CSCI 222 Lab #4

Traffic Simulator

The goal of a traffic simulation is to collect data about the traffic flow. In our project we start with modeling one orthogonal intersection controlled by a traffic light. Each roadway leading to and from the intersection will have a fixed length (providing space for a fixed number of cars). The simulation is driven by a timer. Vehicles (e.g. cars, trucks, buses, motorcycles) arrive onto the four lanes of traffic leading to the intersection every timer unit with a probability 1:n (n selected by user - suggested value 6). All cars travel at a constant speed of one half car length per timer unit, unless the road ahead is blocked (by another car or by the color of the traffic light). Cars exit the simulation when the front of the car 'falls off the road'. The duration of the light cycles for the traffic light is selected by the user. Due to the geometry of our design a car needs six timer units to clear the intersection. This determines the duration of yellow light.

At this point we do not define the statistics we will collect about the individual cars or about the overall simulation performance, other than counting the number of cars that traversed the entire road in each direction. During the development process the user input is limited to selecting the traffic light timing, the probability of a vehicle arriving (e.g., higher for cars, medium for motorcycles, and lower for trucks and buses), during any given timer unit, and the length of the simulation run. Later, a more complex user interface will allow the user to select the statistics that should be collected and displayed.

The objects that will be involved in the simulation include:

Objects:

- road (two lanes of traffic - a static structure)
- car, truck, bus, motorcycle (inherit from class vehicle, moving in a given direction, currently in a given position)
- collection of vehicles (every object in the collection needs to be updated during each timer loop)
- traffic light

Actions:

- a mechanism is needed to record whether a given place on the road is free to enter or is blocked
- a vehicle needs to be able to check whether it can move forward
- a vehicle needs to be able to move
- a vehicle needs to detect when it arrives at the end of the road
- the traffic light needs to set its timing
- the traffic light needs to be able to update its status
- the traffic light needs to signal that a particular place on the road is blocked due to its current color

- vehicle collection needs to update all existing vehicles
- vehicle collection needs to remove exiting vehicles
- vehicle collection needs to add newly arrived vehicles
- for vehicle collection at intersection, use of the queue STL is required

Our simulation is controlled by a discrete timer. The granularity of the timer units determines the granularity of the road surface and the atomic car movement increments. Each car occupies two square units of the road surface and can move one unit forward during one timer cycle. These square units (places) form the basis of our simulation.

Place Class

This class is at the heart of the entire program. Each place represents one patch of asphalt on the road. It contains four pointers to other places adjacent to this one. The pointers contain valid links only if travel in that direction is permitted. So, all places in the northbound lane (except for the last one) will have links to their neighbors north of them, but no other links. When we start building intersections, the places where two directions of travel cross will have links to both neighbors. A place can be blocked if it is occupied by a car or if the light ahead is not green.

To display a place (e.g., after a vehicle moved away) we also need to record its physical location in the display. Because the whole display can be treated as a grid specifying the row and column information is sufficient.

Vehicle Class

The vehicle class contains the location, speed, weight information of each of its derived vehicle type (car, truck, bus motorcycle).

Car Class

Each car is represented by an object in the Car class derived from Vehicle. To distinguish the cars from each other, each is given a color at random (from a list of about 12 colors). The car dimensions are such that each car fits inside two road places. The car object will have two links to the two places it occupies on the road. The car object will also record the direction in which it is facing. This information provides the information needed to display a car or to repaint the background as the car moves. Additional member data may be added - either to indicate the intent to turn or to collect statistics during the travel. Member functions will allow us to check if a car can move, to move the car, to check if it is at the end of the road, and, of course, to erase and display the car.

