Report

In this report, I will be discussing the design of my program created to model a simulation of temporary traffic lights. The design decisions and assumptions will be discussed, as well as an exploration of the results of running the simulation. Output of the code will also be included.

Design Decisions and Assumptions

Structures

To begin with, two data structures are used. I have used a queue to represent each side of the road, and a stats structure to store information based on each queue and simulation.

The queue is constructed as a linked list and is based around the assumption that no cars will leave the queue once they have entered, and that no cars will be able to over-take each other. Additionally, it is assumed that a car will immediately move to take the place of the car in front when one car is removed. With these assumptions in mind queue has been created where each car carries its own wait time. Additionally, it has been implemented using a linked list, as the length of the queue is expected to change both dynamically and randomly.

The stats construct is used to store information relating to both one simulation, and the cumulative value of all simulations. This information consists of the number of cars passed within a simulation, the cumulative wait time of all passing cars within given simulation, the maximum wait time of a given car, and the time to clear after the first 500 iterations of cars arriving has passed. These statistics allow the collection of data stated within the specification. Additionally, two sets are used outside of the simulations to collate information from all simulations. In this case, the cumulative wait refers instead to the cumulative average wait of all simulations.

The main function

Looking at the structure of the main function, there are 3 distinct parts.

Firstly, inputs are requested, and memory is allocated to store statistics of the simulations. When inputs are requested, two similar functions are called: *getInputChance* and *getInputPeriod*. These two functions ask for user input, repeating until a valid input is received, or a non-integer is entered, causing the program to exit. The difference between these two functions relates to the limits of each. As the input is meant to be a chance, I've chosen that the chance should be represented as an integer between 0-100. This can be represented as a percentage and allows testing of a wide range of chances. The period function instead asks for a range between 1 and 500. The limit of 500 is due to the fact there will be no useful difference with periods greater than that, as one side of the road would have a constant green light, meaning the only effect would be on how long the other side must wait until the end of that period. Additionally, a period of 0 would result in an instantaneous switch, which would mean that no car passes through a specific side. This would cause the simulation to not terminate in any case where a car would arrive and would essentially be the simulation of a one-way road.

After inputs have been received and memory for information allocated, the *runOneSimulation* function is run 100 times, each time using a different random seed based on the iteration to ensure different random behaviour between runs.

Finally, information is displayed based on the findings throughout the simulations. Memory allocated to the statistics is freed, and the program ends.

Queue functions

The queue functions consist of *enqueue, dequeue, addToWaits* and *countCars*. Excluding dequeue, all the functions use recursion to manipulate the queue, where they are called with a queue node,

an action is taken depending on whether the node has any other node following it, and then the function is run again using that following node (assuming there is one). *addToWaits* adds 1 to the wait time of a given car, and *countCars* adds one to an integer as it progresses through the queue. *Enqueue* calls itself with the next node, creating a node at the end of the queue once it is reached. *Dequeue* first checks if a queue is empty, and if not, removes the first node whilst returning the value stored within. The pointer for the queue is then changed to point to the node following the first.

Stats functions

Several functions are also used to manipulate the stats structure. *createStats* and *freeStats* are used to allocate and free memory to stats respectively, whilst *addToStats* is used to add information from a car into a set of stats in a simulation. *addToTotals* is used to add the statistics from a simulation into the total statistics for all simulations.

Checking functions and pseudo-Booleans

In place of the Boolean data type, a character is used as a substitute, acting as *0* for *false*, and *1* for *true*. This is used within the input functions whilst checking for valid inputs, and it is also linked to how the two functions, *randomCheck* and *lightCheck* function.
RandomCheck takes an integer as an argument, and then retrieves the value of a random number modulo 100. If the number is less than or equal to the random number, then *false* is returned as the character 0. Otherwise, *true* is returned as a 1. This represents chance as an integer percentage).
lightCheck takes a set of arguments and returns either a 0 or 1 depending on whether the light should change based on those arguments.

