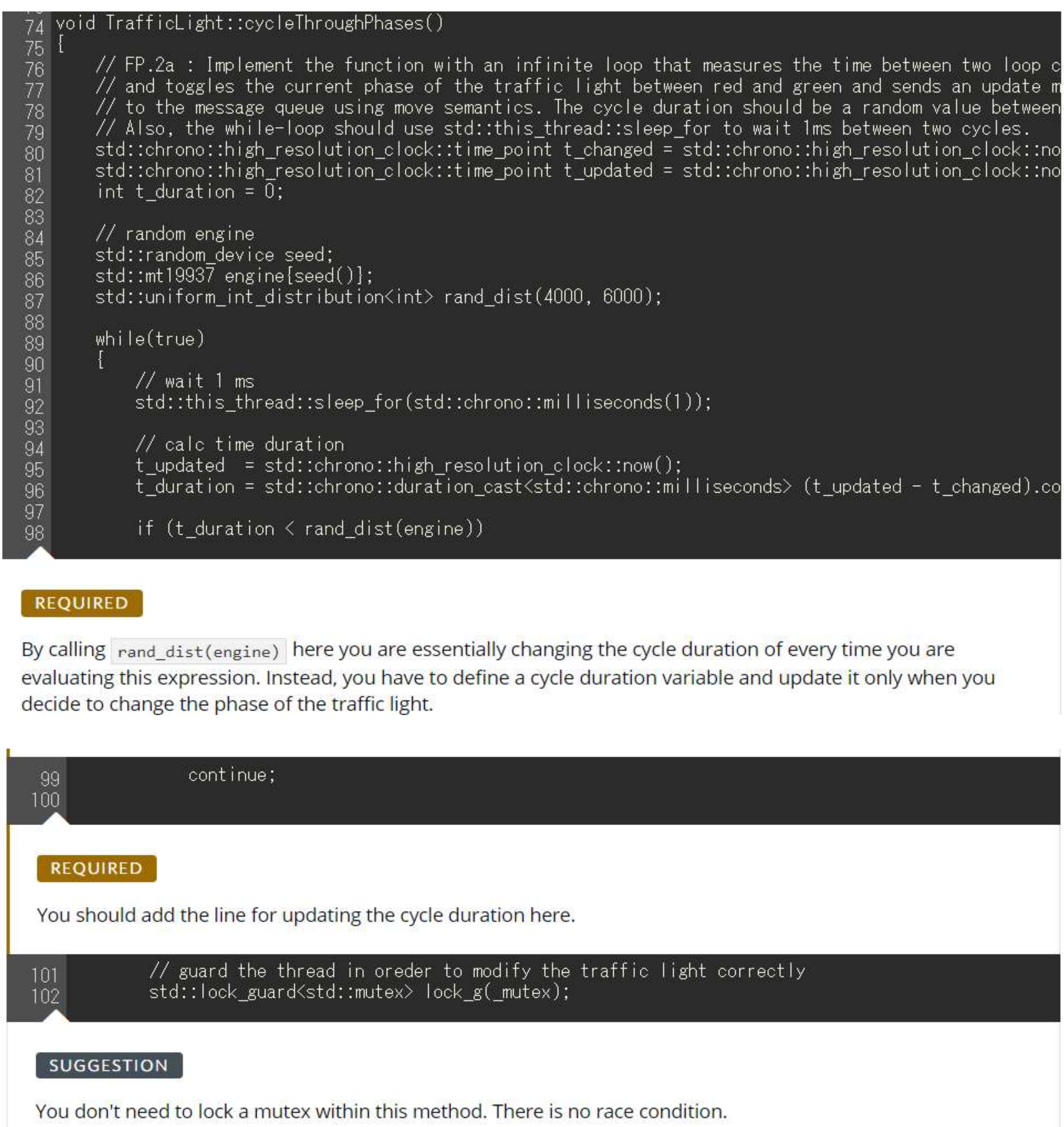
Code Review 1st

● TrafficLight.cpp



```
void TrafficLight::cycleThroughPhases()
{
    // FP.2a : Implement the function with an infinite loop that measures the time between two loop c
    // and toggles the current phase of the traffic light between red and green and sends an update m
    // to the message queue using move semantics. The cycle duration should be a random value between
    // Also, the while-loop should use std::this_thread::sleep_for to wait 1ms between two cycles.
    std::chrono::high_resolution_clock::time_point t_changed = std::chrono::high_resolution_clock::no
    std::chrono::high_resolution_clock::time_point t_updated = std::chrono::high_resolution_clock::no
    int t_duration = 0;

    // random engine
    std::random_device seed;
    std::mt19937 engine{seed()};
    std::uniform_int_distribution<int> rand_dist(4000, 6000);