Truck Class

Each truck is represented by an object in the Truck class derived from Vehicle. To distinguish the trucks from each other, each is given a color at random (from a list of about 12 colors). Each truck contains an additional weight data field for the trailer carrying cargo (in addition to the weight of the engine stored in the Vehicle class) and a destination location data field. The truck dimensions are such that each truck fits inside five road places. The truck object will have five links to the five places it occupies on the road. The truck object will also record the direction in which it is

facing. This information provides the information needed to display a truck or to repaint the background as the truck moves. Additional member data may be added - either to indicate the intent to turn or to collect statistics during the travel. Member functions will allow us to check if a truck can move, to move the truck, to check if it is at the end of the road, and, of course, to erase and display the truck.

Bus Class

Each bus is represented by an object in the Bus class derived from Vehicle. To distinguish the buses from each other, each is given a name of the school or MBTA. Each bus contains an additional number of passengers data field on the bus. The bus dimensions are such that each bus fits inside four road places. The bus object will have four links to the four places it occupies on the road. The bus object will also record the direction in which it is facing. This information provides the information needed to display a bus or to repaint the background as the bus moves. Additional member data may be added - either to indicate the intent to turn or to collect statistics during the travel. Member functions will allow us to check if a bus can move, to move the bus, to check if it is at the end of the road, and, of course, to erase and display the bus.

Motorcycle Class

Each motorcycle is represented by an object in the motorcycle class derived from Vehicle. To distinguish the motorcycles from each other, each is given a brand name (Harley, Kawasaki, etc.). The motorcycle dimensions are such that each motorcycle fits inside one road place. The motorcycle object will have one link to the place it occupies on the road. The motorcycle object will also record the direction in which it is facing. This information provides the information needed to display a motorcycle or to repaint the background as the motorcycle moves. Additional member data may be added - either to indicate the intent to turn or to collect statistics during the travel. Member functions will allow us to check if a motorcycle can move, to move the motorcycle, to check if it is at the end of the road, and, of course, to erase and display the motorcycle.

Road Class

This is an interesting class. Each lane of traffic is represented by one object in this class. Each Road object is a linked list of Place objects (you should use the linked list STL). The Road object knows its direction, can return a pointer to a specified place on the road, especially to the first place where a car will enter the road.

Traffic Light Class

The TrafficLight class is responsible for timing the light changes and signaling the current color. The signaling of the color is implemented by blocking and clearing the four places in the roads that are just before the entrance to the intersection. This behavior emulates gates at toll booths - a rather drastic method for enforcing the traffic rules. However, it greatly simplifies the object interaction.

To initialize the timing of the traffic light, we need to know the duration of the three light cycles. For the light to function properly, the duration of the red light should be the sum of the green and yellow light in the opposite direction. Knowing that the yellow light has a fixed duration of six

cycles the user only needs to specify the duration of green light in two directions (north/south vs. east/west).

The constructor for this class should ask the user to enter the timer settings. The Update function should advance the traffic light timer, display the new state, and block or clear appropriate road places.

VehicleQueue Class

There is one VehicleQueue object for each direction of travel. It is a simple queue of car objects (use the ArrayList class to implement the queue). New vehicles enter at the tail and cars that reach the end are removed from the head. In general, this should be an unstructured collection and we should even be able to update all cars in parallel. However, a queue works well for traffic without turns.

Traffic Simulation

The decision that each Place object has four links makes the car movement in orthogonal directions easy to implement. It is also possible to make 90° turns in either direction. However, a gradual turn in the road or traffic with a passing lane cannot be easily modeled using this class.

The Place object contains a function **FreeToMove(direction)** that indicates whether there is a link to another Place object in that direction and whether this object is currently marked as **free**. It also has a member function **Next(direction)** that returns a pointer to the next Place object. It is used by the car's **Move** function.

The Car/Truck/Bus/Motorcycle class allow us to make every car into an independent object. Each vehicle should be able to move, erase, and display itself. To do this it needs access to the two Place objects that represent its position. These cannot be member data of the vehicle object - we need to use only reference to the Place objects that are part of the Road. As the vehicle moves, it unblocks the Place in the rear and blocks the Place ahead. Vehicle can also collect statistics about itself - the total time on the road, the waiting time, the distance traveled, etc. We see that a vehicle designed this way has no knowledge of the world around it other than its position on the road, the direction of travel, and whether the place ahead is free.

