Jaamsim Lab 2 – Extended Radiology Clinic

In this lab we will extend the simulation developed in the previous lab to include a priority value for patients, use a priority order for scanning, and make the scanners require half an hour of maintenance every 8 hours. The maintenance should only occur when the machine is free, if it is busy when the 8 hours are up the maintenance should take place the next time it is free. We assume that there are 5 levels of priority (1, 2, 3, 4, 5) and more important patients (lower value of priority e.g. 1 is more important than 2) are always seen before any patients of lower priority. In addition, priority 1 and 2 patients are urgent so they do not need to check in, they go directly to queueing for a scan. We want to use the simulation to determine the 90th percentile of time that patients spend in the clinic, between arriving and leaving. In addition we want to compare the time that the different priority levels spend in the clinic.

Once again, since the aim of the lab is to learn Jaamsim, the conceptual model is not given here, instead it is available separately in the file Radiology_Clinic_2_Conceptual_Model.pdf.

1 Experiments

In this lab we will perform one experiment with 50 replications each 1 week long. We will use the same distributions for the interarrival time, check in time, and scan duration for appointment patients as in the previous lab. The proportion of each type of patient in each priority group is given in Table 1:

Table 1: Patient Priority Proportions

Priority	Proportion
1	0.05
2	0.2
3	0.15
4	0.4
5	0.2

2 Jaamsim Model

To model the priorities, priority order, and maintenance we need to add some components to the model from the previous lab, so create a new folder called 'RC2' and copy your .cfg file (and the .png files so that the graphics work) from the previous lab folder into this folder and rename it to 'RC2.cfg'. First we need to add a priority attribute to the Patient entity. We can do this under the 'Options' tab on the PatientEntity using the AttributeDefinitionList. Table 2 shows how to create the priority attribute and make the default value 0.

Next we need a distribution to model the probabilities of the priorities. We use a DiscreteDistribution object as this allows us to define a list of values and the probability of each value occuring. Create a DiscreteDistribution object called PriorityDistribution with the values shown in table 3.

Table 2: Priority Attribute

Object	Keyword	Value
PatientEntity	AttributeDefinitionList	{ priority 0 }

Table 3: Priority Distribution

Object	Keyword	Value
PriorityDistribution	UnitType	DimensionlessUnit
	RandomSeed	4
	ValueList	1 2 3 4 5
	ProbabilityList	0.05 0.2 0.15 0.4 0.2

So far we have created the priority attribute and distribution, but we need to assign values from the distribution to the patient entities. With the HCCM objects we can assign attributes in the same object that create the arrival.

Table 4: Assign Priority

Object	Keyword	Value
PatientArrival	AssignmentList	{ 'this.obj.priority = [PriorityDistribution].Value' }

Now that we have the priority attribute we can use it change the path of the patients. We can use a Branch object (under Process Flow palette) to send the patients to different places based on the priorty attribute: those with priority 1 and 2 should go straight to the scan queue, while those with priorities 3, 4, and 5 go to wait for check in.

Table 5: Priority Branch

Object	Keyword	Value
PriorityBranch	NextComponentList	WaitForScan WaitForCheckIn
	Choice	'this.obj.priority $\leq$ 2 ? 1 : 2'

We also need to update the routing from the Arrival object so that the patients go from the Arrive to the Branch.

Table 6: Update Routing

Object	Keyword	Value
PatientArrival	NextAEJObject	PriorityBranch

Now we need to add the new Maintenance activity, and RequireMaintenance event. We need an additional event as well as the activity because we do not want to interrupt a scan with maintenance, if the machine is currently in use when the 8 hour time is reached. This means we cannot simply schedule another maintenance activity 8 hours after the last one was scheduled, as the machine might be in use at this time. Instead, we schedule an event (called a logic event in Jaamsim) in 8 hours time, the event then triggers some logic

which checks to see if the machine is free and can start maintenance, and if not changes an attribute so that it will start maintenance the next time it is free. We therefore also need to add an attribute on the CTMachineEntity called NeedMaintenance, which defaults to 0 and we will set to 1 when it has been 8 hours since the last maintenance.

For the maintenance activity create a process activity with a duration of 30 minutes with the CTMachineEntity as the only participant and the next activity is WaitForTaskCTMachine. Also use the start assignment list to set the value of the NeedMaintenance attribute to 0.

Then create a logic event called RequireMaintenance and for now just use the assignment list to set the NeedMaintenance attribute to 1.

Now we will add the new logic before connecting it with new triggers. Follow the same instructions as in the previous lab to create a new class called RC2ControlUnit in the labs package, and add the lines to the sim_custom.inc file so that it is available in the HCCM palette. Then replace the RC1ControlUnit with a RC2ControlUnit, also copy the java code from the RC1ControlUnit to the new RC2ControlUnit and replace the references to the RC1ControlUnit with RC2ControlUnit in the trigger objects: StartWaitCheckIn, StartWaitScan, StartWaitTaskReceptionist, and StartWaitTaskCTMachine.

