Rachel Wofford and Jordan Bone

CS 273

**Requirements Specifications**

The purpose of this simulation is to simulate the events of a hospital waiting room in 273ville minute-by-minute over a one-week time period (10,080 minutes). There are 2000 people in 273ville who all have an equal chance of needing to go to the emergency room. At the beginning of the simulation, the user will input the average hourly patient arrival rate. This value should not be more than 60. Exceptions will be used to test this value. The user’s input will then be divided by 60 to get the average patient arrival rate per minute. The process patients go through once they arrive at the hospital is to wait in a queue that begins at the waiting room, then moves to the emergency room, and ends with discharge.

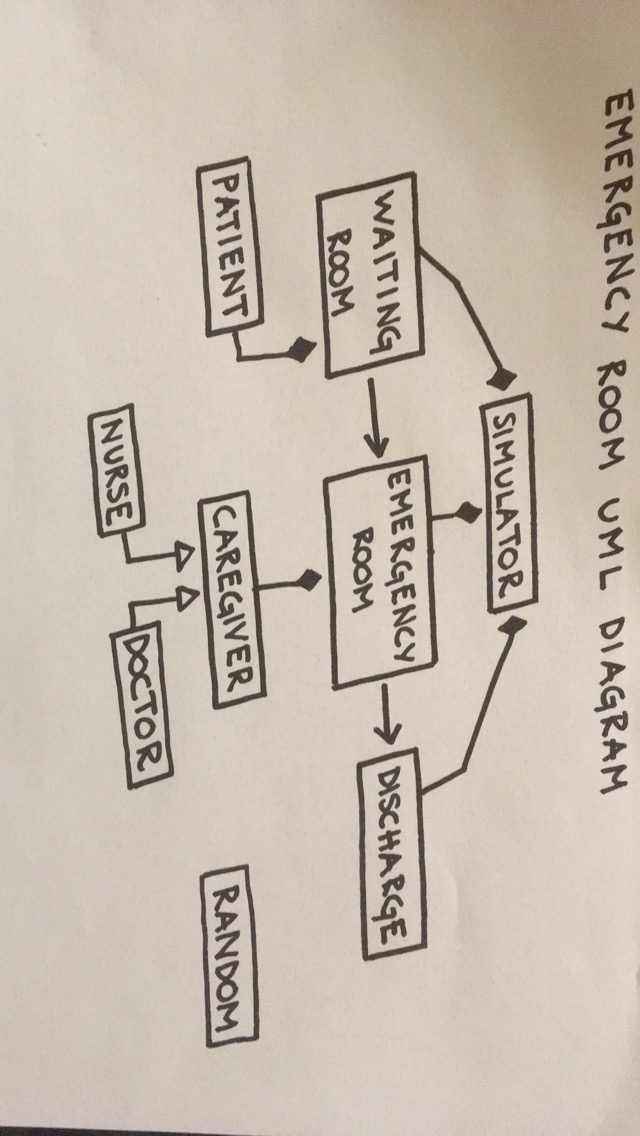
When the patient arrives in the waiting room, they will be assigned a priority number based on the severity of their illness. The patient’s name and priority value of illness will be stored and their number of visit to the emergency room will be incremented. A vector will hold the number of visits and priority values for those visits of each patient. Later, the user will be able to access these records for each patient based on the patient’s name. The priority number will be between 1 and 20. Priority 1-10 illnesses occur 70% of the time. Priority 11-15 illnesses occur 20% of the time. Priority 16-20 illnesses occur 10% of the time. All illnesses within each priority level occur with equal probability. In this simulation, these values will be assigned through a random number generator and will factor in the different probabilities. In the waiting room, higher priority values mean the patient’s illness is more serious, so higher priority patients will be treated first. This will be done using a priority queue. Additionally, when a patient first arrives in the waiting room, their arrival time will be recorded so their total wait time from arrival to discharge can be calculated later.

