SPL 151 Assignment 2

Publshed - 14/11/2013 11:30
Dead line - 28/11/2013 13:00

1. Before You Start

- It is mandatory to submit all of the assignments in pairs. Assignment 1 and assignment 2 must be submitted with the same partner. It is recommended to find a partner as soon as possible and **create a submittal group** in the submission system. Once the submission deadline has passed, it will not be possible to create submission group even if you have an approved extension.
- Read the assignment together with your partner and make sure you understand the tasks. Please do not ask questions before you complete reading the assignment.
- We will use INI files to read input from and write output to. You can read about INI file format here: http://en.wikipedia.org/wiki/INI_file
- Write a skeleton of the assignment (i.e. interfaces and class structure).

2. City driving simulator

In this assignment, we will build a traffic simulator of a road system. We will design a system to reflect the state of the roads and junctions at any given time. We are given a description of the road map, and a list of events, and we will update the state of our system accordingly. The events include the addition of new cars, and car faults that may occur. We are given commands according to which we will produce reports on the system state.

2.1 Assignment Goals

The objective of this assignment is to design an object-oriented system and gain implementation experience in C++ while using classes and standard STL-based data-structures. You will learn how to handle memory in C++ and avoid memory leaks. You will learn to work with a 3rd party library - Boost
– This library will help you work with the ini files. The resulting program must operate efficiently as will be explained.

3. General Overview

Simulation systems are used to describe how a complex system of interacting objects can evolve over time. Each object acts according to specific rules, and it can react to external events or to the action of other objects. Simulation systems are important research tools to analyze complex systems when no analytical framework exists to predict the evolution of a system given the rules of the single elements. They are used extensively for digital circuit analysis and weather prediction.
In a discrete simulation system, the simulator works according to the following design. The participants are a simulation controller and simulated objects. The state of the system corresponds to the state of all the simulated objects present in the system at any given time. Each simulated object has its own internal state (a set of properties, i.e. the location of a car and its speed) and connections to other simulated objects (i.e. a car is located on a road).
The simulation controller simulates the progress of time as follows: it runs a loop and manages an internal clock. At each iteration, the clock advances by a set amount of time (a measure known as the "time slice" of the simulation). For each time slice, the simulation controller sends an event to all existing simulated objects asking them to "react" to the fact

that "time progresses". The simulation controller can also trigger external events according to a simulation script (i.e. make a new car enter the system at a given clock time).

For each time slice, simulated objects re-compute their internal state, possibly re-compute the set of objects they are connected to, and as they make transitions from one state to another, they may send events to the objects they are connected to, which in turn will react.

The simulation terminates when the controller reaches a set condition (for example after the internal clock has reached a set limit or when there are no more cars in the system etc.).

The traffic system we simulate includes three types of **actors**: Junctions, Roads and Cars. In addition, there are two types of **events**: car arrival and car fault and two types of **commands**: termination and report.

4. System Constants

The following constants appear in Configuration.ini file:
MAX_SPEED: the maximal speed of a car in the simulation.
DEFAULT_TIME_SLICE: the initial number of time units for green light.
MAX_TIME_SLICE: the maximal possible number of time units for green light.
MIN_TIME_SLICE: the minimal possible number of time units for green light.

5. Simulated Objects

Junctions
Each junction has unique ID which is a string with no spaces. The junction should manage traffic in the following way:

- At each time unit, only one incoming road can have a green light.
- At the beginning of the system, the number of time units for each green light should be DEFAULT_TIME_SLICE. This value can change during the simulation according to traffic load as will be explained later.
- Scheduling green lights is in Round-Robin manner. That is only one incoming road will have green light at each time unit in circular order. The order of green lights is defined in the RoadMap.ini input file.

Roads
Each road is one-way. If there is a 2-way road between two junctions, it will be considered as two different roads. Each road connects two junctions (starting and ending junctions) and has a length (in meters). On each road there are cars. The location of a car on a road is given by the distance of the car from the beginning of the road ([0..length]).
The driving speed of a car in the road is a function (will be given later) of the length of the road, the number of cars that are currently driving on that road, and the number of faulty cars ahead of that car in the road.

Cars
Each car has a unique id string and a road plan which is a sequence of junctions the car is going to visit during the simulation. A car will be inserted into the simulator at a certain time given in the car arrival event.
A car may be in good condition or faulty. A car may experience a fault, which is an event that causes the car to hold in a fixed position for a given amount of time. A car fault also affects other cars in the road as will be discussed below.

