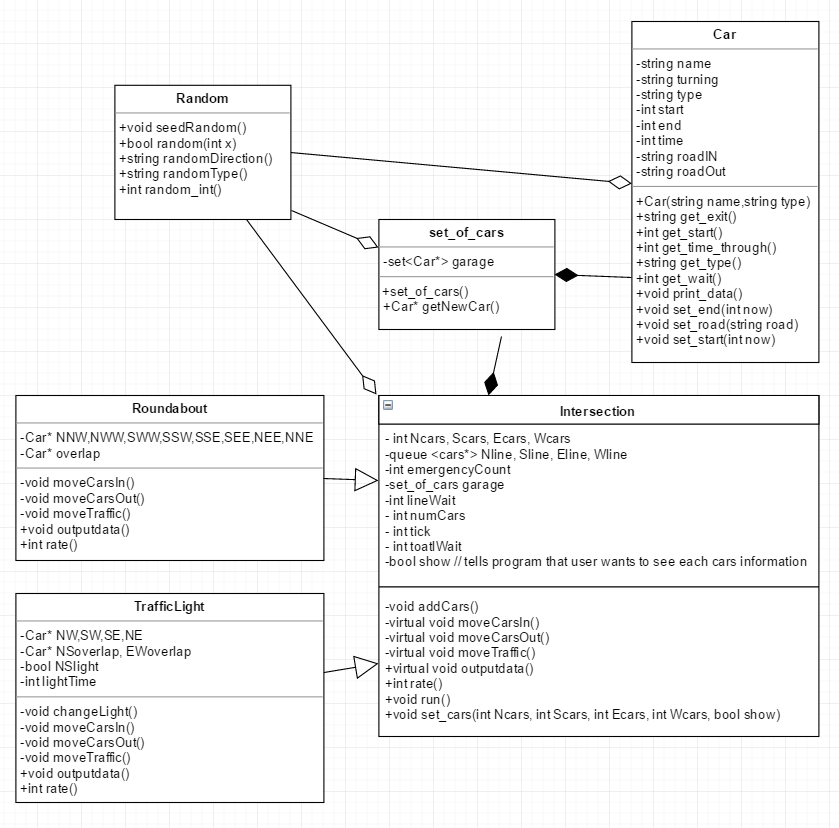
The Car objects are used to keep track of the driver name and car type, the direction from, and direction going to. Begin and end time is set and held by a car to be later added to totalTime in the intersection simulations. Cars have getters and setters for the information that needs to be received or changed by intersection. Upon construction the car will use its entering road, and a randomly generated turn direction (from random class) to calculate roadOut. The print\_data() function will show all the important information about that car upon leaving the intersection, assuming the user wanted to be shown all individual data.

**Intersection Diagrams:**





**UML Diagram:**

****

**Use Cases:**

-Enter variables (no intersection user data)

|  |  |  |
| --- | --- | --- |
| 1 | **Use Input** | **System Response** |
| 2 |  | Print welcome message and prompt for average cars per hour |
| 3 | Enter average cars per hour hour | If entry < 0 ask user to try again |
| 4 |  | Ask user to enter % traffic from North street |
| 5 | Enter number for percent traffic from North | If entry < 0 or > 100 ask user to try again |
| 6 | Same as above for South, East, and West | Repeat last sequence for South, East, and West car traffic. |
| **7** |  | If North + South + East + West are not equal to 100 ask user to try again |
| **8** |  | Convert from percent traffic N,S,E,W to amount of cars coming from N,S,E,W. |
| **9** |  | Send N,S,E,W to roundabout and traffic light tests. |
| **10** |  | Ask user if they would like to view traffic user data. |
| **11** | Enter ‘n’ (or anything else) | Continue as normal |
| **12** |  | Run roundabout and traffic light and return simulation data |
| **13** |  | Print which intersection type is recommended from simulation |

**-**Choose to see intersection users (inserted starting at row 10 from previous use case);

|  |  |  |
| --- | --- | --- |
| 1 |  | Ask user if they would like to view traffic user data |
| 2 | Enter ‘y’ |  |
| 3 |  | Run roundabout and traffic light and return each vehicle name type turn and time taken |
| 4 |  | Return simulation data |
| 5 |  | Print which intersection type is recommended from simulation |

**Psuedo-code:**

Get\_input()

(ALL cin calls should be calls to a read\_int function that catches non int exceptions)

1. While (true)
2. Cout << enter cars per hour
3. Cin >> avgcars
4. If input restart loop < 0 restart loop
5. Cout << Enter north south east west percent traffic
6. Cin Ncars Scars Ecars Wcars
7. If any inputs are < 0 or > 100 or if all four value are equal to 100 then
8. Cout << do you want to see individual vehicle information
9. Cin >> ‘y’
10. Set show to true
11. Pass roundabout and lightedintersection .set\_cars(Ncars,Scars,Ecars,Wcars,show)
12. Return

Run()

1. Int tick = 0
2. While (tick < 14400)
3. addCars()
4. moveCarsOut()
5. moveTraffic()
6. if emergency count is equal to 0
7. move cars in
8. else if all spaces in intersection are empty
9. emergency – 1
10. return

addCars()

1. if(random(Ncars)) //car approach rate
2. car\* entering = new car
3. if( new car is emergency vehicle)
4. emergency = entering->get\_time\_through()
5. if (show) entering-> print data
6. else
7. Nqueue.push\_back(entering)
8. //do this for 4 times for all directions

moveCarsOut() in roundabout

1. if space before and exit is full, and that car wants to exit there
2. car at that space -> set\_end()
3. if (show) car at that space -> print data
4. that space = null
5. //do this for 4 times for each exit

moveCarsOut() in lighted intersection

(same as in roundabout but different code because different space locations)

moveCarsIn() in roundabout

1. If space ahead of Nline is free
2. Move car in
3. Do this for all 4 directions

MoveCarsIn() in lighted intersection

1. If space ahead is free
2. If Nline.front-> get\_turn()= = right
3. Move car in
4. Else if north/south light = go
5. if Nline.front-> get\_turn() == straight
6. Move car in
7. Else If Sline.empty() or Sline.front->get\_turn() == left //opposite road empty or turning left
8. Move car in
9. //do this for all four queue
10. //this gets confusing even in pseudo code but nested if statements just follow basic 4-way lighted intersection rules

MoveTraffic() in roundabout

1. Move NNE space to overlap
2. Move every space to the next space in a counterclockwise order (prevents collision)
3. Move overlap to NNW

MoveTraffic() in roundabout

//if car is turning right then moveCarsOut would’ve already move it out

1. If(car moving straight) move to space ahead
2. Else(car must be moving left)
3. Put car in overlap space
4. Do this again for opposite road but replace overlap space with this space
5. Move overlap space to space diagonal from original space
6. //2 opposing left turns cause this confusion because they need to swap places
7. //do it all again for the two roads perpendicular