Project 1 - Health Center Simulation

Arquiteturas de Software 2021/2022 - Universidade de Aveiro

Tiago Rainho 92984
Diogo Mendonça 93002

1. Introduction

In this work, our goal is to simulate the management process in a small health center, using a program with concurrent processes and threads. The simulation is made up of two main entities:

- Health Center Process (HCP) - process responsible for the patients' treatments;
- Control Center Process (CCP) - process responsible for controlling the simulation.

In this simulation, each patient will start by entering the hospital, and then enter in multiple distinct halls, in order to have his illness evaluated and cured, finishing with the payment of his medical appointment, while always waiting for his turn. All the patients' movements are tracked using a logger and a GUI.

2. Threads

This program includes 5 types of active entities, each with their corresponding non-active counterpart:

- Patient - Entity that tries to enter and leave each hall in succession as fast as possible. Characterized by its ID, age group (child or adult), and degree of severity (DoS) of illness.
- Nurse - Responsible for evaluating the DoS of patients' illnesses. The DoS can be Red, Yellow or Blue, and the Nurse has an equal chance of choosing one of them (random generator with uniform distribution). Evaluations last for a variable amount of time (EVT).
- Doctor - Responsible for curing the patients. Treatments last a variable amount of time (MDT).
- Cashier - Responsible for receiving the treatments' payment. Payments take a variable amount of time (PYT).
- Call Center - Supervisor of the patients' movement.

3. Health Center Process

The Health Center Process includes all classes related to the different types of threads and their non-active counterpart, the hospital (ManchesterHospital), halls and corresponding monitors, rooms and the simulation logger.

The hospital encompasses 5 halls:

- Entrance Hall (ETH);
- Evaluation Hall (EVH);
- Waiting Hall (WTH);
- Medical Hall (MDH);
- Payment Hall (PYH);
- Call Center Hall (CCH).

When a new instance of ManchesterHospital is created, all previously mentioned halls and corresponding monitors are initialized.

In the developed program, most halls (except PYH and CCH) include a certain number of rooms, where patients wait for their turn, are evaluated or treated. Both the halls and the rooms provide the methods for the patients to enter the place, be processed and then leave the place following a pre-established order. To ease this task, a priority queue and corresponding monitor were developed, alongside two subclasses: MSeatRow and MWaiting.

MSeatRow is a class that is used to represent a defined number of seats in each room of the hospital. This class implements the methods seat (patient attempts to sit in the room) and move (patient gets up and leaves the room). The visual representation of the rooms in the GUI is also done here, using a GuiBox array, with each GuiBox instance corresponding to a seat.

MWaiting is a class that is used to represent a waiting queue in the halls and it has no maximum size. In terms of methods, this class is very similar to MSeatRow, also providing a seat and a move method. Despite having an unlimited size of seats, MWaiting only shows a maximum of 10 seats in the GUI, prioritizing the ones nearest the head of the priority queue.

The GuiBox class extends the jTextField java class and uses its methods to create square boxes (seats in rooms) or rectangular boxes (waiting queue entries in hall) that allow represent the patient's ID with text, the patient's age group with the background (white for children and gray for adults), and the patient's DoS of illness with the border (Red, Yellow or Blue), as can seen in Figure 1.