6. System Events

All events will appear in Events.ini file with no guaranteed order. Each event will be in a different section. Section names will be [event_#] where the # is a number.

Car Arrival

A car arrival event adds a new car to the simulation. Each event consists of the time when the car should be added to the simulation, the car's ID and the car's road plan (itinerary).
Each car arrival event will have the following format:

```
[event_#]
type=car_arrivel
time=<time>
carId=<carId>
roadPlan=<startJunction>, <middleJunction,>* <endJunction>
```

Car Fault

A car fault event causes a certain car to enter a faulty state for a given amount of time. Each event consists of the time the car should enter the faulty state, the car's ID and the duration the car should remain in faulty state. A car that becomes faulty with duration T will miss exactly T steps of advancing on the road. When a car in faulty state gets another a fault event, the new fault duration should be added to the current fault duration. A car in a fault state in a junction cannot proceed in green light and other cars will pass it.
Each car fault event will have the following format:

```
[event_1]
type=car_fault
time=<time>
carId=<carId>
timeOfFault=<time to remain in fault>
```

An example for Events.ini file:

```
[event_2]
type=car_arrivel
time=1
carId=C1
roadPlan=j1,j2,j4

[event_3]
type=car_arrivel
time=3
carId=C2
roadPlan=j3,j2,j4

[event_4]
type=car_arrivel
time=3
carId=C3
roadPlan=j2,j3,j4

[event_5]
type=car_fault
time=4
carId=C1
timeOfFault=3
```

7. Commands

The simulation controller can execute commands found in the simulation script. Commands are the way we use to get information on the state of the simulated system. There are 2 types of commands in our system, reports and termination. There are 3 types of reports. Commands will be given in Commands.ini file in no guaranteed order. Each command will be in a different section. Section names will be [comman_#] where # is a number. The format and example of Commands.ini file will be given in the end of this section.

Termination

This command causes the system to stop the simulation. Note that upon exit, our program should return all resources it uses to the operating system.
The command's parameter is the time when the simulation should stop.

Reports

Each report has an ID (string). It will help us connec the report commands to their output. All report outputs should be written to a file Reports.ini.

Car Report

This command produces a report concerning a given car. The parameters of the command are:

- The time when the report should be generated.
- The ID of the report.
- The car's ID.

The reports output should have the following format:

```
[<ReportID>]
carId=<carId>
history=(<timeId>,<startJunctionId>,<endJunctionId>,location>)*
faultyTimeLeft=<faultyTimeLeft> // zero if not-faulty.
```

Example of a car report:

```
[report_1]
carId=C1
history=(1,j1,j2,0)(2,j1,j2,20)(3,j1,j2,40)(4,j1,j2,60)
faultyTimeLeft=3
```

This report output means that car C1 was
At time 1: on the road connecting j1 and j2 at distance 0 from j1
At time 2: on the road connecting j1 and j2 at distance 20 from j1
At time 3: on the road connecting j1 and j2 at distance 40 from j1
At time 4: on the road connecting j1 and j2 at distance 60 from j1

Road Report

This command produces a report concerning a given road. The parameters of the command are:

- The time when the report should be produced.
- The ID of the report.
- The starting and ending junctions of the road.

The reports output should have the following format:

```
[<ReportID>]
```

```
startJunction=<startJunctionId>
endJunction=<endJunctionId>
cars=(<carId,location>)*
```

Example of road report:

```
[report_2]
startJunction=j1
endJunction=j2
cars=(C1,60)
```

This report output means that at the time of the report the only car on the road connecting j1 and j2 was C1 and it was at distance 60 from j1.

The report should consist of each car currently in the road, its ID and location, sorted by the location of the car on the road. In case two cars are in the same location, the order should be the order they go to the location. If there is more than one car arrival event at the same time unit to the same road (which means that the cars got to the same location at the same time) then they should have the same order of the corresponding events in Events.ini. If there are no cars on that road than the value of the "cars" key should be "" (cars=).

Junction Report

This command produces a report concerning a given junction. The parameters of the command are:

- The time when the report should be produced.
- The ID of the report.
- The junction's ID.

The reports output should have the following format:

```
[<ReportID>]
junctionId=<JunctionId>
timeSlices=(<timeSilce>,x)*
<incomingJunctionId>=(<carId>)*
```

Example of Junction report:

```
[report_3]
junctionId=j3
timeSlices=(2,0)(2,-1)
j2=(C3)
j1=
```

This report output means that at the time of the report, the time slice for the first incoming junction is 2 and it has a green light for 0 time units. The time slice for the second incoming junction is 2 and it has a red light.