The Road class is a classic example of a static linked structure that is traversed or visited by other objects, but its structure does not change throughout the duration of the simulation.

The Road class and the Place class need to work together here. When building the Road object we need to modify the link member data of Place objects as new ones are added to the road. Once the road is built, these links never change - they are only traversed. The proper design here forces us to do the following:

- there should be no member function in the Place class that changes these links
- the constructor in the Place class should set these links to null
- the Place class should be a friend to Road class, so that the road class has direct access to the link member data

- Road class generates the Place objects and initializes links between them to represent roadways.
- Places that are at the intersection of two roadways are created by the first Road object and the intersecting Road object has to reference the already existing Place object.

This design assures that once the road is built, the links will remain intact throughout the simulation - an important feature. Such static data structures are important in some system programs and when writing a shared code.

An additional problem arises when two roads cross. Now one Place object that is at the point of intersection will belong to two Road objects. Because in C++ we need to do our own garbage collection, writing a destructor for this linked list requires the use of a reference count. We can only delete Place objects that are not parts of any other Road object.

The TrafficLight object has several interesting features. First, we need to represent the current status of the light and define the transitions between the possible states. Our traffic light design requires that the signal in the two opposite directions (north/south or east/west) are the same at all times. There are at least two methods for recording the current state and computing the new state on each timer update. The first method requires that we store the timing data supplied by the user and every time the light color changes we start a countdown with the appropriate value. The design works, but the implementation is quite messy. Our implementation recognizes the fact that the light cycle repeats after a finite (not very large) number of steps. We represent each possible state of the signals by a pair of letters. GR indicates that the traffic light in the north/south direction if green and that is it red in the east/west direction. The constructor for the TrafficLight object computes the sequence of states during the entire cycle and stores them in an array. It also records the length of the cycle. The current state of the traffic light is then encapsulated in one array index. The update function needs only to increment this array index modulo the cycle length. The key design issue here is computing the state each time vs. storing the state transition diagram data directly. For the purposes of readability, maintenance, and even time efficiency, the second design is clearly a better choice. The additional space requirement for recording the state array is not a problem in this case. Either one of these methods can be easily extended to accommodate light cycles in which the two opposite directions are not synchronized.

The second problem that needs to be addressed is the interaction between the light signal and the cars traveling on the road. To simplify things, the traffic light controls the four Place objects at which the cars enter the intersection. It can block them when the light turns yellow and free then when the light turns green. We may think there is a gate there, like at a toll booth. While this makes it impossible to ignore the traffic rules, it simplifies the object interactions. The discussions of the alternative design options for this interaction will undoubtedly expose a number of interesting problems. A Place object can have a marker indicating we are at a light, maybe even a pointer to the TrafficLight object that the car can then query about its state. Things get more complicated and the design must respect the autonomous nature of each object.

Finally, we examine the alternatives for representing the collection of all vehicles on the road. The simplest design uses one queue for each traffic lane. Vehicles enter the queue, traverse the lane,

reach the end and exit. The Road object supplies the pointer to the first Place object on the road. When the vehicle reaches the end, the vehicle's first Place pointer becomes null. VehicleQueue update function performs three tasks:

- moves every vehicle that can move
- removes a vehicle from the front of the queue if it has reached the end of the road
- adds a new vehicle to the queue if "the random number generator says so" and there is a space for this vehicle on the road.

There are no public functions to either insert into the queue or to remove from it - both of these are called internally by the update function.

Also, make sure the creation and destruction of vehicle objects are done appropriately.

When vehicles turn, the vehicle collection needs to be implemented differently. A pointer based linked list is one option. Another one is to allocate memory (array based) to the maximum number of vehicle objects we may have and mark those objects currently on the road as active. On each update, only the active vehicles try to move. New vehicles are given one of the inactive vehicle object locations. When a vehicle object reaches the end of the road, it marks itself inactive.