The first patient in the priority queue will move from the waiting room to the emergency room when the proper type of caregiver becomes available. There will be two types of caregivers: Doctors and Nurses. The Doctor and Nurse classes will be derived from the main Caregiver class. The number of Doctors and Nurses in the simulation will be input by the user at the beginning of the simulation. This input will be tested using unit testing. Nurses are capable of treating patients with priority values of 1-10. Doctors can treat patients with priority values of 1-20. This means the nurse can’t always take the first person in the priority queue. If someone in the queue has a priority value between 11-20, they must wait for a doctor. This will be incorporated into the simulation through an assignment of the priority values to each patient.

Once in the emergency room, Nurses can treat patients in anywhere from 1-10 minutes and Doctors can treat patients in 1-20 minutes. In the simulation, this amount of time will be assigned using a random number generator for each type of caregiver. After this amount of time has passed, the patient will be discharged and the doctors/nurses no longer have to worry about them. The patient’s discharge time will be stored. Then, their total wait time will be calculated by subtracting their arrival time from their discharge time. This value will be added to the total wait time for all patients. At the end of the simulation the total wait time for all patients will be used to calculated the average wait time for patients.

When the simulation is over, a menu will be displayed giving the user the options to display all of the residents of 273ville who have been treated or retrieve the record of a user by “name”, which will be stored for each user when they are discharged. The user may also choose to quit. Additionally, the user will be able to see the average visit time from arrival to discharge for emergency room patients. This will be calculated by dividing the total amount of patient wait times by the number of patients served.

**UML Diagram**

****

|  |
| --- |
| **Simulator** |
| -const int total\_time  -int clock  -int num\_doctors  -int num\_nurses  -WaitingRoom \*patient\_priority\_queue  -Discharge \*discharge\_queue  -vector <Caregiver\*> caregivers  -int read\_int() |
| +Simulator () { }  +vector<Caregiver\*> get\_Caregiver\_Vector() {}  +void data\_entry()  +void update\_caregivers()  +void run\_simulation()  +void display\_menu()  +void ListOfPatients()  +void FindPatient()  +friend class EmergencyRoom |

|  |
| --- |
| **Waiting Room** |
| -double arrival\_rate  - Priority\_queue <Patient \*> patient\_priority\_queue |
| + WaitingRoom () {}  + void setArrivalRate()  +void update()  +friend class EmergencyRoom |

|  |
| --- |
| **Emergency Room** |
| -int num\_doctors  -int num\_nurses  -Simulator \*s  -Caregiver \*caregiver\_pointer  -Discharge \*patient\_discharge\_queue  -WaitingRoom \*wr\_patient\_priority\_queue |
| +Emergency Room (Simulator \*s1) {}  +void set\_wr\_patient\_priority\_queue()  +void set\_patient\_discharge\_queue()  +void set\_num\_doctors()  +void set\_num\_nurses()  +void add\_patient\_to\_discharge()  +void add\_patient\_to\_treatment\_queue()  +friend class Discharge  +friend class Caregiver |

|  |
| --- |
| **Discharge** |
| -queue<Patient \*> discharge\_queue  -vector<Patient \*> treated\_patients  -multiset<Patients> records  -int total\_wait\_time  -int num\_served  -int patient\_wait |
| **+**Discharge() : total\_wait\_time(0), num\_served(0) {}  +int get\_num\_served()  +int get\_total\_wait()  +multiset<Patient> get\_records()  +vector<string> get\_treated\_patients()  +void update  +friend class EmergencyRoom |

|  |
| --- |
| **Caregiver** |
| -EmergencyRoom \*emergency\_room  -bool is\_busy |
| +Caregiver(EmergencyRoom \*emergency\_room) {}  +virtual void set\_care\_time()  +virtual int get\_care\_time()  +virtual int get\_clearance\_level()  +void set\_is\_busy()  +void set\_is\_empty()  +bool get\_is\_busy()  +friend class EmergencyRoom |

