Hospital Emergency Room Simulation: Object Oriented Design Issues for CS2

Viera K. Proulx
College of Computer Science, Northeastern University
Boston, MA 02115
vkp@ccs.neu.edu

Abstract

This paper describes a project suitable for students in CS2 that combines the classical implementation of basic data structures (priority queues, lists, arrays) with the design and implementation of several interesting interacting classes. In addition, students can follow up with analysis of simulation results.

1 Introduction

The second computer science course for majors (CS2) traditionally focused on basic algorithms and data structures. The introduction of object oriented design and libraries like the STL is changing the way the course is taught. Instructors look for projects that reinforce the study of classical algorithms and data structures and at the same time provide an environment for presenting an object oriented programming and design style. In addition, instructors want the students to gain the experience in using the standard class libraries.

We describe a CS2 project that provides a rich context for exploring object oriented design, algorithms, data structures, and, optionally, the analysis of empirical data. Students use or implement several typical CS2 data structures and algorithms, together with several classes, each representing a different design pattern. The project is simple enough to complete in two weeks and can easily be run on any platform..

Partial support for this work has been provided by the National Science Foundation Leadership in Laboratory Development , award #DUE- 9650552

SIGCSE'99, February, 1999.

Project Description

The project is a simulation of the flow of patients through an emergency room of a hospital with a given number of beds. The simulation focuses on how long it takes to treat patients based on the severity of their problems and the constraints on the number of beds.

The user of the simulation program can vary the discrete probability distributions of three events: patient arrivals, the severity of the patient's problem, and the estimated length of treatment in minutes. At each minute, the number of arriving patients is generated. As each patient is admitted, he is assigned a priority code based on the severity of the perceived problem and is also assigned an estimated length of treatment. The program fills all of the available beds with the patients that have problems of the highest severity. The remaining patients wait in the waiting room. In a simple version, the actual length of treatment can be set to be the same as the estimated length of treatment. Alternately, a simple modification of the estimated time may be introduced randomly to simulate more accurately the real life situation.

The program collects summary statistics about the number of patients, their arrival times, the waiting times, the treatment time, the severity of the problem, and possibly the bed utilization. The data is saved to a file and can be analyzed using a spreadsheet program.

The user has a number of options. The data that specifies the discrete probability of the three events is read from an input file and can vary from one run of the simulation to another. The user can also choose the number of available beds and the length of the simulation run.

We do not include in the simulation a doctor or nurse object. For simplicity, we assume that there is a health care professional available to deliver treatment to any patient that has been assigned to an available bed. By including the doctor/nurse availability issue, the project can be extended to a reasonably complex term project.

Students are given a fairly detailed description of the objects and classes they should use but the final design decisions are theirs. We believe that at this stage students are not ready to design an entire project. They need to start

with a well thought out design to which they can contribute.

2 Objects and Classes

We look at the classes students need to implement and examine the design options that students have to consider.

3.1 Patients and Beds

The first object we think of represents a patient. The member data of this class represents fairly faithfully the timing aspects of a real world patient chart. We record the time of arrival, the assigned priority code, the time when the patient has been assigned a bed, and the time of discharge.

We can discuss whether the patient's priority code should remain static throughout the stay in the hospital or whether changes are allowed. We then look at the implications for design.

Next we consider the object that will group all patients together. There are several choices here. The patients in the waiting room seem to correspond to a priority queue. However, once a patient has been assigned to a bed, the patient's record is removed from the priority queue. On the other hand, we need to represent the one-to-one (though transient) relationship between a patient and a bed.

The patient object needs to migrate between two different collection structures over the period of its existence and be properly destructed at the end. One solution to this problem is to use a reference to the patient object in both the priority queue and when assigning a patient to a bed. A discharged patient destructor then follows the discharge procedure that includes reporting its statistics to a data collection object.

The bed object itself is very simple. It only needs to know whether a bed is available, or which patient is using the bed. A reference to a patient object can encode both pieces of information - a null reference indicates that the bed is available.

Next we decide on the structure that will keep track of all beds. We can have two lists - a list of free beds and a list of beds in use, or we can use a vector (array-like) structure that will need to be traversed in search of empty beds each time. A third alternative is to have an array of beds with a free-bed chain. The tradeoffs between the three implementations can be discussed and students can choose any one of these. At this point, students may also choose to use one of the library supported structures, such as a STL list or a STL vector, if they are working in C++.

3.2 The Discrete Probability Distribution

This is in some ways the most interesting class in this project. We can think of it as an extension of a scaling function class used in [3]. It encapsulates nicely the notion of black box functional behavior.

Let us first describe the discrete probability distribution, using as example the patient's expected length of treatment. Suppose the expected length of treatment is 3 min for 50% of patients, 10 min for 30% of the patients, 30 min for 15% of patients and 60 min for 5% of the patients.

