

ENGSCI 355 Labs

Thomas Adams

2024-08-29

Table of contents

Preface	3
I Practical Lab	4
1 Operations System in Practice	5
1.1 Making Paper Cars	5
1.2 Reflections	6
II Conceptual Modelling Labs	7
2 Inputs, Outputs, Entities, and Behaviour	8
2.1 Understanding of the Problem Situation	8
2.2 Identification of Modelling and General Objectives	8
2.3 Defining Output Responses	9
2.4 Defining Input Factors	9
2.5 Model Content	9
2.5.1 Identifying Entities	9
2.5.2 Drawing Behavioural Paths	9
3 Events, Activities, and Logic	11
III Jaamsim Labs	12
4 Setting Up VSCode and Java	13
5 Setting Up JaamSim and HCCM	14
5.1 Prerequisites	14
5.2 Create the Project Folder Structure	14
5.3 Clone HCCM into the project folder	14
5.4 Create files to load HCCM and customised components	15
5.5 Create a VSCode Java Project	15
5.6 Configure Source Folders	15
5.7 Configure JDK	16
5.8 Configure Libraries	16
5.9 Integrate with JaamSim	16
5.10 Run Custom JaamSim	16
5.11 Running an HCCM Model	17
6 Radiology Clinic	18
6.1 Experiments	18
6.2 Jaamsim Model	18
6.2.1 Creating Model Objects	18
6.2.2 Configuring Objects	20
6.3 Model Logic – Java	23
6.4 Model Output	28
6.5 Task	29
7 Extended Radiology Clinic	30
7.1 Experiments	30

7.2	Jaamsim Model	30
7.3	Task	35

IV Conceptual Models 36

8	Making Paper Cars Conceptual Model	37
8.1	Understanding of the Problem Situation	37
8.2	Modelling Objectives	37
8.3	General Objectives	38
8.4	Defining Output Responses	38
8.5	Defining Input Factors	39
8.6	Identifying Entities	39
8.7	Drawing Behavioural Paths	40
9	Radiology Clinic	45
9.1	Data	45
9.2	Components	46
9.3	Activity Diagrams	47
9.4	Control Policies	49

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 imputation.

Part I

Practical Lab

1 Operations System in Practice

The goal of this lab is to give you some hands-on experience with an operations system, the type of system that we will be focussing on simulating. Hopefully this will give you some idea of what is needed to simulate a system in terms of:

- the components of the system and how they interact with each other (entities and their behaviour);
- the type and amount of information/data that is needed, both for activity durations and control policies;
- the types of experiments that can be performed and how the system can be redesigned.

1.1 Making Paper Cars

The system that we will use as an example is making a car out of paper. You will each be given a piece of paper with the net of paper car on it as in Figure 1.1.

You will also get a pair of scissors, some tape, and blank pieces of paper. To make the car:

1. Trace the net onto a new piece of paper.
2. Cut the new net out.
3. Fold the paper and tape the edges shut placing the tabs on the inside.

Figure 1.2 shows an example of a completed car.

First everyone should make one car by themselves. Once you have, show one of the instructors to get signed off. Then, discuss with you group how you can work together to make paper cars. You might want to experiment with different setups/policies and try making a few cars to see how long it takes and gather some data.

There will be a competition to see which group can make the most cars in 10 minutes. Before the time starts each group must submit an estimate of how many cars they believe they will be able to make. The score for each group will then be comprised of the following elements:

- 1 point for each car completed up to and including the estimated number.
- 0.25 points for each car completed above the estimated number.
- -0.75 points for each car not completed in the estimated number.

Additionally, the following rules must be followed:

1. Each car must be traced and cut individually.
2. Cars must be the same shape as the original template, including tabs.
3. You can have as many stencils as you like.
4. All final cars must have started as a blank, unfolded piece of paper.
5. You may not have any pre-cut tape or nets.
6. All cars must have been made only by members of your group.
7. All cars must be folded and taped neatly to count. The lecturer has final say on whether a car meets the required neatness.

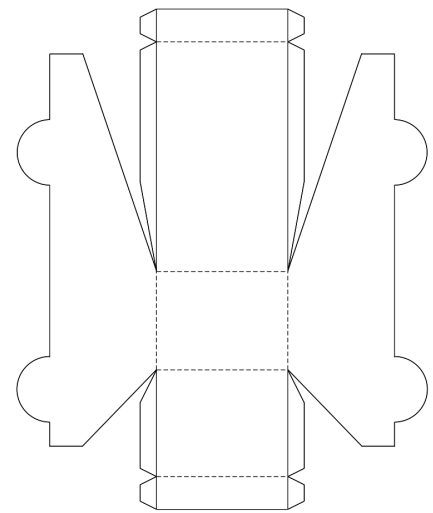


Figure 1.1: The Net Used to Make Paper Cars

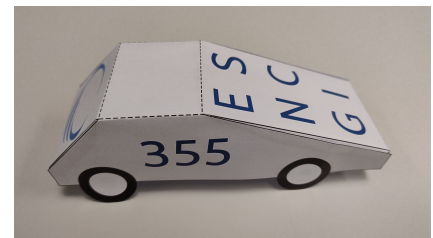


Figure 1.2: A Completed Car

1.2 Reflections

Now that you have attempted to make as many cars as you can you may wish to reflect on the process by asking yourself the following questions:

- Did your group have any traced/cut out cars left at the end?
- What was the bottleneck/slowest part of the system?
- Did you collect any data/do any experiments? If so, did they help? Would you do more/different ones now?
- What would you do differently next time?

The process that we considered was relatively simple. Consider how would your group's strategy change if any of the following additional conditions were added:

- Blank pieces of paper for you to trace onto only become available one at a time every two minutes;
- You have to make different styles of cars on demand;
- There is a limit to how many traced nets/cut pieces of tape you can have at any point (buffer limit);
- Each time a pair of scissors is stopped being used there is a cooldown time of 1 minute.

Part II

Conceptual Modelling Labs

2 Inputs, Outputs, Entities, and Behaviour

In this lab and the next one, we will work through the Hierarchical Control Conceptual Modelling (HCCM) framework to build a conceptual model, aligned with the HCCM standard from lectures, that represents the practical activity, i.e., making paper cars, from Chapter 1.

Working in the same groups as for the practical activity, you will work through the phases for HCCM modelling shown below and complete templates for those steps. In this lab you will complete phases 1, 2, 3, and start phase 4. The remainder of phase 4 will be completed in the next lab. Chapter 1 provides a partially completed conceptual model of the car making system that you can use as a starting point.

2.1 Understanding of the Problem Situation

To understand the problem situation, you need to summarise what is happening in a concise way. There is no strict rule for the best way to do this. One good approach is listening to the problem “holder”, i.e., person/people who have the problem such as a client, then reflecting what you have heard in a couple of paragraphs with lists of key details and questions. You can then work through one or more iterations of feedback and refinement to get a final, agreed upon problem description.

2.2 Identification of Modelling and General Objectives

As described in lectures, there are two types of objectives to consider when developing a simulation:

“The second step deals with the determination of the objectives. According to Robinson [26] they drive all aspects of the modeling process and are a subset of an organization’s aims. Further, objectives can be classified into modeling and general objectives, where the latter are concerned with the flexibility, run-speed, visual-display and model/component reuse.”

For the modelling objective you may like to think about what you trying to discover using simulation, and what level of performance you are trying to achieve in which areas/metrics.

2.3 Defining Output Responses

Output responses are things that can be measured and compared to understand how a system has behaved/performed. They are the metrics used to compare different simulation scenarios. The output responses should let you know whether the modelling objectives have been achieved and why or how. You may also want to consider how this will be reported (tables, graphs, etc.).

2.4 Defining Input Factors

Input factors are things that can be changed and may modify how a system behaves/performs. They are often defined to create multiple different scenarios to compare via simulation. They are also what you can change to try and achieve the modelling objectives.