The report should consist of each incoming junction (at the same order they appear in the RoadMam.ini file) the current time slice and the number of time units it had a green light (-1 for incoming junctions with red light). In addition, for each incoming junction, it should have the list of cars standing in line on the road in the same order they will go through the junction (their position in the queue managed by the junction).

The Commands.ini format is:

```
[command_#]
type=termination
time=<termination time>

[command_#]
type=car_report
```

```
time=<time>
id=<reportId>
carId=<carId>

[command_#]
type=road_report
time=<time>
id=<reportId>
startJunction=<startJunction of read to report>
endJunction=<endJunction of read to report>

[command_#]
type=junction_report
time=<time>
id=<reportId>
junctionId=<junctionId>
```

An example for commands.ini file:

```
[command_1]
type=termination
time=50

[command_2]
type=car_report
time=4
id=report_1
carId=C1

[command_3]
type=road_report
time=5
id=report_2
startJunction=j1
endJunction=j2

[command_4]
type=junction_report
time=6
id=report_3
junctionId=j3
```

NOTE: your output must match EXACTLY the requirements of this specification - including spaces and capital/small letters. You are not required to the sort the different reports in the file according to a certain order.

8. Input Files

The input for our simulator will be given in 4 different files :

- RoadMap.ini: contains the road map, junctions, roads and their length.
  - The format of each junction in the file is:

```
[<Junction 1 name>]

<Incoming junction 1>=<length of road from incoming junction 1>

<...>

<Incoming junction k>=<length of road from incoming junction k>
```

  - An example RoadMap.ini file:

```
[j1]
j4=300

[j2]
j1=100
j3=50

[j3]
j2=50
j1=100

[j4]
j2=50
j3=75
```

That means that:

There is a 300 meters road from j4 to j1
There is a 100 meters road from j1 to j2
There is a 50 meters road from j3 to j2
There is a 50 meters road from j2 to j3
There is a 100 meters road from j1 to j3
There is a 50 meters road from j2 to j4
There is a 75 meters road from j3 to j4

The order of incoming junction in each section is important. This is the order the junction will manage its green lights. In this example, in junction j3, cars coming from j2 will have green light prior to cars coming from j1.

- Configuration.ini: contains constant values used in the assignment.
  - Configuration.ini format:

```
[Configuration]

MAX_SPEED=<MAX_SPEED value>

DEFAULT_TIME_SLICE=<DEFAULT_TIME_SLICE value>

MAX_TIME_SLICE=<MAX_TIME_SLICE value>

MIN_TIME_SLICE=<MIN_TIME_SLICE value>
```

  - An example Configuration.ini file:

```
[Configuration]
MAX_SPEED=20
```

```
DEFAULT_TIME_SLICE=3
MAX_TIME_SLICE=10
MIN_TIME_SLICE=2
```

- Events.ini: contains the events that the simulator should process as described above.
- Commands.ini: contains the commands that the simulator should execute as described above.

9. Output

The output will be in a single file, Reports.ini, which will contain all the requested reports that were given in the commands.

10. Simulation Workflow

The system is initialized according to the configuration files. After the system is initialized, it works according to the following algorithm:

Until termination command, in each time unit:

1. Execute the events of current time unit.
2. Execute the commands of current time unit.
3. Advance cars in roads.
4. Advance cars in junctions.

- **Important note**- the output of the reports should be consistent with the simulation state just after step 1. This means, prior to cars advancing.

Detailed description of car advancing:

Advance Cars in Roads

At this stage, we change the location of cars within the roads. The location of each car cannot exceed the length of the road. If the location after advancing it becomes bigger than <length> it will automatically be set to <length>. The way cars are advanced on roads is:

- First, determine the base speed of the road, that is, ceiling(<length> / <number of cars in the road>)
- Each faulty car causes all the cars driving behind it (not including cars in the same location) to decrease their speed to half (rounded to ceiling). If there is more than one faulty car ahead, the speed of the car behind will be decreased to half (rounded to ceiling) several times.
- Increase the location of the car by its speed. (Without exceeding the length of the road).
- Cars that are faulty at this step are not advanced at all.