|  |
| --- |
| **Doctor** |
| -const int clearance\_level  -int care\_time  -EmergencyRoom \*emergencyroom |
| +Doctor(EmergencyROom \*emergency\_room) : Caregiver(emergency\_room) {}  +void set\_care\_time()  +int get\_care\_time()  +int get\_clearance\_level() |

|  |
| --- |
| **Nurse** |
| -const int clearance\_level  -int care\_time  -EmergencyRoom \*emergencyroom |
| +Nurse(EmergencyRoom \*emergencyroom) : Caregiver(emergency\_room) {}  +void set\_care\_time()  +int get\_care\_time()  +int get\_clearance\_level() |

|  |
| --- |
| **Patient** |
| +Patient(int clock, int priority\_level, string p\_name) : priority\_level(priority\_level), name(p\_name), patient\_visits(-1), arrival\_time(clock), start\_treatment\_time(-1), treatment\_time(-1) {}  +Patient(string n) {}  +int priority\_level  +string name  +int patient\_visits  +int arrival\_time  +int start\_treatment\_time  +int treatment\_time  +Caregiver \*cg  +vector<string> treated\_patient\_names  +void setName()  +string getName()  +void set\_treatment\_time()  +void set\_caregiver()  +void set\_start\_treatment\_time()  +int get\_start\_treatment\_time()  +void set\_patient\_visits()  +int get\_patient\_visits()  +int get\_arrival\_time()  +vector<string> getTreatedNames()  +double getPriority() const  +multiset<Patient> patient\_record  +void set\_patient\_record()  +multiset<Patient> getPatientRecord()  +static string getRandomName()  +bool operator()  +friend class Simulator |

|  |
| --- |
| **Random** |
| - |
| +Random () {}  +Random (int seed) {}  +int next\_int ()  +double next\_double () |

**Use Cases**

**User starts simulation:**

|  |  |  |
| --- | --- | --- |
| Step | User’s Action | System’s Response |
| 1 | Start simulation |  |
| 2 |  | Output welcome statement |
| 3 |  | Prompt user for average hourly patient arrival rate |
| 4 | Input average hourly patient arrival rate |  |
| 5 |  | Call read\_int function to check that user’s input is valid |
| 6 |  | If user’s input is invalid, repeat steps 3-5 until input is valid |
| 7 |  | Prompt user for number of doctors |
| 8 | Input number of doctors |  |
| 9 |  | Call read\_int function to check that user’s input is valid |
| 10 |  | If user’s input is invalid, repeat steps 7-9 until input is valid |
| 11 |  | Prompt user for number of nurses |
| 12 | Input number of nurses |  |
| 13 |  | Call read\_int function to check that user’s input is valid |
| 14 |  | If user’s input is invalid, repeat steps 11-13 until input is valid |

**Patient arrives at waiting room:**

|  |  |  |
| --- | --- | --- |
| Step | Simulation’s Action | System’s Response |
| 1 | Patient arrives at waiting room |  |
| 2 |  | Assign patient a priority number |
| 3 |  | Add patient to queue – placement in queue will be dependent on priority number and number of other people in the waiting room  If no one else is in the waiting room, this patient will be the first in the queue  If this patient has a priority number that is higher than any other patients in the waiting room, they will be at the front of the queue  If this patient’s priority number is not the highest or lowest priority number, they will be assigned a place in the queue relative to the other patient’s priority numbers  If this patient’s priority number is the lowest priority number in the waiting room, they will be at the end of the queue |
| 4 |  | Add patient’s name to vector of patients served |
| 5 |  | Record patient’s arrival time |
| 6 |  | If patient is a new patient, create a vector to keep track of the priority numbers of the patient’s trips to the emergency room  Otherwise, store this visit in the patient’s vector of priority numbers that has already been created |
| 7 |  | Increment patient’s number of trips to the emergency room |