2.5 Model Content

For the model content definition of our conceptual model we will follow the new HCCM standard. This standard is presented in an academic article (currently under review) that is available on Canvas under Files > Lectures > Conceptual Modelling in the file [hccm-standard.pdf](#)

2.5.1 Identifying Entities

Before formally defining entities it is often useful to identify entities in the system and whether they are active, i.e., have behaviour like a doctor or patient, or passive, i.e., are part of the system that should be modelled but that don't have explicit behaviour like a waiting room with a given capacity, but that doesn't actually have defined actions.

The goal is to identify everything that is involved in a meaningful way in all of the activities that are important to the system. Thinking about the inputs and outputs can also be useful. Clearly the entities must be influenced in some way by the inputs, and they must themselves influence the outputs. You may also consider that an activity does not have a significant influence on the performance of the system, and decide to exclude it – and therefore any entities that are involved only in that activity. Likewise the participation of a particular entity in an activity might be deemed inconsequential and therefore excluded.

2.5.2 Drawing Behavioural Paths

Once preliminary identification of identities has been done, behavioural paths for each of the active entities should be drawn. These are essentially flowcharts with a special structure. Circles represent events, usually used when entities are arriving and leaving. Rectangles represent activities, including when entities have to wait for another activity. Red squares at the top left of an activity (or sometimes an event) let us know that some logic is triggered when the activity starts. This generally occurs at the start of “wait” activities and is used to check whether the conditions that mean the entity can stop waiting and move on to the next activity are met.

What we are trying to do when drawing the behavioural paths is identify the activities and events that the entities participate in, the possible orders that these can occur in, and any points where some control logic needs to be used.

Both when identifying the entities and drawing the behavioural paths it is important to keep track of any assumptions and simplifications that you make.

3 Events, Activities, and Logic

In summary, this chapter has no content whatsoever.

Part III

Jaamsim Labs

4 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:

```
1 public class Hello {  
2     public static void main(String[] args) {  
3         System.out.println("Hello!");  
4     }  
5 }
```

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](#).

5 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.

5.1 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](#).

5.2 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.

5.3 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
```

5.4 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 5.1.

```
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 5.1: 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**.

5.5 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.

5.6 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

5.7 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'.

5.8 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'.

5.9 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'.