    while(true)
    {
        // wait 1 ms
        std::this_thread::sleep_for(std::chrono::milliseconds(1));

        // calc time duration
        t_updated  = std::chrono::high_resolution_clock::now();
        t_duration = std::chrono::duration_cast<std::chrono::milliseconds> (t_updated - t_changed).co

        if (t_duration < rand_dist(engine))
```

REQUIRED

By calling `rand_dist(engine)` here you are essentially changing the cycle duration of every time you are evaluating this expression. Instead, you have to define a cycle duration variable and update it only when you decide to change the phase of the traffic light.

```
            continue;

```

REQUIRED

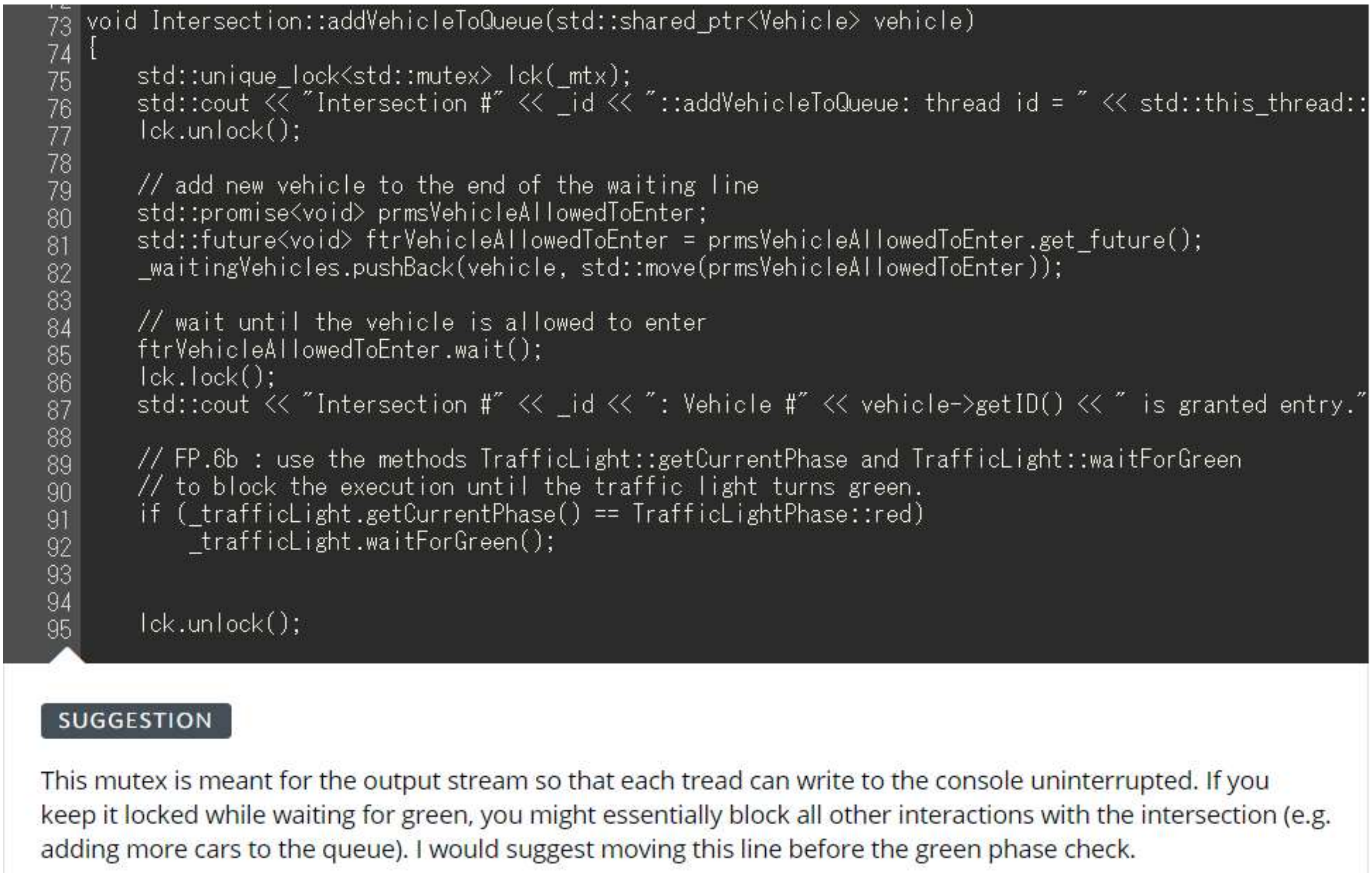
You should add the line for updating the cycle duration here.

```
        // guard the thread in oreder to modify the traffic light correctly
        std::lock_guard<std::mutex> lock_g(_mutex);
```

