Emergency Room Simulator

Sam Wynsma and Geoff Rich

Requirements Specification

Our program will simulate a hospital emergency room over a period of at least one week. There are 2000 possible patients and their names are stored in a text file. They all have an equal probability of entering the emergency room. User will set average patient arrival rate, the number of doctors and/or nurses, and the simulation time.

When a patient arrives, they are assigned an illness priority number from 1-20 (20 is high priority, 1 is low priority). 1-10 is 70% likely, 11-15 is 20% likely, and 16-20 is 10% likely. Patients with priority 1-10 will be placed in a low-priority priority queue and patients with priority 11-20 will be placed in a high-priority priority queue.

There are two types of Physicians: Nurses and Doctors. Doctors can treat all patients, and Nurses can only treat patients with a priority of 1-10. Nurses take 1-10 minutes to treat a patient and Doctors take 1-20 minutes to treat a patient. The treatment time will be decided randomly every time they accept a patient.

Every minute, we do three things. First, we check for new patients based on the patient arrival rate. If one arrives, we generate a priority number and add them to the appropriate queue. Second, we check for available doctors. If there is one, they take a new patient from the appropriate queue. Last, we check if any patients are done being treated. If they are, we remove them from the emergency room and make their Physician available. We then update their record, the total number of patients, and the total treatment time.

Every Patient has a corresponding Hospital Record. It stores the number of visits to the emergency room and their priority number for each visit. At the end of the program, the user can look up a record by name and have it displayed. They can also display the names of all treated patients.

At the end of the program, statistics are displayed: average treatment time, total treatment time, and total number of patients.

Use Cases

User Inputs at the start of the program

|  |  |
| --- | --- |
| User | Program |
|  | Asks the user to input in a number of doctors into the hospital. |
| User inputs the number of doctors that the hospital should have. | The program stores the input. |
|  | The hospital now asks the user how many nurses that it should have. |
| User inputs a number of nurses. | The program stores the input |
|  | Program asks for number of hours to run simulation |
| User inputs number of hours | The program stores the input |
|  | Program asks for number of patients per hour. |
| User inputs the amount of patients will come per hour. | The program stores the input. If this number >= 60, the program will ask the question again. |
|  | Patient arrival rate and number of hours are stored in the appropriate variables. |
|  | The available physicians list is populated with specified number of doctors and nurses. |

Display menu at end of simulation

|  |  |
| --- | --- |
| User | Program |
|  | Output statistics: total patients, total visit time, average visit time, number of untreated patients, patients treated by each physician. |
|  | Output menu options: list all patients treated, look up an individual record by name, or quit the program. |
| User enters number corresponding to desired option | Do what the user requested. After completion, give them the option to return to the menu or quit the program. |

Access record at end of simulation

|  |  |
| --- | --- |
| User | Program |
|  | Prompt user for name of patient |
| User enters name of patient | Look up patient’s record in the Hospital Record map. |
|  | If the patient exists, output the number of visits and priority number for each visit. Otherwise, inform the user that the patient does not exist. |

Pseudocode

Get People from File

* Create file stream object
* Open text file using file stream object
* If unsuccessful, output error message and abort
* Else read each line of file into TownspeopleNames vector
* Close the file
* Initialize TownspeopleStatus to be the same size as TownspeopleNames, every index initialized to false

Add Patient

* If number of people in emergency room is less than the number of people in the town
  + Grab a random name from the list of townperson names until you get one that isn’t in the emergency room (based on EmergencyStatus vector)
  + Set corresponding index in EmergencyStatus to true and increment number of people in emergency room
  + Create a patient object with that name, an illness number based on a random number generator, and ID (the random index)
  + If illness number is greater than 10
    - Place patient in the high illness priority queue
  + Else (patient number is from 1-10)
    - Place patient in the low illness priority queue.

Treat Patient

* If there are no available physicians, return
* If there are patients in the low priority queue
  + For each available physician, if they’re a nurse
    - Dynamic cast from Physician\* to Nurse\*
    - Set their patient to the top of the low priority queue
    - Pop the queue
    - Set their treatment time randomly
    - Place nurse in the unavailable doctors queue.
    - Remove nurse from the available doctors queue.
  + Break if the low priority queue is empty
  + For each available physician, if they’re a doctor
    - Dynamic cast from Physician\* to Doctor\*
    - If there are patients in the high priority queue
      * Set their patient to the top of the low priority queue
      * Pop the queue
      * Set their treatment time randomly
      * Place doctor in the unavailable doctors queue.
      * Remove doctor from the available doctors queue.
    - Else if there are patients in the low priority queue
      * Do the same as above
    - Else return

Remove patient

* For every unavailable physician, if they’re a nurse
  + Dynamic cast from Phsyician\* to Nurse\*
  + If the time that treatment ends is equal to the clock
    - If they’re not in the HospitalRecord yet, add them
    - Else update their record with this visit
    - Remove the patient from the nurse
    - Set their TownspeopleStatus to false
    - Decrement number of people in emergency room
    - Add nurse to available physicians
    - Remove nurse from unavailable physicians
    - Increment total patients treated and add total visit time for patient to the total visit time in the hospital
* If they’re a doctor, do the same thing (but different dynamic cast)

Display Hospital Record

* Attempt to find from given key (name)
* If the key is in the map
  + for each item in PriorityNumber vector, output index (aka visit number) followed by the priority number for the visit
* Else say they can’t be found

UML

Class Emergency Room

* List<Physician\*> Available\_Physicians
* List<Physician\*> Unavailable\_Physicians
* Priority\_Queue<Patient\*> High\_Priority\_Patients
* Priority\_Queue<Patient\*> Low\_Priority\_Patients
* Vector<string> TownspeopleNames
* Vector<bool> TownspeopleStatus
* Map<String, Record> HospitalRecord;
* Int PatientArrivalRate;
* Int totalPatients;
* Int timeSpent;
* Int SickPeople
* Int NumOfMinutes

+ void DisplayMenu();

+ void LoadNamesFromFile()

+ void InitializeVariables

+ void AddPatient(int);

+ void RemovePatient(int);

+ void TreatPatient(int);

+ getters for disease number, arrival rate, minutes, and patients

+ EmergencyRoom()

Class Doctor

+ Doctor()

+ void SetTreatmentTime(int)

Class Nurse

+ Nurse()

+ void SetTreamentTime(int)

Virtual Base Class Physician

# vector<Patient\*> TreatedPatients

# Patient\* CurrentPatient

# int TreatmentTime;

# int InitialTime

# virtual void SetTreatmentTime(int)

+ virtual void RemovePatient();

+ getters for TreatmentTime, IntialTime, Num of Patients, and CurrentPatient

+ void SetPatient(Patient\*)

Class Patient

- Int priorityNumber;

- String Name;

- Int ID

- Int EnterTime

+ Patient(int, string, int, int)

+ getters and setters for Name, ID, PriorityNumber

+ getter for EnterTime

+ bool operator< (const Patient&)

Class Record

* Int numberOfVisits;
* Vector<int> PriorityNumber;

+ Record(int, int)

+ Record()

+ getters for visits and priority number

+ void AddToRecord(int)