5.10 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:

```
1  "type": "java",
2  "name": "GUIFrame",
3  "request": "launch",
4  "mainClass": "com.jaamsim.ui.GUIFrame",
5  "projectName": "sim_d11998cc",
6  "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.

5.11 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.

6 Radiology Clinic

In this lab you will be introduced to the basics of creating a simulation model using the discrete event simulation software Jaamsim and the HCCM module. To do this we will use the CT service of a radiology clinic as an example. At the clinic patients: arrive according to a known distribution 24/7; check in at reception, which takes a uniformly distributed amount of time; and then have a scan, the duration of which also follows a known distribution; and finally leave.

We want to use the simulation to determine the average time that patients spend in the clinic, between arriving and leaving. We want to compare this time to the time that patients would spend in the system if interarrival times and scan durations were always equal to the average of the distributions for all patients. Typically we would first formulate the simulation model by defining the objectives, benefits, conceptual model, and experiments. For the sake of brevity we will only cover the experiments. As the aim of the lab is to learn the basics of Jaamsim, the conceptual model is not given here, instead it is available in Chapter 9.

6.1 Experiments

We will perform just one experiment, using distributions for the arrival, check in, and scan processes. We will use a Poisson distribution with $\lambda = 8/\text{hour}$ for the arrival process, a uniform distribution between 2 and 5 minutes for the check in durations, and a log-normal distribution where the underlying normal variable has a mean of -1.34 and standard deviation 0.29 for the scan durations. For the experiment we will run 50 replications that each last for 1 week.

6.2 Jaamsim Model

6.2.1 Creating Model Objects

Run Jaamsim by opening your VSCode project and the clicking the run button and select GUIFrame. The HCCM palette on the left hand side allows us to create Jaamsim objects that correspond to the components of our HCCM conceptual models. Based on the problem description and conceptual model we need three types of entities: patients, receptionists, and CT Machines. To create each of these expand the HCCM palette in the Model Builder window, select **ActiveEntity**, and dragging it into the View Window, see Figure 6.1. Then in the Object Selector window select **ActiveEntity1**, press F2, and rename it **PatientEntity**.

Repeat this process two more times and create ActiveEntities called **ReceptionistEntity** and **CTMachineEntity**.

An ActiveEntity object by itself does not create any entities in the simulation, it just acts as a prototype for entities. To create entities an ArriveEvent object is used, which simulates patients/receptionists/CTMachines arriving

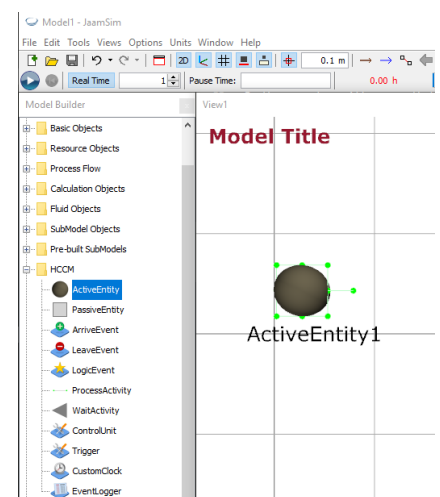


Figure 6.1: Screenshot of an ActiveEntity

Table 6.1: Arrival Distribution Inputs

Object	Keyword	Value
ArrivalDistribution	UnitType	TimeUnit
	RandomSeed	1
	Mean	0.125 h

at the clinic. The ArriveEvent object creates a series of entities that are passed to the next object in a process. The PrototypeEntity keyword identifies the entity to be copied. The rate at which entities are generated is determined by the InterArrivalTime and FirstArrivalTime keywords. Create three ArriveEvents called **PatientArrival**, **ReceptionistArrival**, and **CTMachineArrival**, and set the PrototypeEntity to be the related entity (patient, receptionist, CTMachine).

We also need to create objects that represent the entities leaving, called LeaveEvent, we will only create one for the patients, as we are assuming that the receptionist and CT machines are available 24/7 so they do not need to leave. Drag and drop a leave event into the simulation, rename it **PatientLeave**, and set the Participant to be the patient entity (under the HCCM tab).

The patients waiting for check in and scanning, and both the receptionist and CT machines waiting for tasks can be represented by WaitActivities, so create four WaitActivities and rename them **WaitForCheckIn**, **WaitForScan**, **WaitForTaskReceptionist**, and **WaitForTaskCTMachine** respectively, and once again set the Participant to the respective entity.

We can then represent the patient doing check in with the receptionist, and the patient being scanned by a CT machine as process activities. Create two process activities and rename them **CheckIn**, and **Scan**.

We also need to create objects to represent the probability distributions that the interarrival, check in, and scan times come from. Probability distributions can be represented in Jaamsim with distribution objects. If we examine the PatientArrival object we see two keywords FirstArrivalTime and InterArrivalTime which determine the rate that the entities are created. For a Poisson process with an average of 8 arrivals per hour the interarrival times can be modelled by an exponential distribution with mean 0.125 hours. We therefore go into the Probability Distributions palette in the Model Builder window and create an ExponentialDistribution object and name it **ArrivalDistribution**. First we set the UnitType keyword to be **TimeUnit**, then we set the mean of the distribution to **0.125 h**. The UnitType tells Jaamsim what type of value we want the distribution object to create, in our case this is the time between arrivals in hours, which is a unit of time. Also make sure that the **RandomSeed** is 1, this determines the seed for the random number generator. Table 6.1 shows the keywords and values for the **ArrivalDistribution** object.

We need to repeat these steps for the check in and scan processes, which follow uniform and log-normal distributions respectively, so create a UniformDistribution object called **CheckInDistribution** and a LogNormalDistribution object called **ScanDistribution**. Then update the keywords of the distribution objects as follows in Table 6.2:

The final object we need at this stage is a Statistics object, to capture some output about the patients. This is found under the ProcessFlow palette, create a Statistics object and call it **TimeInSystem**.

Table 6.2: Check In and Scan Distributions

Object	Keyword	Value
CheckInDistribtuion	UnitType	TimeUnit
	RandomSeed	2
	MinValue	2 min
	MaxValue	5 min
ScanDistribution	UnitType	TimeUnit
	RandomSeed	3
	Scale	1 h
	NormalMean	-1.34
	NormalSD	0.29

At this point you should have the objects shown in Table 6.3 in your simulation.

Once you have created all of these objects lay them out similarly to as shown in Figure 6.2.

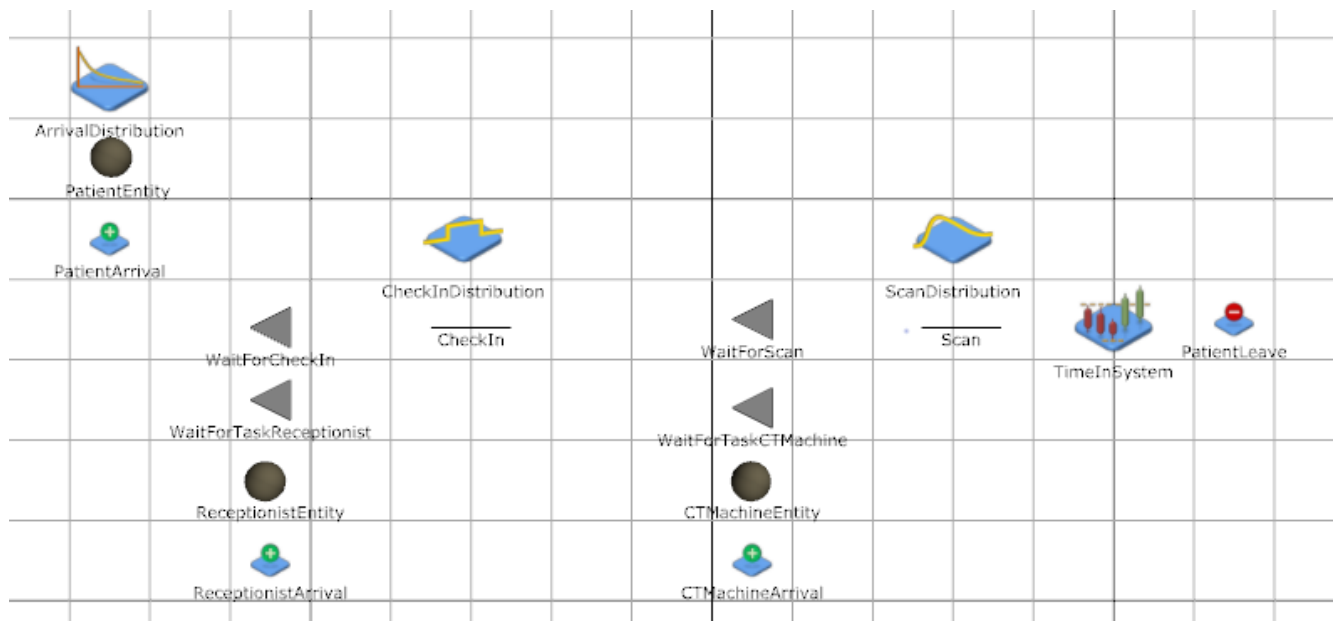


Figure 6.2: Screenshot of Simulation Model Layout

Create a new folder in the **labs** folder called **RC1** and save your simulation as **radiology_lab.cfg** or something similar inside the new folder. Also take this opportunity to change the graphics of the PatientEntity, ReceptionistEntity and CTMachineEntity. Download the patient.png, receptionist.png, and ctscanner.png (icons made by Freepik from www.flaticon.com) files from Canvas and save them in the same folder as your simulation .cfg file. Then in Jaamsim right click on PatientEntity and select Change Graphics. Click on Import and navigate to your downloaded patient.png, import it (it may be called patient-model) and accept the change. Repeat the process for the receptionists, and CT scanners.