SUGGESTION

You don't need to lock a mutex within this method. There is no race condition.

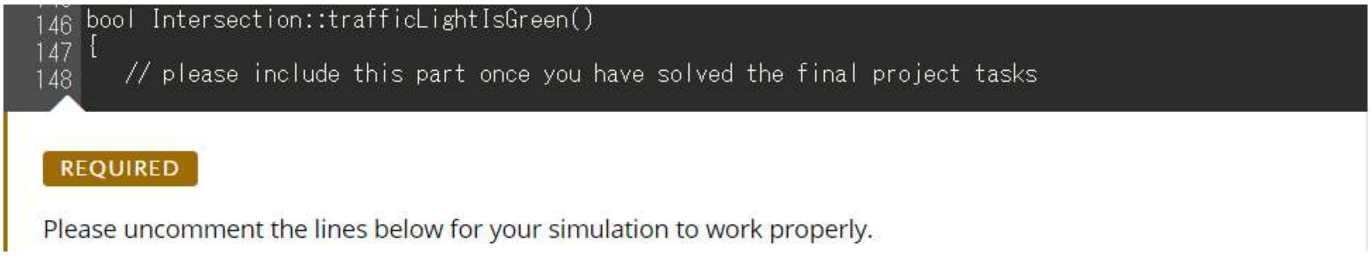
● Intersection.cpp



```
void Intersection::addVehicleToQueue(std::shared_ptr<Vehicle> vehicle)
{
    std::unique_lock<std::mutex> lck(_mtx);
    std::cout << "Intersection #" << _id << "::addVehicleToQueue: thread id = " << std::this_thread::
    lck.unlock();

    // add new vehicle to the end of the waiting line
    std::promise<void> prmsVehicleAllowedToEnter;
    std::future<void> ftrVehicleAllowedToEnter = prmsVehicleAllowedToEnter.get_future();
    _waitingVehicles.pushBack(vehicle, std::move(prmsVehicleAllowedToEnter));

    // wait until the vehicle is allowed to enter
    ftrVehicleAllowedToEnter.wait();
    lck.lock();
    std::cout << "Intersection #" << _id << ": Vehicle #" << vehicle->getID() << " is granted entry."

    // FP.6b : use the methods TrafficLight::getCurrentPhase and TrafficLight::waitForGreen
    // to block the execution until the traffic light turns green.
    if (_trafficLight.getCurrentPhase() == TrafficLightPhase::red)
        _trafficLight.waitForGreen();


    lck.unlock();
```

SUGGESTION

This mutex is meant for the output stream so that each tread can write to the console uninterrupted. If you keep it locked while waiting for green, you might essentially block all other interactions with the intersection (e.g. adding more cars to the queue). I would suggest moving this line before the green phase check.