**Nurse becomes available:**

|  |  |  |
| --- | --- | --- |
| Step | Simulation’s Action | System’s Response |
| 1 | Nurse becomes available to treat a patient |  |
| 2 |  | Examine if first patient in queue can be treated by a nurse (patient has a priority level of 1-10)  If first patient cannot be treated by a nurse, iterate through the queue until the first patient that can be treated by a nurse is found |
| 3 |  | Remove patient from waiting room queue |
| 4 |  | Move patient to the emergency room by adding this patient to the queue of patients that nurses are treating |
| 5 |  | Calculate a random treatment time for this patient (between 1-10 minutes for a nurse) |

**Doctor becomes available:**

|  |  |  |
| --- | --- | --- |
| Step | Simulation’s Action | System’s Response |
| 1 | Doctor becomes available to treat a patient |  |
| 2 |  | Remove first patient from waiting room queue (regardless of priority level because doctors can treat all priority levels) |
| 3 |  | Move patient to the emergency room by adding this patient to the queue of patients that doctors are treating |
| 4 |  | Calculate a random treatment time for this patient (between 1-20 minutes for a doctor) |

**Nurse finishes working with a patient:**

|  |  |  |
| --- | --- | --- |
| Step | Simulation’s Action | System’s Response |
| 1 | Randomly calculated treatment time for this patient is complete |  |
| 2 |  | Remove patient from emergency room queue |
| 3 |  | Record patient’s discharge time |
| 4 |  | Calculate patient’s total visit time (discharge time – arrival time) |
| 5 |  | Add patient’s total visit time to the total wait time for all patients |
| 6 |  | Increment the number of patients treated |
| 7 |  | Nurse becomes available to treat another patient |

**Doctor finishes working with a patient:**

|  |  |  |
| --- | --- | --- |
| Step | Simulation’s Action | System’s Response |
| 1 | Randomly calculated treatment time for this patient is complete |  |
| 2 |  | Remove patient from emergency room queue |
| 3 |  | Record patient’s discharge time |
| 4 |  | Calculate patient’s total visit time (discharge time – arrival time) |
| 5 |  | Add patient’s total visit time to the total wait time for all patients |
| 6 |  | Increment the number of patients treated |
| 7 |  | Doctor becomes available to treat another patient |

**Calculate and display the average wait time:**

|  |  |  |
| --- | --- | --- |
| Step | Simulation’s Action | System’s Response |
| 1 | 10,080 minutes of simulation have finished |  |
| 2 |  | Retrieve total wait time for all patients and number of patients treated |
| 3 |  | Calculate average wait time in this emergency room (total wait time for all patients/number of patients treated) |
| 4 |  | Display this value to user in menu |

**Display menu:**

|  |  |  |
| --- | --- | --- |
| Step | Simulation’s Action | System’s Response |
| 1 | 10,080 minutes of simulation have finished |  |
| 2 |  | Output menu to user with options to:   1. List names of all residents who were treated 2. Retrieve the record of a resident by name 3. Average wait time for patients in the ER 4. Quit |

**User chooses to list names of all residents who were treated:**

|  |  |  |
| --- | --- | --- |
| Step | User’s Action | System’s Response |
| 1 | Choose menu option to list names of all residents who were treated |  |
| 2 |  | Retrieve patient’s names from vector of patients who have been served |
| 3 |  | Output list of name to user |

**User chooses to receive the record of a resident by name:**

|  |  |  |
| --- | --- | --- |
| Step | User’s Action | System’s Response |
| 1 | Choose menu option to retrieve the record of a resident by name |  |
| 2 |  | Output patient’s name |
| 3 |  | Output number of times patient went to emergency room |
| 4 |  | Retrieve vector of priority numbers associated with emergency room visits |
| 5 |  | Output severity of illness for each emergency room visit to user |

**User chooses to receive average wait time:**

|  |  |  |
| --- | --- | --- |
| Step | User’s Action | System’s Response |
| 1 | Choose menu option to retrieve average wait time |  |
| 2 |  | Output average wait time |