6.2.2 Configuring Objects

Now that we have created the objects we need, we need to set the options for the each of them, starting with the ArriveEvents. The PatientArrival

Table 6.3: Model Objects

Object Type	Name
ActiveEntity	PatientEntity
ActiveEntity	ReceptionistEntity
ActiveEntity	CTMachineEntity
ArriveEvent	PatientArrival
ArriveEvent	ReceptionistArrival
ArriveEvent	CTMachineArrival
LeaveEvent	PatientLeave
WaitActivity	WaitForCheckIn
WaitActivity	WaitForScan
WaitActivity	WaitForTaskReceptionist
WaitActivity	WaitForTaskCTMachine
ProcessActivity	CheckIn
ProcessActivity	Scan
ExponentialDistribution	ArrivalDistribution
UniformDistribution	CheckInDistribution
LogNormalDistribution	ScanDistribution
Statistics	TimeInSystem

should have both the first arrival time and inter arrival times set by the ArrivalDistribution object, use the PatientEntity as a prototype, and the NextAEJObject should be WaitForCheckIn. NextAEJObject stands for next activity/event/Jaamsim object and refers to the fact that the next place an entity goes could be a standard Jaamsim object or a custom HCCM activity or event. For the arrive events we set NextAEJObject to the object that represents the activity that is transitioned to at the end of the event state changes in the conceptual model. For the ReceptionistArrival and CTMachineArrival we need to: set the prototype entity; both MaxNumber and InitialNumber (1 for receptionist, 3 for CT Machine); and set the NextAEJObject to the respective wait activity.

Next we will set the options for the Process Activities (and Statistics) so that the routing/flow for the entities is correct. The Check In activity has both the Patient and Receptionist as participants so we set the Participant list to PatientEntity, ReceptionistEntity. The duration is determined by the check in distribution, so we just set the duration to be CheckInDistribution object. After Check In the Patient starts waiting for a scan and the receptionist goes back to waiting for a task, so we set the NextAEJList to WaitForScan, WaitForTaskReceptionist. The Scan activity has both the Patient and CTMachine as participants and the duration is determined by the ScanDistribution object. After Scan the Patient should just leave, but we want to record some statistics first so we send it to TimeInSystem, and the CTMachine goes back to WaitForTaskCTMachine. For Process Activities the NextAEJList is similar to the NextAEJObject from the Arrive Events (which is similar to NextComponent), the difference is that a list of next objects is given, one for each of the participants in the activity. The participants are sent to the corresponding element of the list so it is important that the next activities are in the same order as the participants.

Note that when you click on the checkboxes in the popup menu for both Par-

Table 6.4: Arrival Event Parameters

Object	Tab	Keyword	Value
PatientArrival	Key Inputs	PrototypeEntity	PatientEntity
PatientArrival	Key Inputs	FirstArrivalTime	ArrivalDistribution
PatientArrival	Key Inputs	InterArrivalTime	ArrivalDistribution
PatientArrival	HCCM	NextAEJObject	WaitForCheckIn
ReceptionistArrival	Key Inputs	PrototypeEntity	ReceptionistEntity
ReceptionistArrival	Key Inputs	MaxNumber	1
ReceptionistArrival	Key Inputs	InitialNumber	1
ReceptionistArrival	HCCM	NextAEJObject	WaitForTaskReceptionist
CTMachineArrival	Key Inputs	PrototypeEntity	CTMachineEntity
CTMachineArrival	Key Inputs	MaxNumber	3
CTMachineArrival	Key Inputs	InitialNumber	3
CTMachineArrival	HCCM	NextAEJObject	WaitForTaskCTMachine

Table 6.5: Process Activity Parameters

Object	Tab	Keyword	Value
CheckIn	Key Inputs	Duration	CheckInDistribution
CheckIn	HCCM	ParticipantList	PatientEntity ReceptionistEntity
CheckIn	HCCM	NextAEJList	WaitForScan WaitForTaskReceptionist
Scan	Key Inputs	Duration	ScanDistribution
Scan	HCCM	ParticipantList	PatientEntity CTMachineEntity
Scan	HCCM	NextAEJList	TimeInSystem WaitForTaskCTMachine

Table 6.6: Collecting Statistics

Object	Keyword	Value
TimeInSystem	NextComponent	PatientLeave
	UnitType	TimeUnit
	SampleValue	this.obj.TotalTime

participantList and NextAEJList the items are added in alphabetical order, not the order you click them in. This is particularly important for the Scan activity as the CTMachineEntity comes before the PatientEntity alphabetically, but for the next activities TimeInSystem is before WaitForTaskCTMachine alphabetically so the two lists will not be in the same order.

The last object we need to configure before the simulation will run (it will run but it will not quite work correctly) is the TimeInSystem object. This is a Statistics object which collects a value from each Entity that passes through it and outputs the mean of the sampled values. We then need to finish the routing so that patients leave after going through the TimeInSystem, and tell the Statistics object which value to record as shown in Table 7.7, note that **this** refers to the Statistics object itself, **obj** refers to the entity that the Statistics object is currently processing, and **TotalTime** is an output on the entity that stores the total time that the entity has been in the simulation for.

Save your simulation again. If you run your simulation now you should see one receptionist arrive and wait, three CT machines arrive and wait, and patients arrive, and wait for check in. However nothing else will happen

and all of the entities will simply be waiting, this is because we have not specified any logic to be triggered when the entities start waiting.

6.3 Model Logic – Java

In your VSCode project you should have a folder called **sim_custom** under the Explorer tab on the left-hand side in VSCode. First right-click on this folder and select New Java Package. Enter **labs** for the name of the package and press Enter.

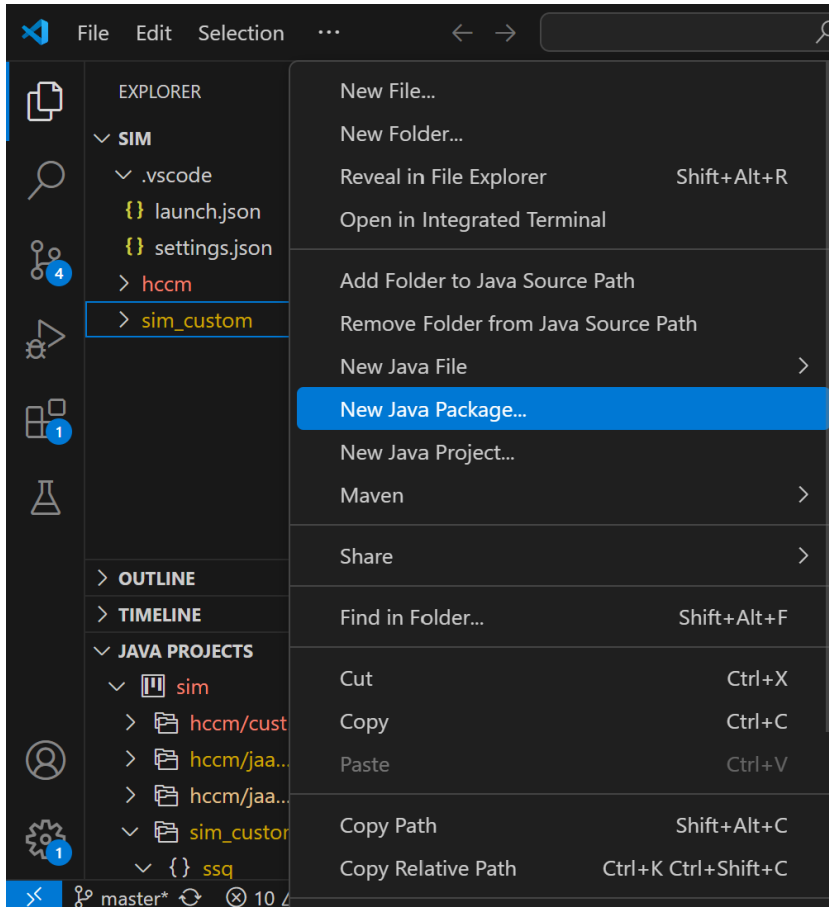


Figure 6.3: First Step in Creating a New Package

A new folder called **labs** should have been created within the **sim_custom** folder. Right click on the newly created **labs** folder and select New Java File → Class. Name the Class **RadiologyControlUnit** and press Enter.

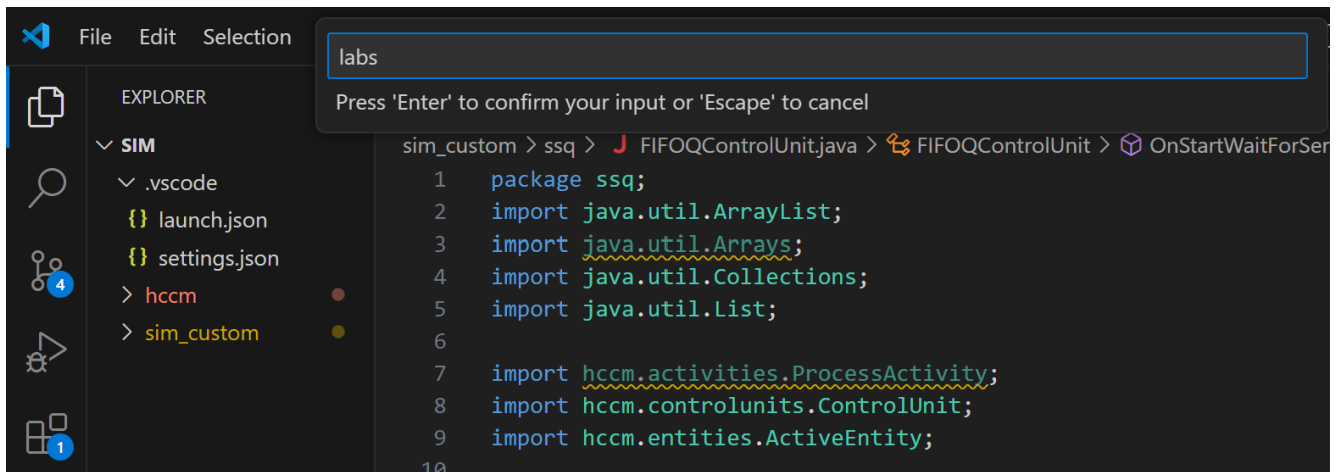


Figure 6.4: Second Step in Creating a New Package

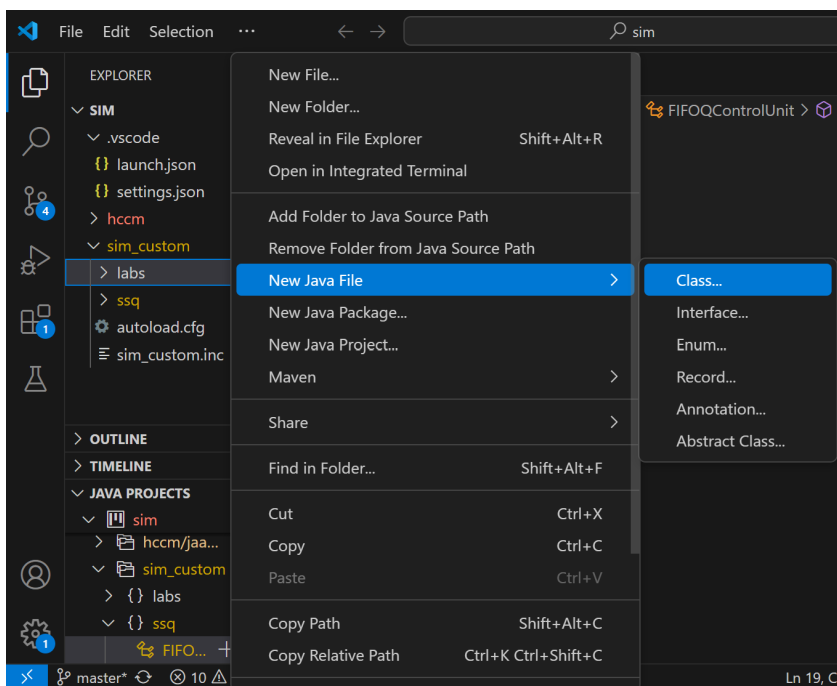


Figure 6.5: First Step in Creating a New Class

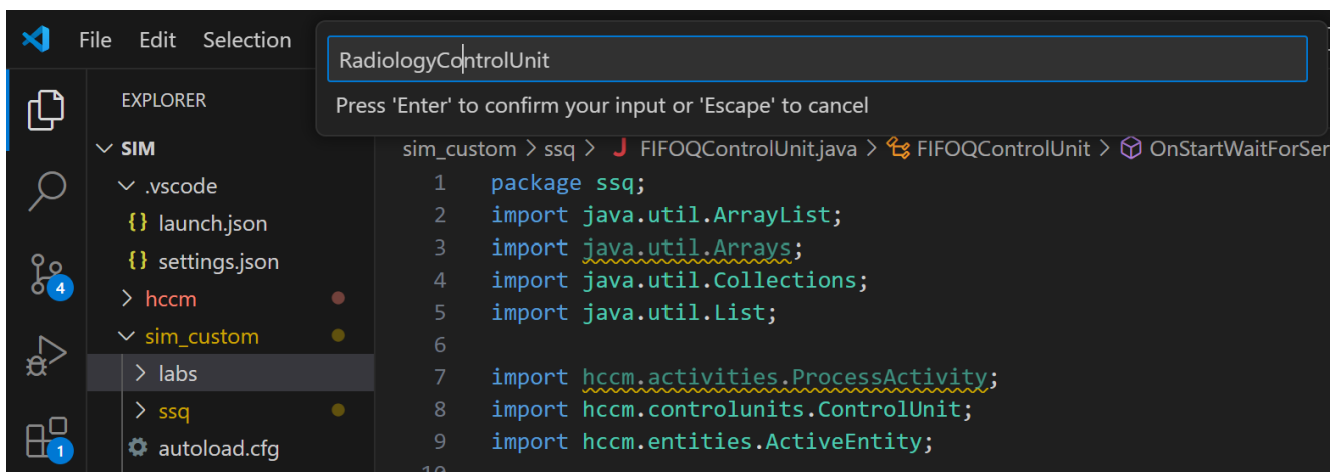


Figure 6.6: Second Step in Creating a New Class

The final step required to make this new object available in the simulation is to add to the contents of the `sim_custom.inc` file that we put in `sim → hccm → jaamsim → src → main → resources → resources → inputs`. There should already be some code there from the previous lab, so you only need to add lines 3, 7, and 10. If you want to copy and paste this make sure the quotes are copied correctly and the returns (arrows) are removed. Alternatively there is a new `sim_custom.inc` file here XXX on Canvas under Jaamsim Lab 1 that you can use directly.

