Traffic Light System

Real-time Software Engineering Lab 1

Eslam El-Sherbieny
eselsh@ttu.ee
IVSM 166778

Tallinn University of Technology 2017

Table of Contents

1. Vision

A software for traffic light system using model-based principles and workflows. A main controller acting as the program mainframe, responsible for controlling traffic flow and managing synchronization between vehicle and pedestrian traffic flows. Using timed constraints and user requests to change the traffic flow between them. Following the realistic traffic light system with pedestrian traffic user requests as in real life applications. Technologies used are: Eclipse UML modeling solution, Ecore, Java, etc… Junit and other methods for testing.

2. Requirements

2.1. HLR

2.2. The system shall allow pedestrians to safely cross a street.
2.3. The system's scope is a controller that drives the lights for cars (red, yellow, green) and pedestrians (red, green).
2.4. The system is equipped with two buttons for pedestrians, one on each side of the road.
2.5. The lights for cars and pedestrians must never be green at the same time
2.6. A timer sends a pulse to the controller every second.
2.7. Alternation of the traffic lights follows the standard light regime.
2.8. In the default state the traffic light is green for cars and red for pedestrians
2.9. Reception of a button press shall initiate the sequence for changing the lights to green for pedestrians and red for cars
2.10. Green light for pedestrians shall stay on for X seconds
2.11. Green light for cars shall stay on for at least Y seconds

2.12. LLR

2.12.1. Controller

2.12.2. Description

The main program logic resides here, this class will handle the synchronization protocol between various elements of the program

2.12.3. Responsibilities

2.12.3.1. Synchronization between Pedestrians and Vehicle traffic
2.12.3.2. Time tracking of each traffic state interval
2.12.3.3. Switching states between Pedestrians and Vehicles

2.12.3.4. Switching Traffic lights states
2.12.3.5. Responding to signals from buttons
2.12.3.6. Calculating safety measures for both Pedestrian and Vehicle traffic to determine the correct time for traffic switching and interval of each state

2.12.4. Attributes
2.12.4.1. Time: Integer
An integer counter that holds the current time value
2.12.4.2. pedlight: PedLight
Object containing the pedestrian lights information and status
2.12.4.3. carlight: CarLight
Object containing the vehicle lights information and status
2.12.4.4. car_traffic_active: Boolean
Boolean variable to define current traffic status of vehicle
2.12.4.5. ped_traffic_active: Boolean
Boolean variable to define the current traffic status of pedestrians
2.12.4.6. buttonA_pressed: Boolean
Boolean variable to define if the button A is pressed or not
2.12.4.7. buttonB_pressed: Boolean
Boolean variable to define if the button B is pressed or not
2.12.4.8. car_green_time: Integer
Integer variable to count the time spent for active vehicle traffic, when limit is reached, it should fire the traffic call and to be reset afterwards and initiated again when vehicle traffic light becomes green
2.12.4.9. ped_green_time: Integer
Integer variable to count the time spent for active pedestrian traffic, when limit is reached, it should fire the traffic call and to be reset afterwards and initiated again when pedestrian traffic light becomes green, **Note:** pedestrian light can go green either through time, or button request

2.12.5. Operations
2.12.5.1. switch_lights_buttonA()
This function will call switch_ped_light function when called to immediately initiate a sequence to turn pedestrian light to green.
2.12.5.2. switch_lights_buttonB()
This function will call switch_ped_light function when called to immediately initiate a sequence to turn pedestrian light to green.
2.12.5.3. timer_reset()
Reset the time

2.12.5.4. init()
Initialize the default values (not used)

2.12.6. Timer

2.12.6.1. Description
A time object, initiated with program start and keep a timestamp count
2.12.6.2. Responsibilities
2.12.6.2.1. Keeping time count of the overall process
2.12.6.2.2. Sending pulse signal to the controller every second
2.12.6.3. Operations
2.12.6.3.1. tick()
Sends new time value on each call
2.12.6.3.2. timer_reset()
Reset the time