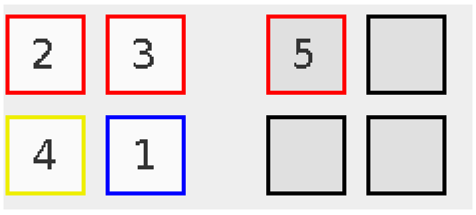
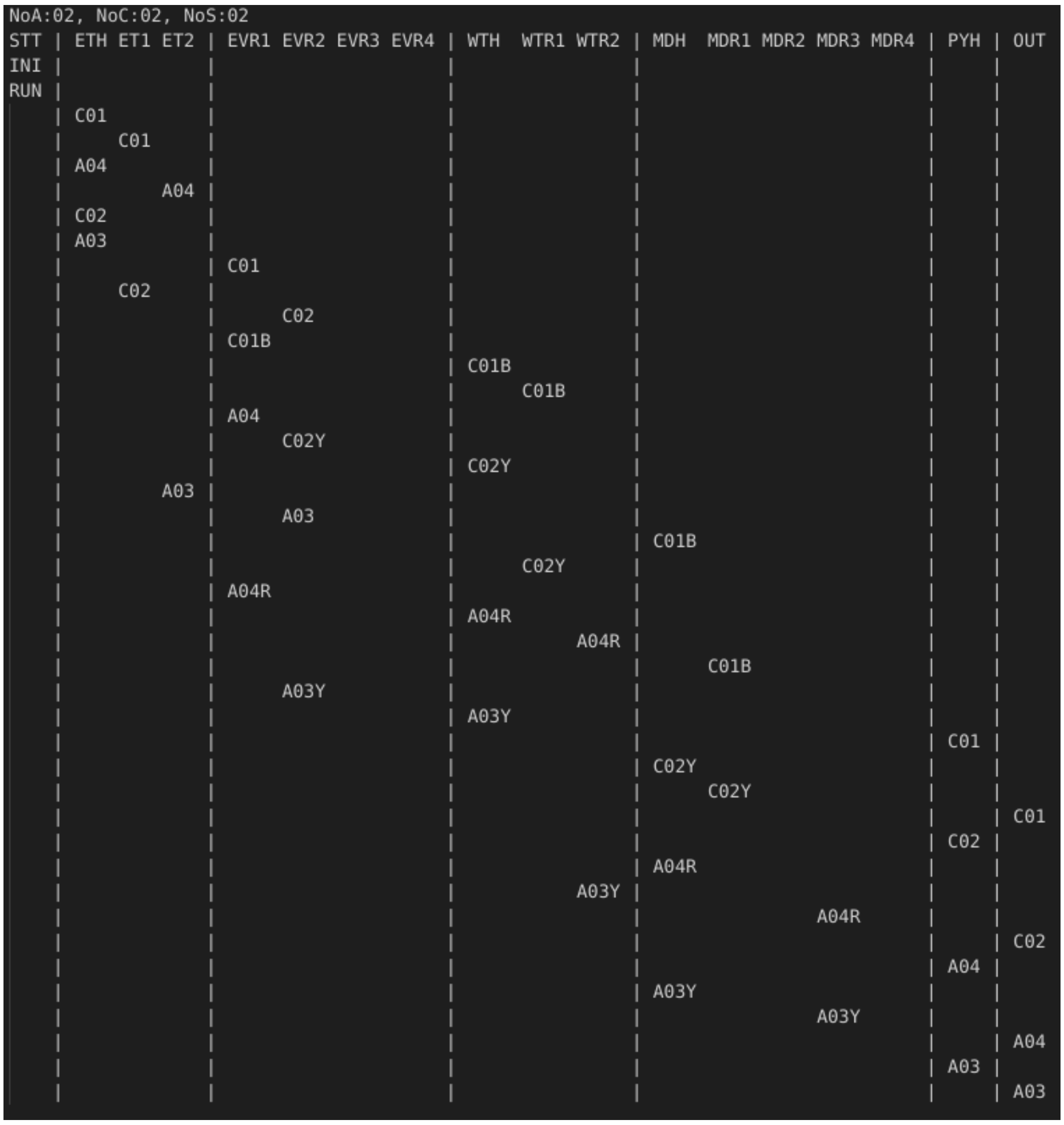


Figure 1: Patients coded by DoS, age group and ID. Note that empty seats do not have text and have black borders to signify no ID and no DoS, respectively.

The Call Center Hall is the entity responsible for the patients' movement, providing methods to wait and notify an event, using notification packets. This allows for patients to notify that a seat is empty when they leave the room they are in, and another patient receives that information and advances to his next room.

The simulation Logger is a singleton class, which means that only one instance of this class is possible and it is easily accessible globally using the method getInstance(). This class is responsible for printing to the console and to a text file (log.txt) all changes in the simulation status and all patients' movements and diagnosis. An example can be seen in Figure 2.



