ENGSCI 355 Labs

Thomas Adams

2024-08-29

Table of contents

Preface

These are an online version of the Labs for ENGSCI 355. The topics covered are: a hands-on simulation of a manufacturing process; conceptual modelling using HCCM; implementing HCCM models in Jaamsim; and missing data imputaton.

Part I

Jaamsim Labs

1 Setting Up VSCode and Java

In this lab you will walk through the set up of running a Java program in VSCode. You will need to be able to do this to implement HCCMs in Jaamsim.

If you do not already have VSCode installed on your machine, download and install the version appropriate for your operating system from here.

If you are using your own laptop, it is best to install a recent version of the Java JDK (unless you are confident you already have a recent version). We recommend Amazon Corretto 21. You can download it from here, and then install it.

To see short videos of the steps described below please visit this site.

Open VSCode, then on the left-hand side click on the Extensions tab. Search for the 'Extension Pack for Java' and install it.

Create a new folder called **ENGSCI355**, if you are using a lab computer create this folder on your H drive. If you aren't, then create it within Documents or wherever you usually keep University related work. Inside the **ENGSCI355** folder create another folder called **Java_Example**. Then in VSCode open this folder by going File -> Open Folder, then navigating to the **Java_Example** folder.

Once you have opened the folder in VSCode, create a new file in it called **Hello.java**. Once you have that file (or any .java file) open VSCode should detect that you are editing a java file and, if there isn't one already, create a Java Project in the same folder. You should see that the 'Java Projects' section has been enabled in the bottom left of the screen.

We now want to make sure that this Java Project is using the correct Java JDK. Hover your mouse over the 'Java Projects' title and then click on the three dots that appear on the right hand side (with the tooltip 'More Actions'), and select 'Configure Java Runtime'. Use the drop-down menu that appears to select the Amazon Corretto 21 JDK. If it is not on the list, select 'Find a local JDK' and browse to the location that you installed the Amazon Corretto JDK.

Now go back to the Hello.java file and add the following code:

```
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello!");
  }
}
```

To run the program either click on the 'Run' button just above the line that declares the main function, or click on the run button in the top right corner of the screen. You should see a line with 'Hello!' printed by itself.

You have now create and run a Java program! If you run into any issues, there are more detailed instructions available here.

2 Setting Up JaamSim and HCCM

In this lab you will walk through the set up of how to run JaamSim from source in a VSCode project and how to include the HCCM library in JaamSim.

2.1 Prerequisites

These instructions were prepared using:

1. Git – 2.47.0.2;
2. GitHub Desktop – 3.4.8 (x64);
3. Java – Amazon Corretto JRE 21;
4. VSCode – 1.95.1.

They should work with more recent versions of the software too. All of this software is standard on Engineering lab machines. Amazon Corretto JRE 21 is available on the University of Auckland's Software Centre.

To see short videos of the steps described below please visit this site.

2.2 Create the Project Folder Structure

Create a new folder on your H drive called ENGSCI355. Then create three folders within this one, called **sim**, **labs**, and **workspace**. The **sim** folder will contain the java code for the simulation software Jaamsim, including custom code that you write, and is the focus of these instructions. The **labs** folder will contain subfolders for each lab with the simulation files for each. Create a folder for HCCM logic functions within the **sim** folder. We will use **sim_custom** in these instructions.

2.3 Clone HCCM into the project folder

Open GitHub Desktop and go to File → Clone repository, then select the URL tab and enter

https://github.com/mosu001/hccm

as the URL. Choose the Local path to be the **sim** folder that you just created.

into an hccm folder within the **sim** folder. This will create an **hccm** folder within the **sim** folder that contains the HCCM and Jaamsim code.

Note, if you use git from the command line, e.g., Git Bash, you need to add the recurse submodules option

```
git clone --recurse-submodules https://github.com/mosu001/hccm
```

2.4 Create files to load HCCM and customised components

From **hccm_custom** copy both **autoload.cfg** and **hccm.inc** into the **sim_custom** folder. Then open **autoload.cfg** with VSCode and edit it so that the content matches that in Figure 2.1.

```
Include units.inc
Include sim.inc
Include units-imperial.inc
Include units-knots.inc
Include displayModels.inc
Include graphics.inc
Include probabilityDistributions.inc
Include basicObjects.inc
Include resourceObjects.inc
Include examples.inc
Include processFlow.inc
Include calculationObjects.inc
Include fluidObjects.inc
Include submodels.inc
Include hccm.inc
Include sim_custom.inc
```

Figure 2.1: Customised autoload.cfg File

Then rename **hccm.inc** to **sim_custom.inc**, open it in VSCode, and delete all the contents so it is blank. Don't forget to save both **autoload.cfg** and **sim_custom.inc**.