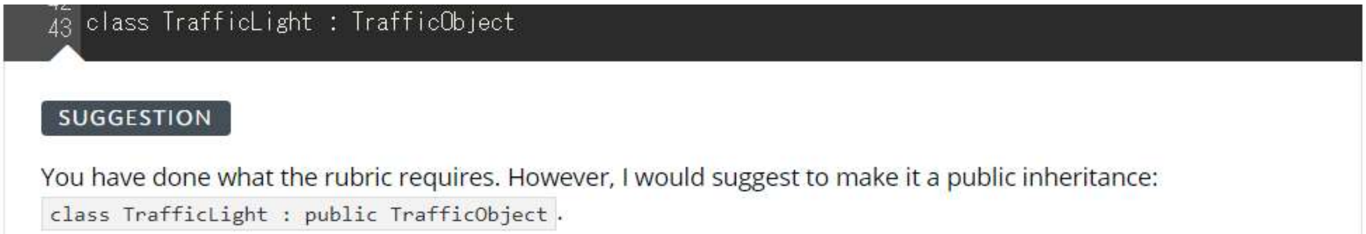


```
bool Intersection::trafficLightIsGreen()
{
   // please include this part once you have solved the final project tasks
```

REQUIRED

Please uncomment the lines below for your simulation to work properly.

● TrafficLight.h



```
class TrafficLight : TrafficObject
```

SUGGESTION

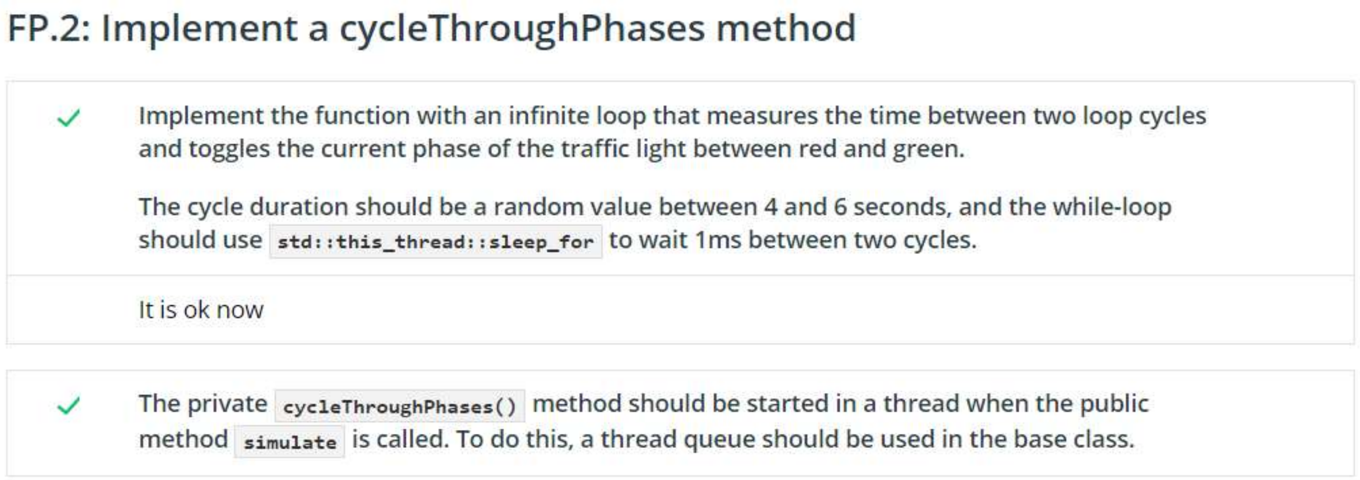
You have done what the rubric requires. However, I would suggest to make it a public inheritance: `class TrafficLight : public TrafficObject`.

Review 2nd

Meets Specifications

Congrats on passing this project, I have added some suggestion for better simulation