```

1 Define ObjectType {
2     FIFOQControlUnit
3     RadiologyControlUnit
4 }
5
6 ControllerIconModel ImageFile {
7     ↪ '<res>/images/Controller-256.png' } Transparent { TRUE
8     ↪ }
9 AssembleIconModel ImageFile {
10    ↪ '<res>/images/Assemble-256.png' } Transparent { TRUE }
11
12 FIFOQControlUnit JavaClass { ssq.FIFOQControlUnit } Palette
13    ↪ { 'Single Server Queue' } DefaultDisplayModel {
14    ↪ ControllerIconModel } IconFile {
15    ↪ '<res>/images/Controller-24.png' } DefaultSize { 0.5 0.5
16    ↪ 0.5 m }
17 RadiologyControlUnit JavaClass { labs.RadiologyControlUnit }
18    ↪ Palette { 'Custom Logic' } DefaultDisplayModel {
19    ↪ AssembleIconModel } IconFile {
20    ↪ '<res>/images/Assemble-24.png' } DefaultSize { 0.5 0.5
21    ↪ 0.5 m }

```

Once you have updated the `sim_custom.inc` file, restart Jaamsim. If everything is working correctly the `RadiologyControlUnit` object should now be available under the Custom Logic palette as shown in the screenshot below:

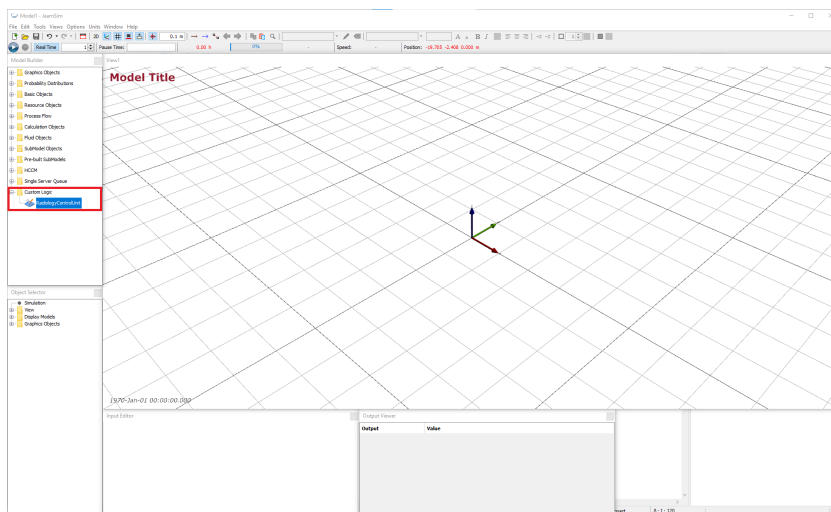


Figure 6.7: Screenshot of Control Unit Object

Once you have the new `RadiologyControlUnit` object available open your simulation and create one.

We now need to add the Java code to the new RadiologyControlUnit class to run the control policies. First add the following imports under the package declaration. **Note** These code snippets for this lab are provided in a separate file for you here XXX.

```
1 package labs;
2 import java.util.ArrayList;
3 import java.util.Arrays;
4 import java.util.Collections;
5 import java.util.List;
6
7 import hccm.activities.ProcessActivity;
8 import hccm.controlunits.ControlUnit;
9 import hccm.entities.ActiveEntity;
```

Then, within the definition of the class we need to create four methods that represent the four control policies in the model. Each control policy is a public method of the class that does not return any value (is void) and takes both a list of Active Entities, and the simulation time as inputs. We will use the same names for the methods as the control policies in the conceptual model: **OnStartWaitForCheckIn**, **OnStartWaitForScan**, **OnStartWaitForTaskReceptionist**, and **OnStartWaitForTaskCTMachine**. In the first of these, **OnStartWaitForCheckIn** we first need to get a list of the Receptionist Entities that are currently in the "WaitForTaskReceptionist" activity, and we also create a comparator object that is used to sort a list of entities by when they started their current activity.

Once we have the list of idle receptionists we check whether it is not empty, and if it isn't proceed to sort it, select the first one, and transition the patient and receptionist to the check in activity.

```
1 public void OnStartWaitForCheckIn(List<ActiveEntity> ents, double simTime) {
2
3     ArrayList<ActiveEntity> idleReceps = this.getEntitiesInActivity("ReceptionistEntity",
4     ↪ "WaitForTaskReceptionist", simTime);
5     ActivityStartCompare actSartComp = this.new ActivityStartCompare();
6
7     if (idleReceps.size() > 0) {
8         Collections.sort(idleReceps, actSartComp);
9
10        ActiveEntity patient = ents.get(0);
11        ActiveEntity receptionist = idleReceps.get(0);
12
13        transitionTo("CheckIn", patient, receptionist);
14    }
15 }
```

Similar methods are defined for the other control policies, with small changes based on the types of entities that are being checked, and the activity that is started. There are gaps that need to be filled in on lines 3, 24, and 42. In the first gap you need to create an array that contains all of the CT Machines that are currently idle. In the second, you need to select which of the patients that are currently waiting should do check in with the receptionist. In the third, you need to start the next activity with the patient and CT Machine. All of these have similar lines in the first method that you can use as a guide.