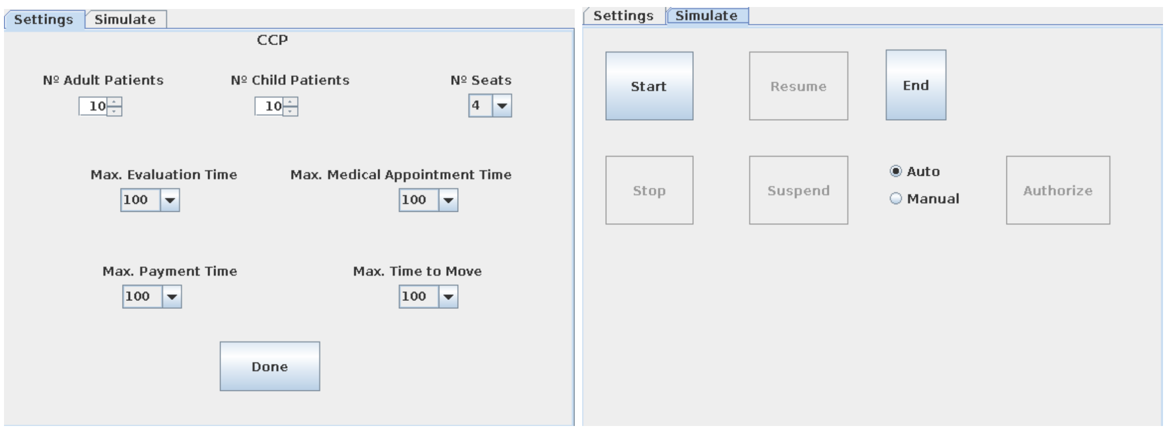
```
NoA:02, NoC:02, NoS:02
STT | ETH ET1 ET2 | EVR1 EVR2 EVR3 EVR4 | WTH  WTR1 WTR2 | MDH  MDR1 MDR2 MDR3 MDR4 | PYH | OUT
INI |             |                     |                |                          |     |
RUN |             |                     |                |                          |     |
    | C01         |                     |                |                          |     |
    |     C01     |                     |                |                          |     |
    | A04         |                     |                |                          |     |
    |         A04 |                     |                |                          |     |
    | C02         |                     |                |                          |     |
    | A03         |                     |                |                          |     |
    |             | C01                 |                |                          |     |
    |     C02     |                     |                |                          |     |
    |             |      C02            |                |                          |     |
    |             | C01B                |                |                          |     |
    |             |                     | C01B           |                          |     |
    |             |                     |      C01B      |                          |     |
    |             | A04                 |                |                          |     |
    |             |      C02Y           |                |                          |     |
    |             |                     | C02Y           |                          |     |
    |         A03 |                     |                |                          |     |
    |             |      A03            |                |                          |     |
    |             |                     |                | C01B                     |     |
    |             |                     |      C02Y      |                          |     |
    |             | A04R                |                |                          |     |
    |             |                     | A04R           |                          |     |
    |             |                     |           A04R |                          |     |
    |             |                     |                |      C01B                |     |
    |             |      A03Y           |                |                          |     |
    |             |                     | A03Y           |                          |     |
    |             |                     |                |                          | C01 |
    |             |                     |                | C02Y                     |     |
    |             |                     |                |      C02Y                |     |
    |             |                     |                |                          |     | C01
    |             |                     |                |                          | C02 |
    |             |                     |                | A04R                     |     |
    |             |                     |           A03Y |                          |     |
    |             |                     |                |                A04R      |     |
    |             |                     |                |                          |     | C02
    |             |                     |                |                          | A04 |
    |             |                     |                | A03Y                     |     |
    |             |                     |                |                A03Y      |     |
    |             |                     |                |                          |     | A04
    |             |                     |                |                          | A03 |
    |             |                     |                |                          |     | A03
```

Figure 2: Log of simulation with 2 adults and 2 children.

4. Control Center Process

The Control Center Process has the main goal of controlling the simulation, being able to change its multiple parameters and execution state through a GUI, by communicating with the HCP using sockets.

The CCP GUI is divided into two parts, each located in one panel of the same frame, as can be seen in Figures 3 and 4. The settings panel is responsible for the adjustment of the simulation's parameters, like the number of seats in certain halls, the maximum evaluation and medical appointment time and the number of adult and child patients. The simulate panel provides buttons to give control over the simulation's state.



Nº Adult Patients 10
Nº Child Patients 10
Nº Seats 4
Max. Evaluation Time 100
Max. Medical Appointment Time 100
Max. Payment Time 100
Max. Time to Move 100

[✓ Auto] Manual

Figures 3 and 4: Panels of CCP GUI

6. Conclusion

In this work, we were able to discover what are the mechanisms used when trying to simulate a hospital's management system, knowledge that can be also used in other possible simulation scenarios.

We could also better understand how to synchronize multiple threads and processes in Java, by using locks and conditions to coordinate threads, and sockets to provide communication between two processes (in this case, HCP and CCP).