Simulation

Each simulation takes 6 arguments. *leftChance* and *rightChance* represent the chances for a car to arrive on the left and right queue per time interval respectively. *leftLimit* and *rightLimit* are used to represent the time period each traffic light will be green before switching to red. Finally, *leftTotals* and *rightTotals* are added to at the end of a simulation to store the results of the simulation.

Initially, two queues and stats are created, along with a value for measuring the current iteration, and a value to record the current length of time a given light has been green. Within the specification, it is said that only one light can be green at a time, whilst the other is red, and the effect of amber lights is not simulated. As this is the case, only two states exist, one where the left light is green, and one where the left light is not green. Because of this, I have chosen to represent the state of the lights as a character acting as a Boolean, called *leftLightGreen*. When *leftLightGreen* is 0, it means that the left light is red, which implies that the right light is green. Arbitrarily, the left light starts red, and the right light starts green.

The majority of the simulation takes place within a while loop. The condition for the loop to run is that the number of iterations is under 500, or that both queues are not empty. Because of this, the required 500 iterations are guaranteed to occur, and the while loop stops after both queues have been cleared.
Within the loop, an if/else statement is used. The condition being the result of the previously mentioned *lightCheck*. The arguments used are the duration of the current light being green, which light is on, and the two inputted periods for each light. If the lights are going to switch, the lights are switched, then the iteration counter is increased, and waits added to cars within the queues. Otherwise, 2 random checks are made for whether a car should be enqueued, and then a car is dequeued from the queue with the respective green light. The information from the dequeued car is added to the relevant statistics, the iteration counter is increased, the period of the currently green

light is increased, and the clearance times are added to if no more cares are arriving.
When the while loop has finished, the statistics from the simulation are added to the totals, and the memory allocated to the statistics of the simulation are freed.

Experiment

A series of experiments will be conducted using the program, and the results will be discussed here. To begin with, we'll look at the effects of changing the traffic light period.

Traffic light period

The first series of tests will be using 100% chance on either side of the road for a car to arrive, and will focus on the periods 3, 5, 7. The periods will be equal on both sides.

Period = 3	Period = 5	Period = 7
Results (averaged over 100 runs): from left: number of vehicles: 375 average waiting time: 252 maximum waiting time: 500 clearance time: 498 from right: number of vehicles: 375 average waiting time: 248 maximum waiting time: 496 clearance time: 494	Results (averaged over 100 runs): from left: number of vehicles: 417 average waiting time: 253 maximum waiting time: 504 clearance time: 503 from right: number of vehicles: 417 average waiting time: 247 maximum waiting time: 498 clearance time: 497	Results (averaged over 100 runs): from left: number of vehicles: 438 average waiting time: 254 maximum waiting time: 504 clearance time: 503 from right: number of vehicles: 438 average waiting time: 246 maximum waiting time: 496 clearance time: 495

As we can see, the main value that changes when period increases are the number of vehicles that pass through each side. This is due to the fact that there are less time iterations spent on the changing of the traffic lights, allowing more opportunities for vehicles to arrive. We also see slightly lower wait values on the right side, likely as a result of the green light starting on that side.

Vehicle arrival chance

The second series of tests will be focus on the effect chance has on either side of the road. As larger numbers of vehicles can pass as a result of higher periods, the light time period will be fixed to 7 for both sides of the road during these tests. The chances 30%, 50% and 70% will be tested.