2.5 Create a VSCode Java Project

In VSCode use File → Open Folder to open the **sim** folder. In the File Explorer open some folder so that you can see a .java file and open it, for example: hccm\custom\hccm\Constants.java. VSCode should then recognise that you have opened a Java file and the Java Projects pane should appear.

2.6 Configure Source Folders

Now we need to tell VSCode where the source code of the project is. To do this we click on the three dots at the right of the 'Java Projects' title and select 'Configure Classpath'. A new menu should come up that allows you to add and remove sources. If anything other than **hccm\custom** is already there remove it by clicking on the x on the far right hand side, then 'Apply Settings'. Add new sources by clicking on 'Add Source Root'. First add **sim\hccm\jaamsim\src\main\java**, then click 'Apply Settings'. Then add both **sim\hccm\jaamsim\src\main\resources** and **sim\sim_custom**, remembering to apply the settings after each one.

You can check to make sure that you have the correct sources configured by opening the **settings.json** file in the **.vscode** folder. Under "java.project.sourcePaths" there should be the following four entries:

- hccm\\custom
- hccm\\jaamsim\\src\\main\\java
- hccm\\jaamsim\\src\\main\\resources
- sim_custom

2.7 Configure JDK

We need to make sure that VSCode is using the version of Java that we want it to. To do this we click on the three dots at the right of the 'Java Projects' title and select 'Configure Java Runtime'. A drop-down menu for JDK should come up. Make sure that JavaSE-21 is selected and then click 'Apply Settings'.

2.8 Configure Libraries

JaamSim also needs the gluegen and jogl libraries to run. These are packaged with JaamSim as .jar files (a compiled Java program). They can be added by opening the project settings by clicking on the three dots at the right of the 'Java Projects' title and selecting either 'Configure Java Runtime' or 'Configure Classpath'. Then select the 'Libraries' tab on the right. Click on 'Add Library', then navigate to hccm\jaamsim\jar, select all of the files, and click 'Select Jar File'. Then click 'Apply Settings'.

2.9 Integrate with JaamSim

To integrate HCCM and any custom logic with JaamSim you need to copy your **autoload.cfg** and **sim_custom.inc** files (from **sim_custom**) to sim\hccm\jaamsim\src\main\resources\resources\inputs and replace the autoload.cfg file that is currently there. You also need to copy the file **hccm.inc** in hccm\custom to the same location. To check that they have been copied correctly you can look in the 'Java Projects' section on the left-hand side. Under hccm\jaamsim\src\main\resources\resources\inputs you should see both **hccm.inc** and **sim_custom.inc**. If you don't, try using the menu accessed by clicking the three dots and selecting 'Refresh'.

2.10 Run Custom JaamSim

You should now be able to run JaamSim with the HCCM objects enabled. Start by clicking on the 'Run and Debug' menu on the left-hand side, then click on 'create a launch.json file', and select 'Java' from the list of debuggers that comes up in the middle of the screen. By doing this VSCode analyses the source code to determine which java files you might like to run and creates run configurations for each of them. In the file that is created you should see an entry with the name 'GUIFrame', this is the class that we need to run to start JaamSim. To make the view work correctly when JaamSim is running you need to add another parameter called "vmArgs" with the following entries enclosed in double quotes and separated by spaces on a single line:

- --add-exports java.base/java.lang=ALL-UNNAMED
- --add-exports java.desktop/sun.awt=ALL-UNNAMED
- --add-exports java.desktop/sun.java2d=ALL-UNNAMED

The final entry in the .launch file should look like this:

```
"type": "java",
"name": "GUIFrame",
"request": "launch",
"mainClass": "com.jaamsim.ui.GUIFrame",
"projectName": "sim_d11998cc",
"vmArgs": "--add-exports java.base/java.lang=ALL-UNNAMED
↪  --add-exports java.desktop/sun.awt=ALL-UNNAMED
↪  --add-exports java.desktop/sun.java2d=ALL-UNNAMED"
```

Then, in the top left-hand corner next to the green play button click on the drop-down menu and select 'GUIFrame'. Then click the green play button to run JaamSim. The launch screen should appear but you might also have to click on the JaamSim icon in the Taskbar at the bottom of the screen to open JaamSim. You should see the 'HCCM' palette at the bottom of the 'Model Builder' window, and be able to drag and drop objects into the View.

2.11 Running an HCCM Model

Now that we have JaamSim running with the HCCM objects we can try running an existing model. Download the single server queue model's folder from Canvas (ssq.zip), move it into the **labs** folder and extract **ssq.zip** into that folder. You might want to remove the ssq at the end of the extraction destination to prevent nested ssq folders being created.