In the new class we need to first update the OnStartWaitForTaskCTMachine, to include the logic for having maintenance and prioritising patients, and add two new ones for the logic triggered when a CTMachine arrives, and when the RequireMaintenance event occurs. Note that we don't need to update the OnStartWaitForScan logic as this will only start a scan if the patient is the only one waiting, so the priority does not matter.

First update the OnStartWaitTaskCTMachine code as follows, note that on line 4 a comparitor is created to compare patients by their priority attribute. This is then used on line 14 to sort the patients by priority. Also line 7 used the getNumAttribute function to access the value of the NeedMaintenance attribute of the CT Machine. You will need to fill in the parts labelled A, B, and C. In A the CT Machine should transition to the maintenance activity as it needs maintenance and has just become free. In B we want to save the priority of the highest priority patient that is waiting. In C we want to get the priority of the current patient in the loop, to see if it is the same as the highest priority waiting.

```
1  public void OnStartWaitForTaskCTMachine(List<ActiveEntity> ents, double simTime) {
2
3    ArrayList<ActiveEntity> waitPats = this.getEntitiesInActivity("PatientEntity", "WaitForScan", simTime);
4    AttributeCompare priorityComp = new AttributeCompare("priority");
5    ActivityStartCompare actSartComp = this.new ActivityStartCompare();
6    ActiveEntity ct = ents.get(0);
7    int reqMaintenance = (int) getNumAttribute(ct, "NeedMaintenance", simTime, -1);
8
9    if (reqMaintenance == 1) {
10     // A //
11   }
12
13   else if (waitPats.size() > 0) {
14     Collections.sort(waitPats, priorityComp);
15
16     int highestPriority = // B //
17
18     ArrayList<ActiveEntity> priorityPatients = new ArrayList<ActiveEntity>();
19     for (ActiveEntity wP : waitPats) {
20       int patPri = // C //
21       if (patPri == highestPriority) {
22         priorityPatients.add(wP);
23       }
```

```
    }

    Collections.sort(priorityPatients, actSartComp);
    ActiveEntity patient = priorityPatients.get(0);

    transitionTo("Scan", patient, ct);
  }
}
```

Then create two new methods called OnCTArrival and OnRequireMaintenance:

```
public void OnCTArrival(List<ActiveEntity> ents, double simTime) {

  double maintenanceTimeGap = 8 * 60 * 60;
  LogicEvent le = (LogicEvent) getSubmodelEntity("RequireMaintenance");

  le.scheduleEvent(ents, maintenanceTimeGap);
}

public void OnRequireMaintenance(List<ActiveEntity> ents, double simTime) {

  double maintenanceTimeGap = 8 * 60 * 60;
  LogicEvent le = (LogicEvent) getSubmodelEntity("RequireMaintenance");

  le.scheduleEvent(ents, simTime + maintenanceTimeGap);

  ActiveEntity ct = ents.get(0);
  if (ct.getCurrentActivity(simTime).equals("WaitForTaskCTMachine")) {
    transitionTo("Maintenance", ct);
  }
}
```

To get the new logic used in the simulation we need to add two new triggers: StartCTArrival, and StartRequireMaintenance. Set the control unit for both the triggers to be the RC2ControlUnit, and make the control policies the respective methods in the java code. Then update the TriggerList and TriggerChoice on both the CTMachineArrival and RequireMaintenance to refer to these triggers.

In the previous lab we looked at the mean time that patients spend in the clinic, and we were able to output this by first using a Statistics object to calculate it and then setting it in the Simulation object's RunOutputList. In this instance we are interested in the 95th percentile of time that patients spend in the clinic. Unfortunately the Statistics object does not provide the 95th percentile as an output. Therefore we need to capture each of the individual times that each patient spends in the clinic and calculate the 95th percentile ourselves. We can do this using an EntityLogger object from the Process Flow palette; create one and place it between the TimeInSystem object and the PatientExit, and name it PatientLogger. We then need to update the routing so that patients go through the PatientLogger object before leaving, and tell the PatientLogger object which values to record as shown in Table 7, note that 'TotalTime' is an output on the entity that stores the total time that the entity has been in the simulation for:

Table 7: Collecting Statistics

Object	Keyword	Value
TimeInSystem	NextComponent	PatientLogger
PatientLogger	DataSource	{ [Simulation].ReplicationNumber }
		{ 'this.obj.TotalTime / 1[h]' }
	NextComponent	PatientLeave

Now if you save and run your simulation you should be able to see patients arriving, checking in, being scanned, and leaving.

The simulation object should be configured correctly from the previous lab so we don't need to update it. Now if you save and run your simulation a file should be created called 'yourSimulationName.dat'.

With the model complete and the results recorded we can use R to analyse them. First download the R analysis file provided, then change the working directory and name of the .dat file to match yours, then run the R file. The following table should be printed:

lower95CI	mean	upper95CI
0.864	0.894	0.923

Task

By also saving the priority of the patients in the patient logger, construct 95% confidence intervals for the 90th percentile of the time spent in the clinic for each priority group. You should get the following output:

Priority	lower95CI	mean	upper95CI
1	0.469	0.481	0.493
2	0.495	0.502	0.509
3	0.636	0.650	0.664
4	0.862	0.887	0.911
5	1.62	1.76	1.89