Program a Concurrent Traffic Simulation

Meets Specifications

Dear student,

you have done an exceptional job for the Concurrent Traffic Simulation project, 🎉 **Congratulations on passing the project** 🎉 , you learned a lot of concepts about concurrency through this course, hope you enjoyed your journey, I have provided some suggestions on project files that you can have a look at. I hope you will find them useful! Please refer to the Code Review section

Things I liked the most

- The structure of the classes (TrafficLight, MessageQueue)
- Successful use of template
- Use random_device for random time generator
- Better to use uniform_**real**_distribution
- Define vars outside any loop for good practice
- Add a _queue.clear( ) as it will make the performance better see this post
- Use random time in double or float instead of int

Suggestions

- Make engine and distribution instances static or global
- Use emplace_back directly here as inside its implementation it moves the obj
- Use time in milliseconds instead of seconds

in the meantime, when you get free enough then please visit the site http://www.cplusplus.com/. Try exploring sites like this and you will learn a lot in the process.

Look for concurrency article 1
Look for concurrency article 2

Now you should post this project on GitHub with a very nice readme

Make Read Me
Here is a blog on how to write a readme

FP.1 Create a TrafficLight class

A `TrafficLight` class is is defined which is a child class of `TrafficObject`.

Great to inherit from TrafficObject publicly 👍
You can read more about modes of inheritance here

The class shall have the public methods `void waitForGreen()` and `void simulate()` as well as `TrafficLightPhase getCurrentPhase()`, where `TrafficLightPhase` is an enum that can be either `red` or `green`.

Also, there should be a private method `void cycleThroughPhases()` and a private member `_currentPhase` which can take `red` or `green` as its value.

All methods definition is in the right place 👏
with the enum defined in the top of the TrafficLight.h file 👏

Resources

- Enumerated types or enums in C++
- Access modifiers in C++

FP.2: Implement a cycleThroughPhases method

Implement the function with an infinite loop that measures the time between two loop cycles and toggles the current phase of the traffic light between red and green.

The cycle duration should be a random value between 4 and 6 seconds, and the while-loop should use `std::this_thread::sleep_for` to wait 1ms between two cycles.

- Better to use time in milliseconds instead of seconds for a wide range of values to choose from.
- Good to use the random device for random time generator with std::mt19937 to make it more random.

The private `cycleThroughPhases()` method should be started in a thread when the public method `simulate` is called. To do this, a thread queue should be used in the base class.

cycleThroughPhases() method started in a thread correctly using the simulate function, all parameters to create the thread are passed correctly
Great job 👏

FP.3 Define class MessageQueue

A `MessageQueue` class is defined in the header of class `TrafficLight` which has the public methods `send` and `receive`.

A **MessageQueue** class is defined in the header of class TrafficLight with two methods ***send*** and ***receive*** 👏
Read more about templates here

`send` should take an `rvalue` reference of type `TrafficLightPhase` whereas `receive` should return this type.

Also, the `MessageQueue` class should define a `std::dequeue` called `_queue`, which stores objects of type `TrafficLightPhase`.

Also, there should be a `std::condition_variable` as well as an `std::mutex` as private members.

- send and receive are defined public members 👏
- std : : deque, std : :condition_variable and std::mutex are defined private members 👏
- The class is defined as Generic 👏
  Read more about conditional variable here

FP.4 Implement the method `send`

The method `send` should use the mechanisms `std::lock_guard<std::mutex>` as well as `_condition.notify_one()` to add a new message to the queue and afterwards send a notification.

In the class `TrafficLight`, a private member of type `MessageQueue` should be created and used within the infinite loop to push each new `TrafficLightPhase` into it by calling `send` in conjunction with move semantics.

- Send is perfectly implemented in Cpp file👏
- Use **std::lock_guard<std::mutex>** mechanism to protect the send, place the message in the **queue**, and then notify through the **condition var**.

- Send is used within **cycleThroughPhases** through an infinite while loop
  Here is a good reference on rules of move and copy constructor

FP.5 Implement the methods `receive` and `waitForGreen`

The method receive should use `std::unique_lock<std::mutex>` and `_condition.wait()` to wait for and receive new messages and pull them from the queue using move semantics. The received object should then be returned by the `receive` function.

- **receive** correctly use the std::unique_lock<std::mutex> as a protection mechanism
- **_condition.wait()** is used to wait for and receive new messages
- **receive** process for the message happen through **move semantics** then **returned**

The method `waitForGreen` is completed, in which an infinite while loop runs and repeatedly calls the `receive` function on the message queue. Once it receives `TrafficLightPhase::green`, the method returns.

The **infinite loop** is created correctly and **breaks only** when receive got a **green** sign

FP.6 Implement message exchange

In class Intersection, a private member `_trafficLight` of type `TrafficLight` should exist.

The method `Intersection::simulate()`, should start the simulation of `_trafficLight`.

The method `Intersection::addVehicleToQueue`, should use the methods `TrafficLight::getCurrentPhase` and `TrafficLight::waitForGreen` to block
the execution until the traffic light turns green.

- **_trafficLight** is defined in class **Intersection** as a **private** member
- Intersection::simulate() is used to start a simulation
- in **Intersection : : addVehicleToQueue** used **TrafficLight::getCurrentPhase** to block the traffic when it is **red**

DOWNLOAD PROJECT

RETURN TO PATH