Now we need to create package in our Java Project to hold the custom logic associated with this model. In VSCode right-click on the **sim_custom** folder and select New Java Package. Enter **ssq** for the name of the package and click Finish. This will have created a new folder in the **sim_custom** folder called ssq.

Now go back to the ssq folder you extracted the zip file to and copy the FIFOQControlUnit.java file to the newly created package folder under sim\sim_custom\ssq. This java file defines a new Jaamsim object, in this case the control unit for the SSQ model.

Finally we need to make this new object available in Jaamsim. To do this we need to edit the **sim_custom.inc** file that we put in sim\hccm\jaamsim\src\main\resources\resources\inputs. Open the **sim_custom.inc** file and also open the **ssq.inc** file in the ssq folder. Copy the contents of **ssq.inc** into **sim_custom.inc.**

Run JaamSim with HCCM from the Run and Debug menu again (make sure that GUIFrame is selected in the drop-down). You should now see a Single Server Queue palette in the Model Builder window. It has the FIFO trigger for the Single Server Queue model (you will learn more about triggers in later labs).

Next in Jaamsim in the top left corner select file, then open, and open ssq.cfg from the ssq folder. You can run the model by clicking on the blue play button in the top left and see how the customers and servers join together for service in the queue.

Part II

Conceptual Models

3 Radiology Clinic

3.1 Data

Table 3.1: List of Global Variables

Name	Description	Initial Value
NextPatIdNum	The Id number that will be assigned to the next patient	1
NextReceptionistIdNum	The Id number that will be assigned to the next receptionist	1
NextCTMachineIdNum	The Id number that will be assigned to the next CT Machine	1
P	The set of all patients	$\emptyset$
R	The set of all receptionists	$\emptyset$
C	The set of all CT Machines	$\emptyset$

Table 3.2: List of Data Modules

Name	Source	Identification	Input	Output
PatientInterarrivalTime	Poisson Process	Parameter	Mean interarrival time	Sample from Distribution
NumReceptionists	Constant	Parameter	N/A	Value
NumCTMachines	Constant	Parameter	N/A	Value
CheckInTime	Uniform Distribution	Parameter	Min and max time	Sample from Distribution
ScanTime	Log-normal Distribution	Parameter	Mean and std. dev.	Sample from Distribution

3.2 Components

Table 3.3: List of Entities

Entity	Attributes
Patient	ID State StateTimes
Receptionist	ID State StateTimes
CT Machine	ID State StateTimes

Table 3.4: List of Transitions

No.	Participant	From Event	To Event
1	Patient	Arrive(P)	Wait for check in.Start
2	Patient	Wait for check in.End	Check in.Start
3	Patient	Check in.End	Wait for scan.Start
4	Patient	Wait for scan.End	Scan.Start
5	Patient	Scan.End	Leave(P)
6	Receptionist	Arrive(R)	Wait for task(R).Start
7	Receptionist	Wait for task(R).End	Check in.Start
8	Receptionist	Check in.End	Wait for task(R).Start
9	Receptionist	Wait for task(R).End	Leave(R)
10	CT Machine	Arrive(CT)	Wait for task(CT).Start
11	CT Machine	Wait for task(CT).End	Scan.Start
12	CT Machine	Scan.End	Wait for task(CT).Start
13	CT Machine	Wait for task(CT).End	Leave(CT)

Table 3.5: Activities

Activity	Participants	Event	Type	State Change
Wait for Check In	Patient (p)	Start	Scheduled	1 **TRIGGER** OnStartWaitForCheckIn **WITH** p
		End	Controlled	
Check In	Patient (p), Receptionist (r)	Start	Controlled	1 **SCHEDULE** Check In.End at **TIME** + CheckInTime()
		End	Scheduled	1 **START** Wait **for** Scan **WITH** p 2 **START** Wait **for** Task (R) **WITH** r
Wait for Scan	Patient (p)	Start	Scheduled	
		End	Controlled	1 **TRIGGER** OnStartWaitForScan **WITH** p
Scan	Patient (p), CTMachine (c)	Start	Controlled	1 **SCHEDULE** Scan.End at **TIME** + ScanTime()
		End	Scheduled	1 **START** Leave (P) **WITH** p 2 **START** Wait **for** Task (CT) **WITH** c
Wait for Task (R)	Receptionist (r)	Start	Scheduled	1 **TRIGGER** OnStartWaitForTaskR **WITH** r
		End	Controlled	
Wait for Task (CT)	CTMachine (c)	Start	Scheduled	1 **TRIGGER** OnStartWaitForTaskCT **WITH** c
		End	Controlled	

Table 3.6: Events

Activity	Participants	Type	State Change

Simulation Start	-	Scheduled	`SCHEDULE Arrival (R) at TIME`
			`SCHEDULE Arrival (CT) at TIME`
			`SCHEDULE Arrival (P) at TIME + PatientInterArrival()`