2.12.7. Enumerations

2.12.8. Lights
2.12.8.1. Description
Enumeration holding the three lights variables {Red, Yellow, Green}
2.12.8.2. Attributes
2.12.8.2.1. Red: enumeration literal
2.12.8.2.2. Yellow: enumeration literal
2.12.8.2.3. Green: enumeration literal
2.12.9. DefaultState (not used)
2.12.9.1. Description
Enumeration holding the default state values of the lights
2.12.9.2. Attributes
2.12.9.2.1. PedLightState: enumeration literal
2.12.9.2.2. CarLightState: enumeration literal

2.12.10. PedLight

2.12.11. Description
Object containing the pedestrian light info
2.12.12. Attributes
2.12.12.1. ped_current_light: Lights
Integer value containing the current status of the light, default is 0 for Red
2.12.13. Operations

2.12.13.1. switch_ped_light()
Start switching sequence from one state to the next, **Note:** if the status is green it will return to yellow then red - this will be done in sync with switch_car_light()

2.12.14. CarLight

2.12.15. Description
Object containing the vehicle light info
2.12.16. Attributes
2.12.16.1. car_current_light: Lights
Integer value containing the current status of the light, default is 2 for Green
2.12.17. Operations
2.12.17.1. switch_car_light()
Start switching sequence from one state to the next, **Note:** if the status is green it will return to yellow then red - this will be done in sync with switch_ped_light()

2.12.18. Buttons A & B

2.12.18.1. Description
Objects for button A and B that represents the buttons that would activate abnormal traffic switch from vehicle to pedestrians
2.12.18.2. Responsibilities
2.12.18.2.1. Receive button push from user to activate switch
2.12.18.3. Attributes
2.12.18.3.1. activate():
When button pressed this function call the traffic switch functions to change traffic on the respective side from vehicle to pedestrians

3. Test Case Descriptions

3.1. Scenario 1

Where car traffic is active and time reached the switching state

3.1.1. Controller check light status for car and ped

3.1.2. If car traffic is active and car time is equal X

3.1.3. Activate switching process for both car and ped in parallel

3.1.4. Car light is equal 0 (RED) and ped light is equal 2 (GREEN)

3.1.5. Car traffic stopped, pedestrian traffic resumed

4. Test Procedure Descriptions

4.1. Junit test

4.1.1. ButtonATest

4.1.1.1. testActivate()

Test activate method

4.1.2. ButtinBTest

4.1.2.1. testActivate()

Test activate method

4.1.3. PedLightTest

4.1.3.1. testSwitch_ped_light()

Call switch_ped_light() then check if getPed_current_light() changed value or not

4.1.4. CarLightTest

4.1.4.1. testSwitch_car_light()

Call switch_car_light() the check if getCar_current_light() changed the value or not

4.1.5. TimerTest

4.1.5.1. testTick()

Call tick() function and check if time is incremented

4.1.5.2. testTimer_reset()

Not implemented

4.1.6. ControllerTest

4.1.6.1. testSwitch_lights_buttonA

Call switch_lights_buttonA() and call tick() 4 times, then check if isCar_traffic_active() is true or false, and that time is incremented

4.1.6.2. testSwitch_lights_buttonB

Same as previous test

4.1.6.3. testTimer_reset()

Not implemented

5. Coverage Report

- 5.1. Implemented parts
  - 5.1.1. Class diagram
  - 5.1.2. Sequence diagram
  - 5.1.3. Usecase diagram
  - 5.1.4. Classes code
  - 5.1.5. Implementation code
  - 5.1.6. Test unit code
- 5.2. Implemented tests
  - 5.2.1. ButtonA
  - 5.2.2. ButtonB
  - 5.2.3. Controller
  - 5.2.4. PedLight
  - 5.2.5. CarLight
  - 5.2.6. Timer

6. Verification Report

7. Successful tests:
   - 7.1. testTimer_reset
   - 7.2. testSwitch_car_light
   - 7.3. testTick
8. Failed tests:
   - 8.1. testSwitch_lights_buttonA
   - 8.2. testSwitch_lights_buttonB
   - 8.3. testSwitch_ped_light
9. Error failed tests
   - 9.1. ButtonATest
   - 9.2. ButtonBTest