Cars are advanced according to their position on the road. If two cars reach the end junction of the road in the same time unit, the order they will be inserted to the junction queue should be the same as the order they had on the road. If a car is advanced to the same location as another (faulty) car than it is considered to be behind it (the order of cars in the same location is defined by the order they got to the location).

Advance Cars in Junctions

Note that we advance the cars in roads before advancing them in junctions.

We advance cars in junctions in the following way:

- At each time unit, only one car can pass the junction. The first car in the incoming road that currently has green light will move into its destination outgoing road (with location 0).
- If the number of time units this incoming road had green light has reached the time slice for this incoming road (this is time for rotation of the green light):
    - Switch the green light to the next incoming road (according to the given order).
    - Update the time slice of the incoming road that has just finished its time slice according to the number of cars that passed the junction in the time slice:
        1. If the time slice was fully used (at every time unit a car passed the junction): newTimeSlice = min(oldTimeSlice+1, MAX_TIME_SLICE).
        2. If the time slice was not used at all (no cars passed the junction during the whole time slice): newTimeSlice = max(oldTimeSlice-1, MIN_TIME_SLICE).
        3. Else, the time slice is being left as it is. (newTimeSlice = oldTimeSlice).

11. Implementation Requirements

11.1 Data Maintenance and Efficiency

There are some basic design issues you must pay attention to, before you start implementing the application.

1. An implementation that wastes memory will have points removed. For example, an implementation in which each road of length x holds an array of size x for cars driving on it is wasteful.
2. Each data piece should be held exactly once in memory (you can have many references to it).
3. Efficiency requirements for the simulations:
    1. Each simulation step with no event / report should take O(n log n + k) where n is the number of cars currently in the simulation and k is the number of junctions.
    2. We must be able to access a car, a junction, and a road in logarithmic time.

11.2 Representing events and commands in the system:

Though there are many ways to represent an event or command in the system, we ask you to use the following implementation.

```
class Event
{
	public:
		virtual void performEvent()=0;
};
```

From which you will inherit two classes

a) AddCarEvent
b) CarFaultEvent

```
class Report
{
      public:
            Virtual void writeReport()=0;
};
```

From which you will in inherit two classes

a) CarReport
b) RoadReport
c) JunctionReport

You are allowed to add data members and member function to both classes to suit your needs.

Memory leak

This requirement is basic and mandatory. Memory leaks occur when the program fails to release memory that is no longer needed. A memory leak can diminish the performance of the computer by reducing the amount of available memory. Eventually, in the worst case, too much of the available memory may become allocated and all or part of the system or device stops working correctly, the application fails, or the system slows down. To avoid leaks, you need to release the memory when you finish using it. Basically, every time you use ***new***, you must use ***delete***.

You must check your implementation using Valgrind. Valgrind is an easy to use tool that is already installed in the lab on Linux platforms. For easy start read this.

12. Submission

Your submission should be in a single zip file called "student1ID-student2ID.zip". The files in the zip should be set in the following structure:

src/
include/
bin/
makefile

src directory should include all .cpp files that are used in the assignment.
include directory should include all the header (.h or *.hpp) files that are used in the assignment.
bin directory where the compiled objects (the .o files and the assignment executable) should be placed when compiling your source files (should be empty).
makefile should compile the cpp files into the bin folder and create an executable and place it also in the bin folder (should be named *RoadSimulator*).

The *makefile* should properly compile on the department computers, the objects should be compiled with the following flags:

–Wall –Weffc++ –g

The submissions must be made in pairs.

After you submit your file to the submission system, re-download the file that you have just submitted, extract the files and check that it compiles.

13. Grading

Although you are free to work wherever you please, assignments will be checked and graded on CS Department Lab Computers - so be sure to test your program thoroughly on them before your final submission. "But it worked on my Windows/Linux based home computer" will not earn you any points if your code fails to execute at the lab.

14. Questions

All questions regarding the assignment should be published in the assignment forum. Please search the forum for similar questions before publishing a new one. Question by mail regarding the assignment will be redirected to /dev/null. Please note that before using the forum you must register here.

15. Delays

In case of reserve duty (aka Sherut Miluim), illness, etc. please send an email to the course mail majeek@cs.bgu.ac.il. The mail should include partners name and IDs, explanation and certification for your illness / Miluim. We will not answer other mail in the course mail.

16. Course policy about appeals

The policy is very simple, **There are no appeals!**

Please make sure you made all the necessary QA on your work *before* submitting it. It is highly recommended to download your submission, compile and run it on one of the lab computers on a clean folder.