**User chooses to quit:**

|  |  |  |
| --- | --- | --- |
| Step | User’s Action | System’s Response |
| 1 | Choose menu option to quit |  |
| 2 |  | Output goodbye statement to user |
| 3 |  | End program |

**Pseudo-Code**

**Simulator**

read\_int()

1. If low entry is greater than high entry, throw invalid range exception.
2. Prompt user to try entry again until a valid numeric string is input.

data\_entry()

1. Output welcome statement to user
2. Prompt user to input average hourly patient arrival rate
   1. Validate user’s input for average hourly patient arrival rate using read\_int function
   2. Calculate the average patient arrival rate per minute by dividing average hourly patient arrival rate by 60.0
3. Prompt user to input number of nurses
   1. Validate user’s input for number of nurses using read\_int function
   2. Add a new nurse object through caregiver polymorphism
4. Prompt user to input number of doctors
   1. Validate user’s input for number of doctors using read\_int function
   2. Add a new doctor object through caregiver polymorphism

update\_caregivers(int clock)

1. Update the treatment and discharge queues with each clock tick
   1. Add patient to discharge if it is available
   2. Add patient to treatment\_queue if it is available

run\_simulation()

1. For each clock tick while clock is less than total time
   1. Update the waiting room functions
   2. Loop through each caregiver and update their current status
   3. Update discharge functions

display\_menu()

1. Output menu to user
2. If user chooses to list names of all residents who were treated:
3. If user chooses to receive the record of a resident by name:
   1. Call FindPatient()
4. If user chooses to receive the average wait time
   1. Calculate average wait time
   2. Display average wait time
5. If user chooses to quit:
   1. Output goodbye statement to user
   2. End program

ListOfPatients()

1. Retrieve patient’s names from vector of all patients who have been served
2. Output these names to user

FindPatient()

1. User inputs patient’s name
   1. Iterator through the multiset of treated patients
2. Output number of times patient went to emergency room
   1. Retrieve vector of priority numbers associated with each visit
3. Output the priority number of each visit to user

**WaitingRoom**

update() in WaitingRoom

1. Generate a random name for a patient
2. Generate a random priority level
3. Check if the random number generated is less than the average patient arrival rate
4. Assign patient a priority number
5. Record patient’s arrival time
6. Create patient with
   1. Random name
   2. Name priority number
   3. Arrival time
7. Add patient to priority queue

**EmergencyRoom**

add\_patient\_to\_discharge()

1. Check if a patient in the ER is ready to be discharged
   1. Put this patient at the front of the treatment queue
   2. Check is full treatment time has elapsed
   3. Remove patient from treatment queue
   4. Add patient to priority queue
2. Find the patients caregiver
   1. Set the caregiver to empty so they can be assigned a new patient

add\_patient\_to\_treatment\_queue()

1. If ER is empty AND there is a patient in the waiting room
   1. Create a new patient object pointing to the first Patient in the waiting room queue
   2. Loop through caregivers and determine who is available to treat a patient
   3. Sort through patients in the waiting room until one is found who can be treated by the caregiver
      1. Going from highest to lowest priority
   4. Assign caregiver to patient and remove them from previous vectors

**Discharge**

update ()

1. Check if there are any patients in the discharge queue
   1. Remove patient from discharge queue
   2. Update patient record
   3. Calculate total visit time
   4. Update:
      1. Total wait time for all patients
      2. Total patients served
   5. Add patient to queue of treated patients

get\_treated\_patients()

1. Return a vector of treated patients so it can be accessed by other classes
   1. Allows access to medical records

**Patient**

getRandomName()

1. Run only once:
   1. Open files with list of last and first names
      1. Assign the strings in these files to a vector of last names and a vector of first names
   2. Generate a random number that allows access to a random name in each vector
   3. Combine the randomly chosen first and last names into a full name
      1. Add the full name to a separate vector
2. Access a random full name from the full name vector
   1. Return the full name.