Keep working hard 👏



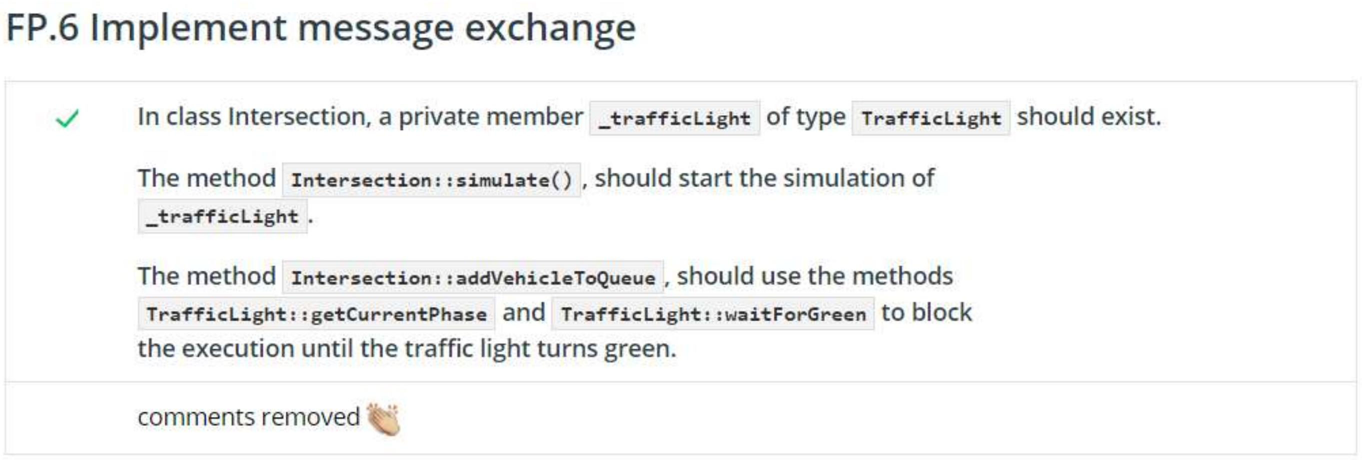
FP.2: Implement a cycleThroughPhases method

✓ 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.

It is ok now

✓ 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.



FP.6 Implement message exchange

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

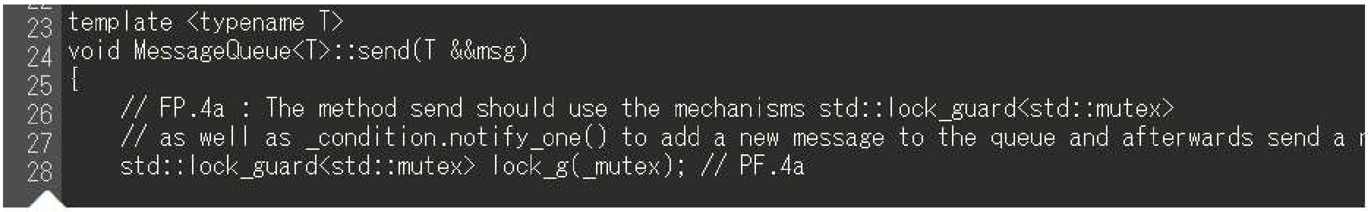
The method `Intersection::simulate()`, should start the simulation of `_trafficLight`.

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

comments removed 👏

Code Review 2nd

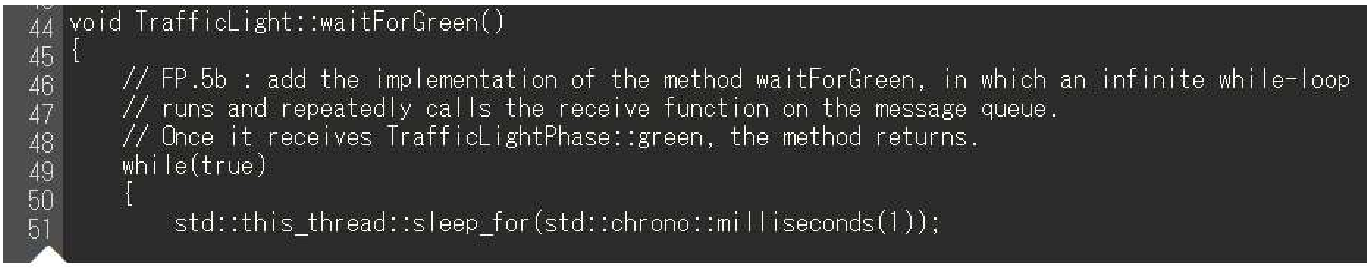
● TrafficLight.cpp



```
template <typename T>
void MessageQueue<T>::send(T &&msg)
{
    // FP.4a : 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
    std::lock_guard<std::mutex> lock_g(_mutex); // PF.4a
```