```

1 public void OnStartWaitForScan(List<ActiveEntity> ents, double simTime) {
2
3     // A //
4     ActivityStartCompare actSartComp = this.new ActivityStartCompare();
5
6     if (idleCTs.size() > 0) {
7         Collections.sort(idleCTs, actSartComp);
8
9         ActiveEntity patient = ents.get(0);
10        ActiveEntity ct = idleCTs.get(0);
11
12        transitionTo("Scan", patient, ct);
13    }
14 }
15
16 public void OnStartWaitForTaskReceptionist(List<ActiveEntity> ents, double simTime) {
17
18     ArrayList<ActiveEntity> waitPats = this.getEntitiesInActivity("PatientEntity",
19     ↪ "WaitForCheckIn", simTime);
20     ActivityStartCompare actSartComp = this.new ActivityStartCompare();
21
22     if (waitPats.size() > 0) {
23         Collections.sort(waitPats, actSartComp);
24
25         // B //
26         ActiveEntity receptionist = ents.get(0);
27
28         transitionTo("CheckIn", patient, receptionist);
29     }
30 }
31
32 public void OnStartWaitForTaskCTMachine(List<ActiveEntity> ents, double simTime) {
33
34     ArrayList<ActiveEntity> waitPats = this.getEntitiesInActivity("PatientEntity", "WaitForScan",
35     ↪ simTime);
36     ActivityStartCompare actSartComp = this.new ActivityStartCompare();
37
38     if (waitPats.size() > 0) {
39         Collections.sort(waitPats, actSartComp);
40
41         ActiveEntity patient = waitPats.get(0);
42         ActiveEntity ct = ents.get(0);
43
44         // C //
45     }
46 }

```

Table 6.7: Trigger Parameters

Object	Tab	Keyword	Value
StartWaitCheckIn	HCCM	ControlUnit	RadiologyControlUnit1
StartWaitCheckIn	HCCM	ControlPolicy	OnStartWaitForCheckIn
StartWaitScan	HCCM	ControlUnit	RadiologyControlUnit1
StartWaitScan	HCCM	ControlPolicy	OnStartWaitForScan
StartWaitTaskReceptionist	HCCM	ControlUnit	RadiologyControlUnit1
StartWaitTaskReceptionist	HCCM	ControlPolicy	OnStartWaitForTaskReceptionist
StartWaitTaskCTMachine	HCCM	ControlUnit	RadiologyControlUnit1
StartWaitTaskCTMachine	HCCM	ControlPolicy	OnStartWaitForTaskCTMachine

Table 6.8: Wait Activity Parameters

Object	Tab	Keyword	Value
WaitForCheckIn	HCCM	StartTriggerList	StartWaitCheckIn
WaitForCheckIn	HCCM	StartTriggerChoice	1
WaitForScan	HCCM	StartTriggerList	StartWaitScan
WaitForScan	HCCM	StartTriggerChoice	1
WaitForTaskReceptionist	HCCM	StartTriggerList	StartWaitTaskReceptionist
WaitForTaskReceptionist	HCCM	StartTriggerChoice	1
WaitForTaskCTMachine	HCCM	StartTriggerList	StartWaitTaskCTMachine
WaitForTaskCTMachine	HCCM	StartTriggerChoice	1

The final step needed to get this logic into the simulation is to define Triggers that initiate these methods and where/when they should be called. To do this create four Trigger objects, called **StartWaitCheckIn**, **StartWaitScan**, **StartWaitTaskReceptionist**, and **StartWaitTaskCTMachine** from the HCCM palette and set the ControlUnit and ControlPolicy for each one. The value of the ControlPolicy keyword needs to exactly match the name of the method you have defined in the java code.

Then update the parameters in the Wait Activities that these control policies should be triggered in:

Now if you save and run your simulation you should be able to see patients arriving, checking in, being scanned, and leaving. If you get an error saying that a method cannot be found on the control unit, first make sure that all of the ControlPolicy inputs exactly match the names of the methods in the control unit java file. Then try closing Jaamsim, cleaning your project, and restarting Jaamsim.

6.4 Model Output

To perform different experiments and multiple replications we make use of Jaamsim's MultipleRuns feature which can be found in the Simulation object at the top of the Object Selector window. Here we can use the **NumberOfReplications** to control how many replications are performed. We want to do 50 replications so we set NumberOfReplications to 50.

Table 6.9: Simulation Parameters

Object	Tab	Keyword	Value
Simulation	Key Inputs	RunDuration	7 d
Simulation	Key Inputs	RunOutputList	{[TimeInSystem].SampleAverage / 1[h]}
Simulation	Multiple Runs	NumberOfReplications	50
Simulation	Multiple Runs	PrintConfidenceIntervals	FALSE

We want each replication to run for one week, so we set `RunDuration` to **7d**. To record outputs we can make use of the `Simulation` object's `RunOutputList`, which saves the final value of outputs at the end of each run. The scenario number, and the replication number are saved by default (by default `PrintRunLabels` and `PrintReplications` are `TRUE`), but we will calculate confidence intervals ourselves so we set `PrintConfidenceIntervals` to `FALSE`. Because `ActiveEntities` are removed from the simulation when they enter a `LeaveEvent`, we cannot get the total time that each patient spends in the clinic at the end of the run. This is why we created a `Statistics` object called `TimeInSystem` that records how long they have been in the system before they are destroyed. We can use the `SampleAverage` output of the `TimeInSystem` object in the `Simulation`'s `RunOutputList` to output the mean time in system for each replication. **Note** The `SampleAverage` is divided by 1[h] to give a raw number in hours for later processing in Python. Otherwise `JaamSim` writes an h to the data file.

Now if you save and run your simulation a file should be created called 'yourSimulationName.dat'. To speed up running the simulation you can turn off the option 'Real time', in the top left corner next to the play button.

With the model complete and the results recorded we can use Python to analyse them. First download the Python analysis file provided, then change name of the .dat file to match yours and make sure it is in the same directory as the Python file, then run the Python file. The following output should be printed:

```

Scenario  Replication  TimeInSystem
0          1           1.0      0.443924
1          1           2.0      0.521371
2          1           3.0      0.441290
3          1           4.0      0.418234
4          1           5.0      0.519311
Mean                               0.449299
CI Half Width                     0.007243
Name: TimeInSystem, dtype: float64
```

6.5 Task

Construct a 95% confidence interval for the average utilisation of the three CT machines in each experiment. You will need to add an entry to the `RunOutputList`. You should get the following output:

```

Mean          0.730582
CI Half Width  0.006098
Name: Utilisation, dtype: float64
```

Hint: there are many ways to do this. Have a look at the outputs provided on the wait activity **WaitForTaskCTMachine**, can you calculate the total time that the three CTMachines have spent waiting using these outputs?

7 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` XXX file link.

7.1 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 7.1:

Table 7.1: Patient Priority Proportions

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

7.2 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 **radiology_lab_extended.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 7.2 shows how to create the priority attribute and make the default value 0.

Table 7.2: Priority Attribute

Object	Keyword	Value
PatientEntity	AttributeDefinitionList	{ priority 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 occurring. Create a DiscreteDistribution object called PriorityDistribution with the values shown in Table 7.3.

Table 7.3: 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 7.4: 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 priority 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 7.5: Priority Branch

Object	Keyword	Value
PriorityBranch	NextComponentList	WaitForScan WaitForCheckIn
	Choice	'this.obj.priority ≤ 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 7.6: 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 **RadiologyExtendedControlUnit** in the labs package, and copy the java code from the RadiologyControlUnit to the new RadiologyExtendedControlUnit. Add the relevant lines to the **sim_custom.inc** file so that it is available in the HCCM palette. Then replace the RadiologyControlUnit with a RadiologyExtendedControlUnit, and replace the references to the RadiologyControlUnit with RadiologyExtendedControlUnit 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 comparator 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",
4         ↪ simTime);
5     AttributeCompare priorityComp = new AttributeCompare("priority");
6     ActivityStartCompare actStartComp = this.new ActivityStartCompare();
7     ActiveEntity ct = ents.get(0);
8     int reqMaintenance = (int) getNumAttribute(ct, "NeedMaintenance", simTime, -1);
9
10    if (reqMaintenance == 1) {
11        // A //
12    }
13
14    else if (waitPats.size() > 0) {
15        Collections.sort(waitPats, priorityComp);
16
17        int highestPriority = // B //
18
19        ArrayList<ActiveEntity> priorityPatients = new ArrayList<ActiveEntity>();
20        for (ActiveEntity wP : waitPats) {
21            int patPri = // C //
22            if (patPri == highestPriority) {
23                priorityPatients.add(wP);
24            }
25        }
26
27        Collections.sort(priorityPatients, actStartComp);
28        ActiveEntity patient = priorityPatients.get(0);
29
30        transitionTo("Scan", patient, ct);
31    }
32 }

