Jaamsim Lab 3 – Using Traces and Scenarios

In this lab we will modify the simulation developed in the previous lab to run off of a pregenerated data trace that contains information about each patient. We will also explore how Jaamsim's built-in scenario indices can be used to run experiments where the values of the simulation's inputs are changed and use an EventLogger to log all events that an entity participates in. Finally we will package the simulation (Jaamsim and the custom Java code) as a .jar file, so that the simulation can be run easily from the command line on all major operating systems.

We are not considering any changes to the system, so the conceptual model is the same as for the previous lab.

1 Jaamsim Model

To run the simulation from a data trace we need to make some changes to the Jaamsim model. Once again create a new folder called 'RC3' 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 'RC3.cfg'. First, download the 'RC_50_week_data.txt' file from Canvas. This file contains 50 weeks of data of patients at the radiology clinic including: the time the patient arrived, the priority of the patient, the time the patient took to check in, and the time the patient took to have their scan.

Before we load the data in we will first change the starting date of the simulation, which defaults to 1970, to instead be 2023, so that the data read from the file is interpreted correctly. To do this go to the Simulation object and under the Options tab enter '2023-01-01' for the StartDate.

To use the data in Jaamsim we use a FileToMatrix object found in the Basic Objects palette. Create a FileToMatrixObject, rename it 'PatientData', and select the 'RC_50_week_data.txt' file as the DataFile.

We can now access the data in the file by using the 'Value' output of the PatientData object. The first place we will use this data is in the PatientArrival object, so that patients arrive according to the data in the file, rather than the distribution used previously. We first create two CustomOutputs (under the options tab) on the PatientArrival object to make accessing the data easier. CustomOutputs are similar to attributes but they can be expressions (formulas) and are re-calculated at each time step in the simulation. The two outputs we create will correspond to the data for the patient that has just arrived (thisPatientData) and the patient that is going to arrive next (nextPatientData). We need both of these so that we can calculate the appropriate interarrival time between the patients.

Once we have created these outputs we use them in the InterArrivalTime, and AssignmentList of the PatientArrival.

Note that in the AssignmentList we are assigning values from the data file to attributes on the patient entity for priority, check in time, and scan time. We will use these attributes later to determine how long those activities take (the priority attribute is already used in the PriorityBranch).

Table 1: Update PatientArrival

Object	Keyword	Value

PatientArrival	CustomOutputList	{ thisPatientData '[PatientData].Value(this.NumberAdded + 1)' }
		{ nextPatientData '[PatientData].Value(this.NumberAdded + 2)' }

	FirstArrivalTime	[PatientData].Value(2)(2)
	InterArrivalTime	'this.nextPatientData(2) - this.thisPatientData(2)'

	AssignmentList	{ 'this.obj.priority= this.thisPatientData(3)' }
		{ 'this.obj.checkInTime= this.thisPatientData(4)' }
		{ 'this.obj.scanTime= this.thisPatientData(5)' }

To avoid getting an error we need to add these attributes to the PatientEntity object. So update the AttributeDefinitionList of the PatientEntity to include checkInTime and scanTime as well as the current priority, all with a default of 0.

We now need to use the checkInTime and scanTime attributes to determine how long the check ins and scans take. We set the Duration of the CheckIn activity to 'this.obj.EntityList(1).checkInTime * 1[min]'. this.obj refers to the group of entities that have just started the activity (for check in this is a patient and a receptionist). The EntityList refers to the list of entities in this group, and we use the index 1 as the patient comes first, then we access the checkInTime attribute. We then need to multiply this by 1[min] to convert the number into a time, and use minutes as the attribute is in minutes.

Similarly for the Scan activity we set the Duration to 'this.obj.EntityList(1).scanTime * 1[h]', note that here we use 1[h] as the attribute is in hours.

Now, suppose we are interested in the time that patients spend waiting for check in and for the scan. We can't use the current PatientLogger as it only records the total time that patients are in the system for. We could add attributes for each time that we are interested in, and assign the value when the entity gets to the relevant stage, but we can instead use an EventLogger from the HCCM palette. The EventLogger records the time that an entity starts each of the activities that it participates in. So, create an EventLogger and call it PatientEventLogger. Then to get the events recorded go to the PatientLeave object and under the HCCM tab enter PatientEventLogger for the EventLogger keyword. Now any entities that are sent to the patient leave will have the start times of any activities that they participated in recorded.