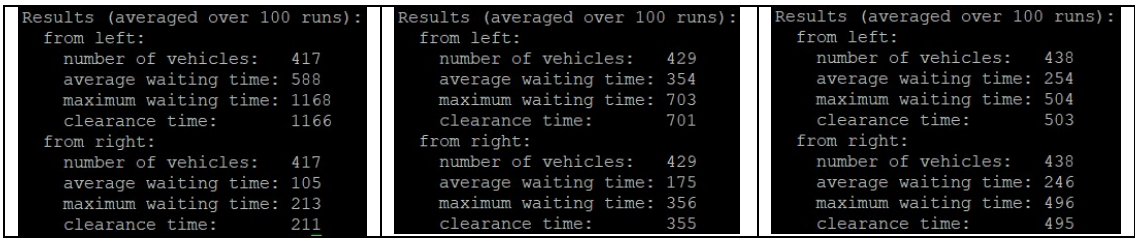
30%	50%	70%
Results (averaged over 100 runs): from left: number of vehicles: 133 average waiting time: 3 maximum waiting time: 10 clearance time: 3 from right: number of vehicles: 132 average waiting time: 2 maximum waiting time: 10 clearance time: 0	Results (averaged over 100 runs): from left: number of vehicles: 221 average waiting time: 15 maximum waiting time: 35 clearance time: 21 from right: number of vehicles: 218 average waiting time: 12 maximum waiting time: 30 clearance time: 16	Results (averaged over 100 runs): from left: number of vehicles: 308 average waiting time: 104 maximum waiting time: 209 clearance time: 206 from right: number of vehicles: 306 average waiting time: 100 maximum waiting time: 202 clearance time: 198

The number of vehicles increases with the chance of a vehicle arriving, quite obviously. Additionally, we can see that the previous assumption of the right side having faster clear times and lower averages due to starting green remains true.

Different periods

As the chance for a car arriving only affects the number of cars when they are equal, this series of tests will focus on differing values for the left side. The right side will be fixed at a period of 7, and both sides will have 100% chance of a car arriving, but now the left hand side will change between periods of 3 and 5 (previous test of 7 will be included for comparison).

Left: 3	Left: 5	Left: 7



From this information, we can see that when one side has a shorter period than the other, the wait times and time to clear can increase drastically. Because of this, I believe that the period of a given side should be proportional to the chance of a car arriving for the shortest total wait times.

Proportional periods

This test will now focus on how the wait times interact when the car arrival chance differs with different periods. The right side will have a fixed 100% chance of a car arriving, with a fixed period of 7. The left hand side will have several different combinations of wait time and chance to examine the relationship between the two variables.

Chance: 70% Period: 3	Chance: 50% Period: 7	Chance: 30% Period: 7
Results (averaged over 100 runs): from left: number of vehicles: 293 average waiting time: 339 maximum waiting time: 679 clearance time: 676 from right: number of vehicles: 417 average waiting time: 105 maximum waiting time: 213 clearance time: 211	Results (averaged over 100 runs): from left: number of vehicles: 221 average waiting time: 15 maximum waiting time: 35 clearance time: 21 from right: number of vehicles: 438 average waiting time: 246 maximum waiting time: 496 clearance time: 495	Results (averaged over 100 runs): from left: number of vehicles: 133 average waiting time: 3 maximum waiting time: 10 clearance time: 3 from right: number of vehicles: 438 average waiting time: 246 maximum waiting time: 496 clearance time: 495
Chance: 70% Period: 7	**Chance: 50% Period: 5**	**Chance: 30% Period: 3**
Results (averaged over 100 runs): from left: number of vehicles: 308 average waiting time: 104 maximum waiting time: 209 clearance time: 206 from right: number of vehicles: 438 average waiting time: 246 maximum waiting time: 496 clearance time: 495	Results (averaged over 100 runs): from left: number of vehicles: 216 average waiting time: 57 maximum waiting time: 116 clearance time: 110 from right: number of vehicles: 429 average waiting time: 175 maximum waiting time: 356 clearance time: 355	Results (averaged over 100 runs): from left: number of vehicles: 127 average waiting time: 23 maximum waiting time: 51 clearance time: 31 from right: number of vehicles: 417 average waiting time: 105 maximum waiting time: 213 clearance time: 211

From these results, we can see that the clearance time of a given queue increases as a smaller proportion of time is allotted to it. We can see that the total clearance time of both queues is lower in situations where a proportional period is allotted based on the chance of cars arriving. This means that, for the least amount of time spent for both queues to clear, periods should be used based on the amount of traffic coming in a certain direction, with greater periods for greater amounts of traffic.

Code example

All code within the experiment was outputted by the code. There is also input handling, and inclusion of inputted parameters.