```

Then create two new methods called `OnCTArrival` and `OnRequireMaintenance`:

```

1 public void OnCTArrival(List<ActiveEntity> ents, double simTime) {
2
3     double maintenanceTimeGap = 8 * 60 * 60;
4     LogicEvent le = (LogicEvent) getSubmodelEntity("RequireMaintenance");
5
6     le.scheduleEvent(ents, maintenanceTimeGap);
7 }
8
9 public void OnRequireMaintenance(List<ActiveEntity> ents, double simTime) {
10
11     double maintenanceTimeGap = 8 * 60 * 60;
12     LogicEvent le = (LogicEvent) getSubmodelEntity("RequireMaintenance");
13
14     le.scheduleEvent(ents, simTime + maintenanceTimeGap);
15
16     ActiveEntity ct = ents.get(0);
17     if (ct.getCurrentActivity(simTime).equals("WaitForTaskCTMachine")) {
18         transitionTo("Maintenance", ct);
19     }
20 }

```

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 `RadiologyExtendedControlUnit`, 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 90th percentile of time that patients spend in the clinic. Unfortunately the `Statistics` object does not provide the 90th percentile as an output. Therefore we need to capture each of the individual times that each patient spends in the clinic and calculate the 90th 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 7.7, note that **TotalTime** is an output on the entity which stores the total time that the entity has been in the simulation for:

Table 7.7: 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 the CT Machines performing maintenance every 8 hours.

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 **radiology_lab_extended.dat**.

With the model complete and the results recorded we can use Python to analyse them. First download the Python analysis file provided, change the name of the .log file to match yours, and make sure it is in the same directory as the Python file, then run the Python file. The following table should be printed:

	TimeInSystem
Mean	0.893513
CI_Half_Width	0.028886

7.3 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:

	Mean	CI_Half_Width
Priority		
1.0	0.481123	0.011888
2.0	0.501677	0.006659
3.0	0.649709	0.013620
4.0	0.886900	0.023870
5.0	1.755913	0.128359

Part IV

Conceptual Models

8 Making Paper Cars Conceptual Model

8.1 Understanding of the Problem Situation

In the box below write a problem description for making paper cars, think about what you are trying to solve/discover by simulating this activity. You may want to look at Chapter 1 again to remind yourself about the process.

8.2 Modelling Objectives

In the box below write the modelling objectives for making paper cars, i.e., what are you trying to discover using simulation?

8.3 General Objectives

In the box below write the general objectives for making paper cars, i.e., what are some of the general properties you'd like your simulation to have?

8.4 Defining Output Responses

In the box below write the output responses for making paper cars, i.e., what are you going to measure to determine the performance of the system?

8.5 Defining Input Factors

In the box below write the input factors for making paper cars, i.e., what are you going to change to achieve the modelling objectives?

8.6 Identifying Entities

In the box below list the entities for making paper cars.

8.7 Drawing Behavioural Paths

The activity diagrams for the pencil & template, and scissors are given below in Figures 8.1, and 8.2.

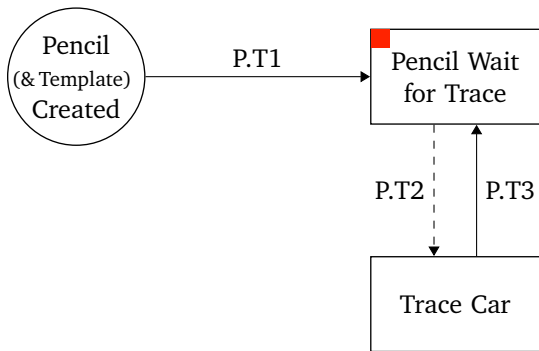


Figure 8.1: Pencil Activity Diagram

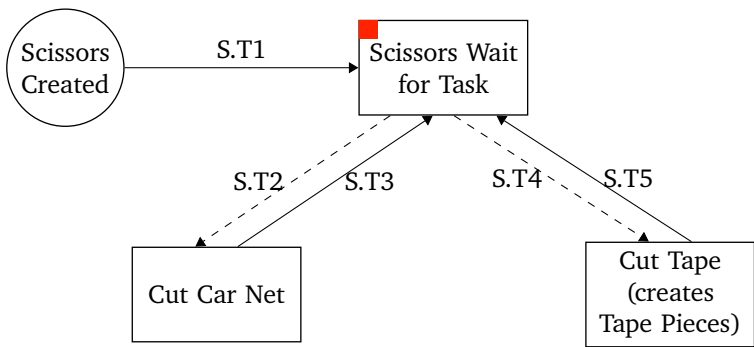
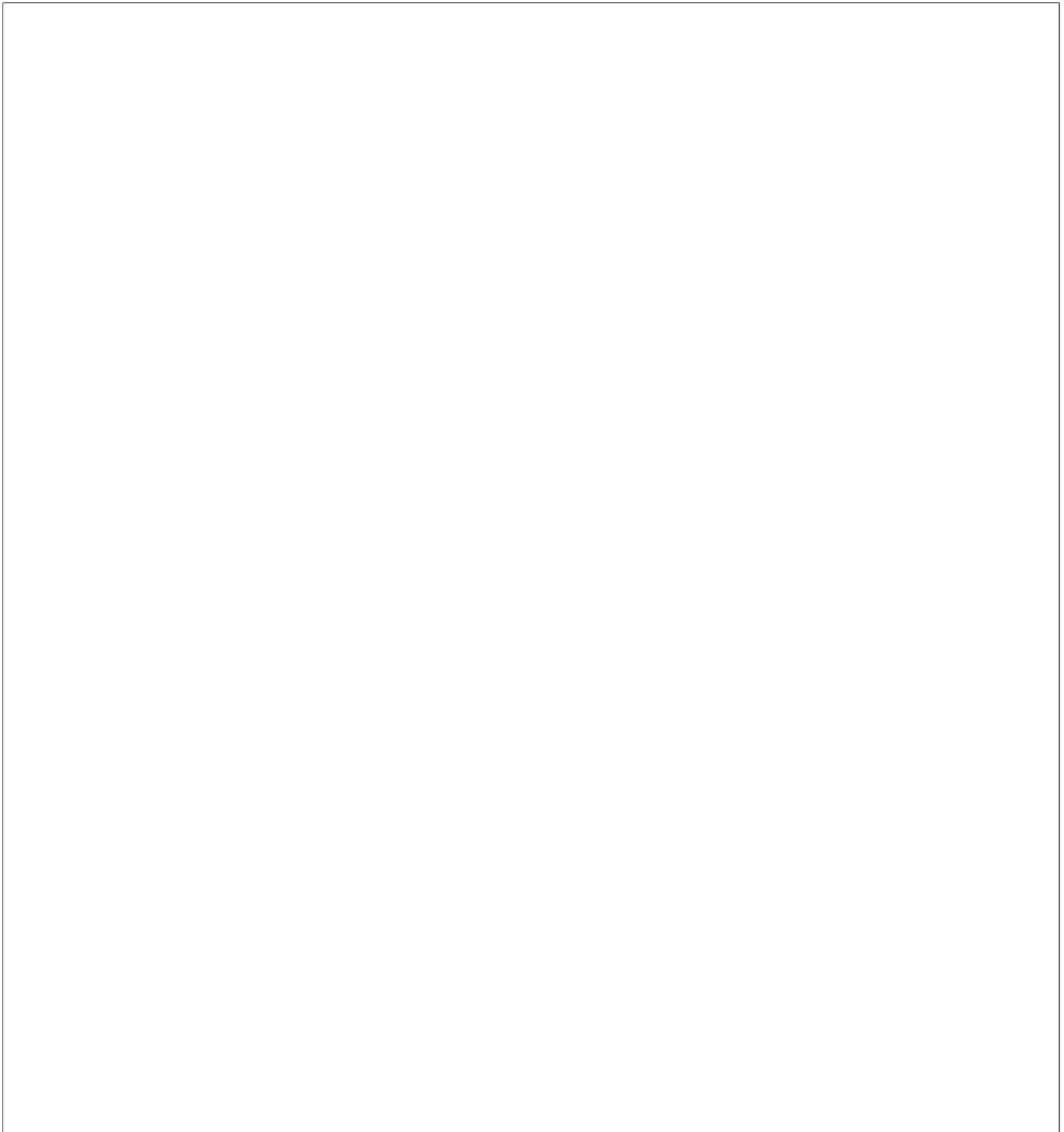
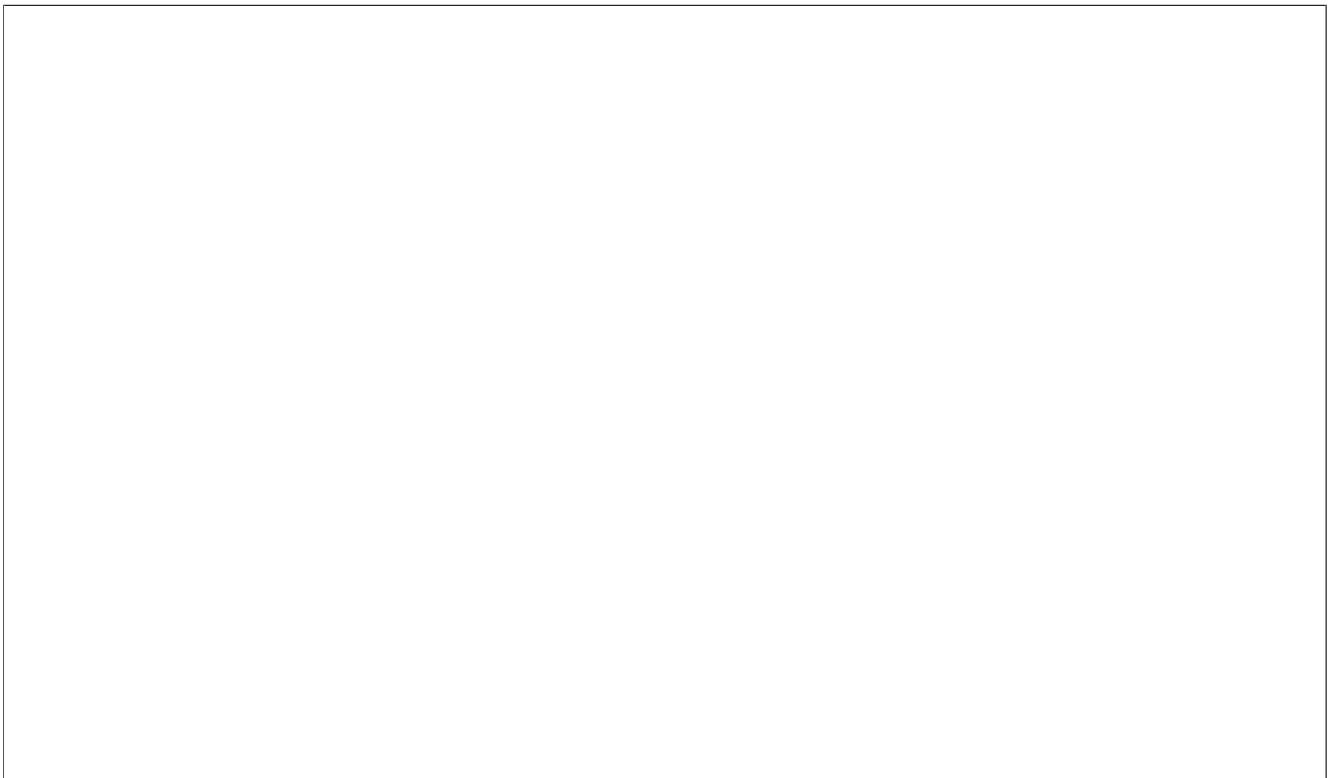