SUGGESTION

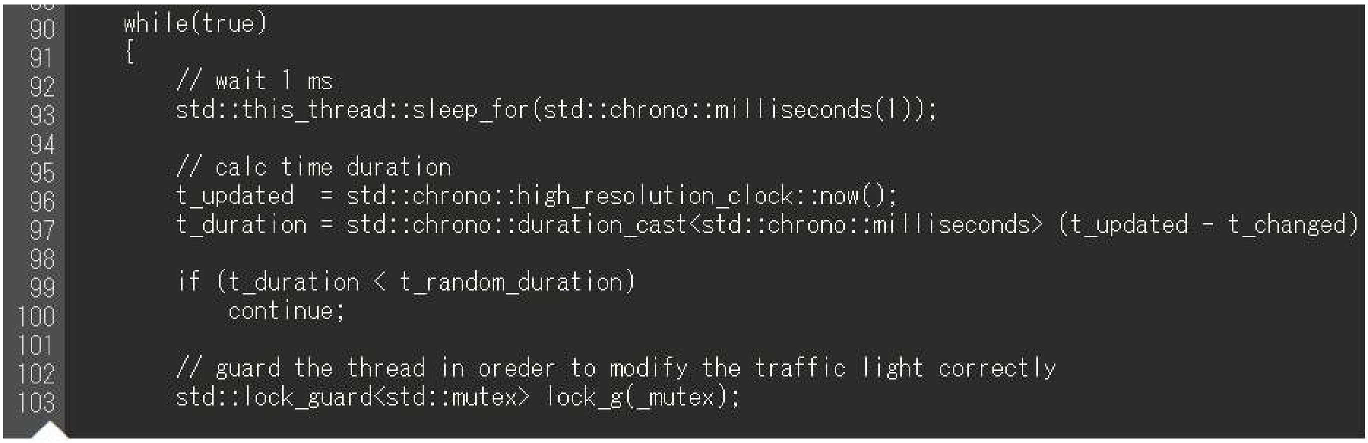
you can add clear here for the queue for better simulation run for the subsidiary traffic lights



```
void TrafficLight::waitForGreen()
{
    // FP.5b : add the implementation of the method waitForGreen, 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.
    while(true)
    {
        std::this_thread::sleep_for(std::chrono::milliseconds(1));
```

SUGGESTION

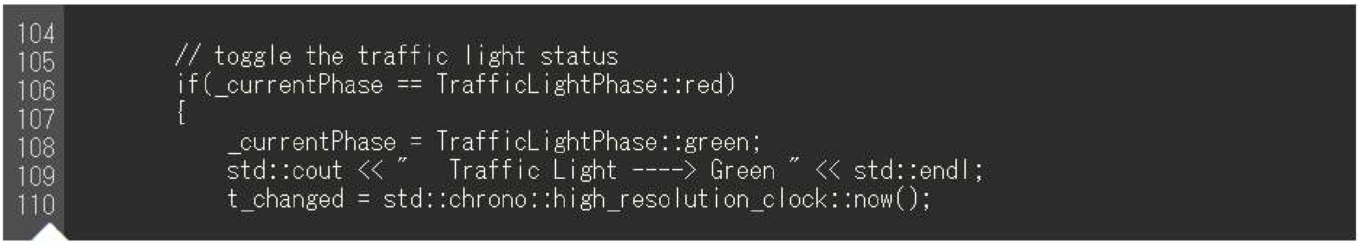
no need for this sleep



```
    while(true)
    {
        // wait 1 ms
        std::this_thread::sleep_for(std::chrono::milliseconds(1));

        // calc time duration
        t_updated  = std::chrono::high_resolution_clock::now();
        t_duration = std::chrono::duration_cast<std::chrono::milliseconds> (t_updated - t_changed)

        if (t_duration < t_random_duration)
            continue;

        // guard the thread in oreder to modify the traffic light correctly
        std::lock_guard<std::mutex> lock_g(_mutex);
```

SUGGESTION

there is no race condition here you can remove mutex



```

        // toggle the traffic light status
        if(_currentPhase == TrafficLightPhase::red)
        {
            _currentPhase = TrafficLightPhase::green;
            std::cout << "    Traffic Light ----> Green " << std::endl;
            t_changed = std::chrono::high_resolution_clock::now();
```

SUGGESTION

t_changed = std::chrono : : high_resolution_clock: :now();
t_random_duration = rand_dist(engine);

those line are duplicated for the two cases so you can put them once outside the two closures