We will now configure the Simulation object to run one long replication for several scenarios. Under the Key Inputs tab enter '50w' for the RunDuration, this will make the simulation run for 50 weeks. We have to run one 50 week replication rather than 50 one week replications as Jaamsim cannot read in a new file when each replication starts.

We want to try out four scenarios with either three or four CT machines, and either one or two receptionists. As there are two factors we are changing we use a ScenarioIndex with two numbers, the first indexes the scenarios relating to the number of CT Machines, and the second those related to the number of receptionists.

Since there are two options for the first index and two for the second we enter '2 2' for the ScenarioIndexDefinitionList under the MultipleRuns tab. We will start from scenario 1 in both the indices and end at 2 for the first one and 2 for the second so StartingScenarioNumber is '1-1' and EndingScenarioNumber is '2-2'. We are going to run just one long replication for each scenario so set the NumberOfReplication to 1.

Now Jaamsim will run 4 scenarios, but we need to make it so that the number of CT Machines and Receptionists actually changes in each of the scenarios. For the CT Machines we set the MaxNumber on the CTMachineArrival to '[Simulation].ScenarioIndex(1) + 2', which gets the value of the first scenario index and adds 2 to it. For the Receptionists we can set the MaxNumber on the ReceptionistArrival to [Simulation].ScenarioIndex(2), in this case we don't need to add one as the scenario index is the same as the number of receptionists we want to use.

Download and run the RC3_Analysis.R file, you will have to update the directory that it reads the data from and the name of the data file used. The script splits each replication into 50 batches, each one week long, and calculates the mean across the batches and the four scenarios of the 90th percentile waiting time for both check in and scan within each of batch. No warm-up period is used, so this assumes that being empty and idle is a typical state of the system. Splitting into batches by week assumes that each week is not correlated to the preceding and following weeks. You should get the following output:

CTMachineScenario	ReceptionistScenario	CheckInWait	ScanWait
1	1	0.0594	0.536
1	2	0	0.548
2	1	0.0594	0.155
2	2	0	0.170

2 Creating an Executable JAR File

We can package the custom Java code in the extended control unit class alongside the base Jaamsim and HCCM code into a jar file that can be run without having to set up an Eclipse project. To do this right click on the 'sim' Java project in Eclipse and select Export.

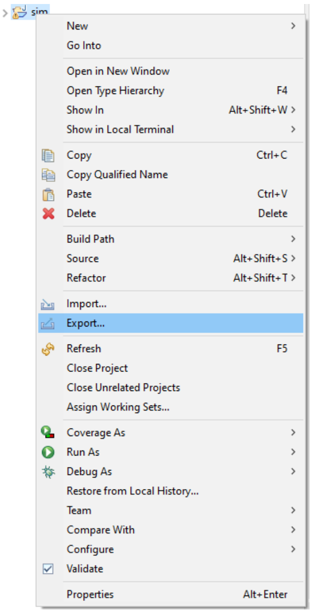
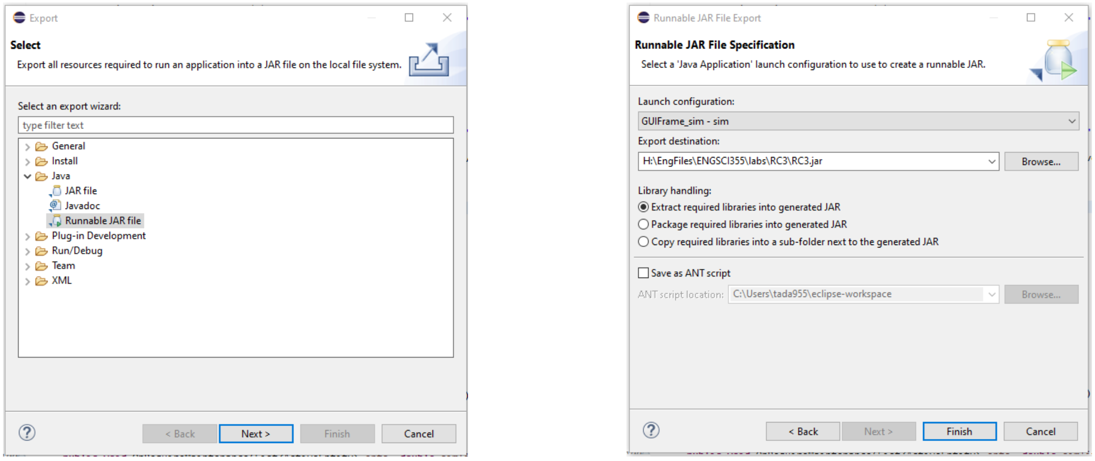


Figure 1: Screenshot of Exporting Project

In the subsequent menu select ‘Runnable JAR file’ and click Next. In the next window choose GUIFrame as the launch configuration. For the Export destination choose the folder that this lab is saved in, call it something like ‘RC3.jar’, and select ‘Extract required libraries into generated JAR’, and click ‘Finish’.