Expected length of treatment:
3/50%, 10/30%, 30/15%, 60/5%

The value of the generated event is the time (3, 10, 30, 60) and each event is generated with the given probability. As a new patient arrives, he is assigned one of the four values of the expected length of treatment that have been generated according to the stated probabilities.

The other two objects in this class generate the number of arriving patients and the severity code for patient's problem. We chose not to correlate the severity code with the expected length of treatment.. A choking patient may require only a few minutes of treatment while taking care of a broken arm may take more than an hour. We have used initially the following values and their corresponding probabilities:

Patients arriving:
0/50%, 1/25%, 2/15%, 3/10%

Severity code:
1/60%, 2/20%, 3/10%, 4/5%, 5/5%

The discrete probability distribution class needs to record as its private member data a list of event values that will be generated and the corresponding probabilities. The class contains two key member functions. A constructor or an Initialize function that will read from a given input stream the number of distinct events and a list of pairs; event value, event probability. It should also initialize the random number generator as needed. The second function Generate returns a value of the next event, using the given discrete probability distribution. So, if the object instance representing the treatment time distribution is called TreatmentTime, the code in the simulation would be:

```
ttime = TreatmentTime.Generate();
```

We could choose to replace the function Generate with the operator (), depending on how comfortable the students and the instructor are with the C++ syntax. (We do not have this option in Java.) The key lesson here is that classes and objects can encapsulate functional behavior [1]. It is clear that during the simulation run the data representing the discrete probability distribution remains static and is of no interest to the simulation itself - and therefore should be hidden.

3.3 Collecting Statistics

There are several options for collecting the results of the simulation. The easiest is to save the results to a file. To make this file usable for further analysis, each line should represent the data for one discharged patient, with individual data items separated by tabs. This format is readable by most spreadsheet applications.

Another option is to create a dynamically growing array of patient discharge data objects. The STL vector class is well suited for this use.

In any case, eventually, students should be able to see and plot the results for at least several different runs of the simulation.

3.4 Designing Experiments

We may continue with designing a suite of simulation runs, where all decisions - the length of the run, the number of beds, and the three discrete probability distributions are read from a file and several runs are done during one execution of the program.

At this point, students work on designing the proper user interface, use input files in a meaningful setting, and work on the design of experiments. The driver for running several simulations in a row that uses input files for modifying the parameters of each run is similar to the mechanism used when designing a test suite for a program.

3 Practical Considerations

This project can be presented to students in a number of different ways, depending on student abilities, the time available, and the instructor's pedagogical goals.

4.1 Presenting the Project

The most straightforward method for assigning this project is to supply students with fairly complete details of the design of the needed classes and leave them only a few choices. We may for example insist that the bed collection be implemented as a library vector class if we want them to have the experience of using the library classes.

If the instructor wants to introduce or reinforce the use of standard design notation (e.g. UML) the project can be presented as an UML design document from which students start coding. For a more challenging assignment, students may first create a design document from the narrative description before they proceed with coding.

Of course, one can start with a brief description of the problem and hold several class discussions during which the students examine the design options and decide on the final design. It may be that, after discussing the design, each student is allowed to make her own choices for the implementation.

Finally, students should design the suite of test runs. They may be asked to prepare a document similar to a test document that will list the tests that will be conducted, specify the input parameters, and maybe even predict the expected outcomes.

5.2 Data Analysis

Students in CS1 and CS2 have little opportunity to examine results of experiments and analyze data. This is a valuable skill, used in all types of performance analysis. The data collected in this simulation is straightforward and understandable. Students can see what happens if more beds are available, measure the utilization of beds, model the busier times in the emergency room (e.g. Saturday night) when the number of patients and the severity of their problems increase, etc. They can look at data and compare the behavior for the different sets of input parameters.

4 Conclusion

Simulations always provide a good environment for illustrating interesting design issues and programming practice [2, 4]. The attractive feature of the project presented here is the fact that it can be easily adopted, it illustrates the use of priority queues, and it uses a function-like object to produce event values. Additionally, the project provides the context and the data for simple data analysis.

5 Acknowledgements

Partial support for this work has been provided by the National Science Foundation Leadership in Laboratory Development Grant, DUE-9650552, and by a grant from Microsoft Corporation. The materials discussed in this article may be obtained in both PC and Macintosh format at:

http://www.ccs.neu.edu/home/vkp/index.html

The author wishes to thank Richard Rasala and Harriet Fell for thoughtful discussions about this work.

6 References

[1] Rasala, R., Function Objects, Function Templates, and Passage by Behavior in C++, ACM SIGCSE Bulletin 29 (1), 35-38, 1997.

[2] Proulx, V. K., Traffic Simulation: A Case Study for Teaching Object Oriented Design, ACM SIGCSE Bulletin 30 (1), 48-52, 1998.

[3] Fell, H. J., Proulx, V. K., and Rasala, R., Scaling: A Design Pattern in Introductory Computer Science, ACM SIGCSE Bulletin 30 (1), 326-330, 1998.

[4] Moser. M, Elevator Simulation Project: University of Colorado, Boulder, Private Communication.