Figure 8.2: Scissors Activity Diagram

In the boxes below draw the activity diagrams for the remaining entities.

A large, empty rectangular box with a thin black border, intended for drawing activity diagrams for the remaining entities.





9 Radiology Clinic

9.1 Data

Table 9.1: 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 9.2: 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

9.2 Components

Table 9.3: List of Entities

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

Table 9.4: 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 9.5: 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	

9.3 Activity Diagrams

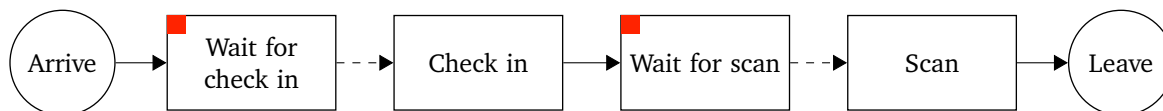


Figure 9.1: Patient Activity Diagram

Table 9.6: Events

Activity	Participants	Type	State Change
Simulation Start	-	Scheduled	1 SCHEDULE Arrival (R) at TIME 2 SCHEDULE Arrival (CT) at TIME 3 SCHEDULE Arrival (P) at TIME + PatientInterArrival()
Arrival (P)	Patient (p)	Scheduled	1 p.ID = NextPatIDNum 2 p.Priority = PatientPriority() 3 NextPatIDNum = NextPatIDNum + 1 4 SCHEDULE Arrival (P) at TIME + PatientInterArrival() 5 START Wait for Check In WITH p
Leave (P)	Patient (p)	Scheduled	1 Calculate statistics for p
Arrival (R)	Receptionist (r)	Scheduled	1 r.ID = NextReceptionistIDNum 2 NextReceptionistIDNum = NextReceptionistIDNum + 1 3 IF NextReceptionistIDNum <= NumReceptionists THEN 4 SCHEDULE Arrival (R) at TIME 5 END IF 6 START Wait for Task (R) WITH r
Leave (R)	Receptionist (r)	Scheduled	1 Calculate statistics for r
Arrival (CT)	CTMachine (c)	Scheduled	1 c.ID = NextCTMachineIDNum 2 NextCTMachineIDNum = NextCTMachineIDNum + 1 3 IF NextCTMachineIDNum <= NumCTMachines THEN 4 SCHEDULE Arrival (CT) at TIME 5 END IF 6 START Wait for Task (CT) WITH c
Leave (P)	CTMachine (c)	Scheduled	1 Calculate statistics for c

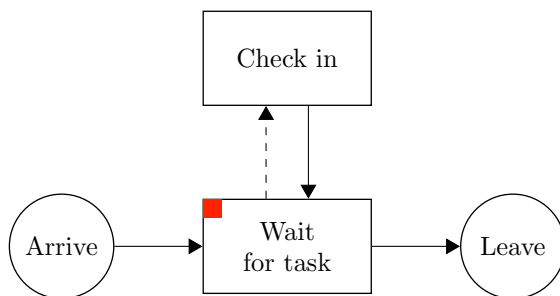


Figure 9.2: Receptionist Activity Diagram

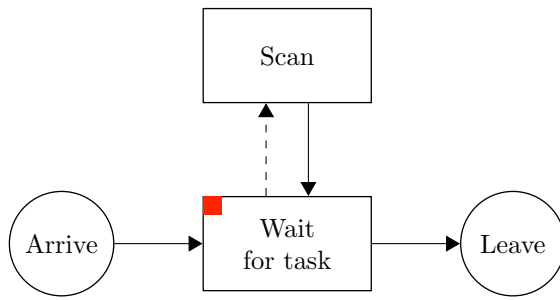


Figure 9.3: CT Activity Diagram

9.4 Control Policies

Table 9.7: OnStartWaitForCheckIn

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

Table 9.8: 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 9.9: 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 9.10: OnStartWaitForTaskCT

Triggered by: CTMachine c	
1	patients = {p FOR p IN P IF p.State = "Wait for Scan"}
2	IF patients IS NOT empty THEN
3	p_hat = argmin{p.CurrentStart FOR p IN patients}
4	START Scan WITH p, r_hat
5	END IF