Export

Select

Export all resources required to run an application into a JAR file on the local file system.

Select an export wizard:

type filter text

Runnable JAR File Export

Runnable JAR File Specification

Select a 'Java Application' launch configuration to use to create a runnable JAR.

Launch configuration: GUIFrame_sim - sim

Export destination: H:\EngFiles\ENGSCI355\labs\RC3\RC3.jar

Library handling:

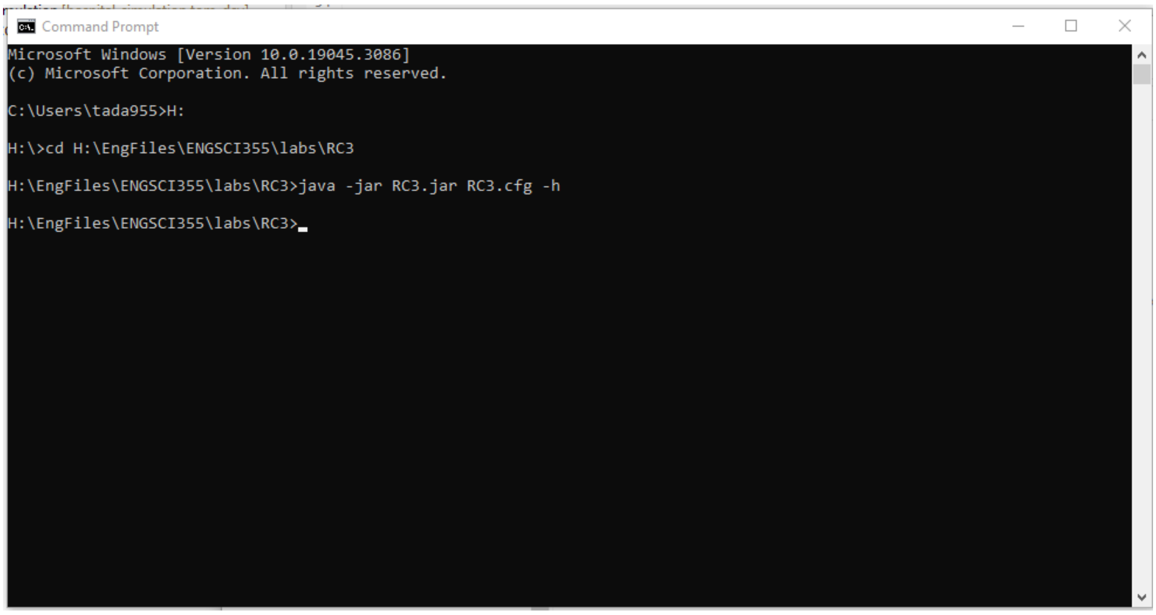
● Extract required libraries into generated JAR
○ Package required libraries into generated JAR
○ Copy required libraries into a sub-folder next to the generated JAR

□ Save as ANT script

ANT script location: C:\Users\tada955\eclipse-workspace

Figure 2: Screenshot of Exporting Project 2

This will create the named .jar file in the given directory, which can be used to run Jaamsim models from the command line. Don’t worry if you get a message saying there were errors or warnings when creating the .jar file, as long as the .jar file is created it should work. Open a command prompt, navigate to the folder with the .jar and .cfg files and run the command ‘java -jar RC3.jar RC3.cfg -h’, the -h option stands for ‘headerless’ and means that Jaamsim will be run without the GUI so that it runs faster. It may take a few minutes to run and you will not be able to see any progress, but if you look in the folder that the .cfg file is in you should see the output files being updated.



Command Prompt

```
Microsoft Windows [Version 10.0.19045.3086]
(c) Microsoft Corporation. All rights reserved.

C:\Users\tada955>H:

H:\>cd H:\EngFiles\ENGSCI355\labs\RC3

H:\EngFiles\ENGSCI355\labs\RC3>java -jar RC3.jar RC3.cfg -h

H:\EngFiles\ENGSCI355\labs\RC3>_
```

Figure 3: Screenshot of Command Prompt

To get the lab signed off, show a tutor that you can run the .jar file from the command line.