Arrival (P)	Patient (p)	Scheduled	`p.ID = NextPatIDNum`
			`p.Priority = PatientPriority()`
			`NextPatIDNum = NextPatIDNum + 1`
			`SCHEDULE Arrival (P) at TIME + PatientInterArrival()`
			`START Wait for Check In WITH p`

Leave (P)	Patient (p)	Scheduled	`Calculate statistics for p`

Arrival (R)	Receptionist (r)	Scheduled	`r.ID = NextReceptionistIDNum`
			`NextReceptionistIDNum = NextReceptionistIDNum + 1`
			`IF NextReceptionistIDNum <= NumReceptionists THEN`
			`  SCHEDULE Arrival (R) at TIME`
			`END IF`
			`START Wait for Task (R) WITH r`

Leave (R)	Receptionist (r)	Scheduled	`Calculate statistics for r`

Arrival (CT)	CTMachine (c)	Scheduled	`c.ID = NextCTMachineIDNum`
			`NextCTMachineIDNum = NextCTMachineIDNum + 1`
			`IF NextCTMachineIDNum <= NumCTMachines THEN`
			`  SCHEDULE Arrival (CT) at TIME`
			`END IF`
			`START Wait for Task (CT) WITH c`

Leave (P)	CTMachine (c)	Scheduled	`Calculate statistics for c`

3.3 Activity Diagrams

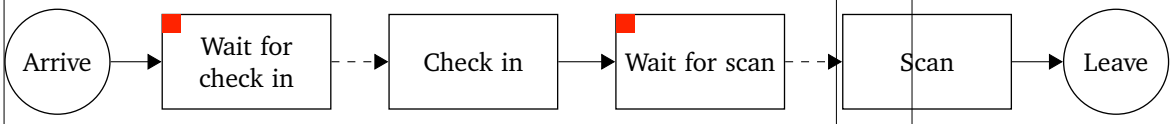

Figure 3.1: Patient Activity Diagram

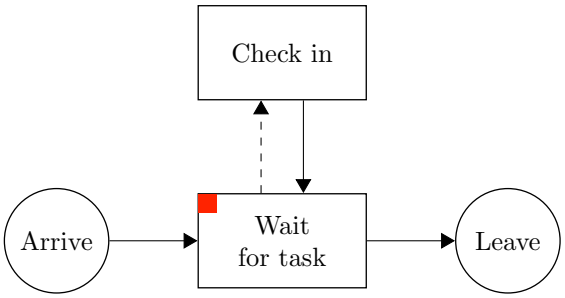

Figure 3.2: Receptionist Activity Diagram

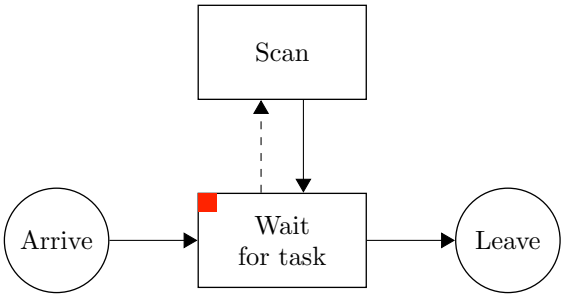

Figure 3.3: CT Activity Diagram

3.4 Control Policies

Table 3.7: OnStartWaitForCheckIn

Triggered by: Patient p

```
1  receps = {r FOR r IN R IF r.State = "Wait for task (R)"}
2  IF receps IS NOT empty THEN
3      r_hat = argmin{r.CurrentStart FOR r IN receps}
4      START Check In WITH p, r_hat
5  END IF
```

Table 3.8: OnStartWaitForScan

Triggered by: Patient p

```
1  cts = {c FOR c IN C IF c.State = "Wait for task (C)"}
2  IF cts IS NOT empty THEN
3      c_hat = argmin{c.CurrentStart FOR c IN cts}
4      START Scan WITH p, r_hat
5  END IF
```

Table 3.9: OnStartWaitForTaskR

Triggered by: Receptionist r

```
1  patients = {p FOR p IN P IF p.State = "Wait for Check In"}
2  IF patients IS NOT empty THEN
3      p_hat = argmin{p.CurrentStart FOR p IN patients}
4      START Check In WITH p, r_hat
5  END IF
```

Table 3.10: OnStartWaitForTaskCT

Triggered by: CTMachine c

```
patients = {p FOR p IN P IF p.State = "Wait for Scan"}
IF patients IS NOT empty THEN
    p_hat = argmin{p.CurrentStart FOR p IN patients}
    START Scan WITH p, r_hat
END IF
```