# ENGSCI 355 Labs

Thomas Adams

2024-08-29

# Table of contents

# Preface

These are the Labs for ENGSCI 355.

To learn more about Quarto books visit https://quarto.org/docs/books.

# 1 Jaamsim – Radiology Clinic Lab

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 separately in the file Radiology_Lab_1_CM.pdf.

## 1.1 Experiments

We will perform just one experiment, using distributions for the arrival, check in, and scan processes. We will use a Poisson distribution with $\lambda = 8$/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.

## 1.2 Jaamsim Model

### 1.2.1 Creating Model Objects

Run Jaamsim by opening Eclipse 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 1.1. 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 at the clinic. The ArriveEvent object creates a series of entities that are
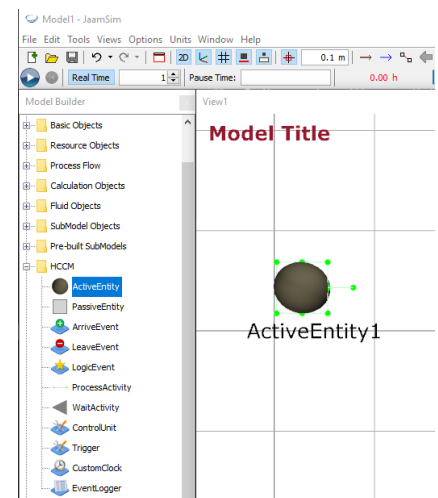


Figure 1.1: Screenshot of an ActiveEntity

4

Table 1.1: Arrival Distribution Inputs

| Object | Keyword | Value |
|---|---|---|
| ArrivalDistribution | UnitType | TimeUnit |
| | RandomSeed | 1 |
| | Mean | 0.125 h |

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 1.1 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 1.2:

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`.

At this point you should have the objects shown in Table 1.3 in your simulation.

Table 1.2: 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 |

Table 1.3: 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 |

Once you have created all of these objects lay them out similarly to as shown in Figure 1.2.
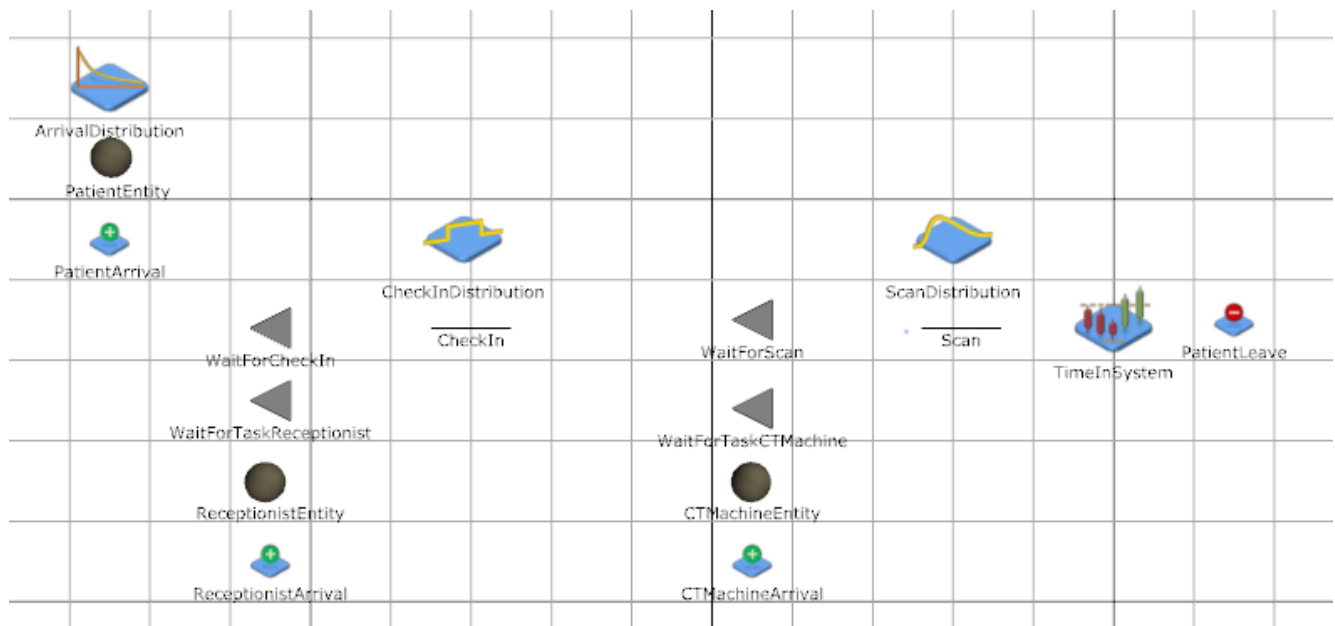


Figure 1.2: Screenshot of Simulation Model Layout

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.

### 1.2.2 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 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

7

Table 1.4: 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 1.5: 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 |

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 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 1.6, 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

Table 1.6: Collecting Statistics

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

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.

## 1.3  Model Logic – Java

In your eclipse project you should have a folder called 'sim_custom' under the Project Explorer on the left hand side in Eclipse. First right-click on this folder and select New → Package. Enter labs for the name of the package and click Finish.



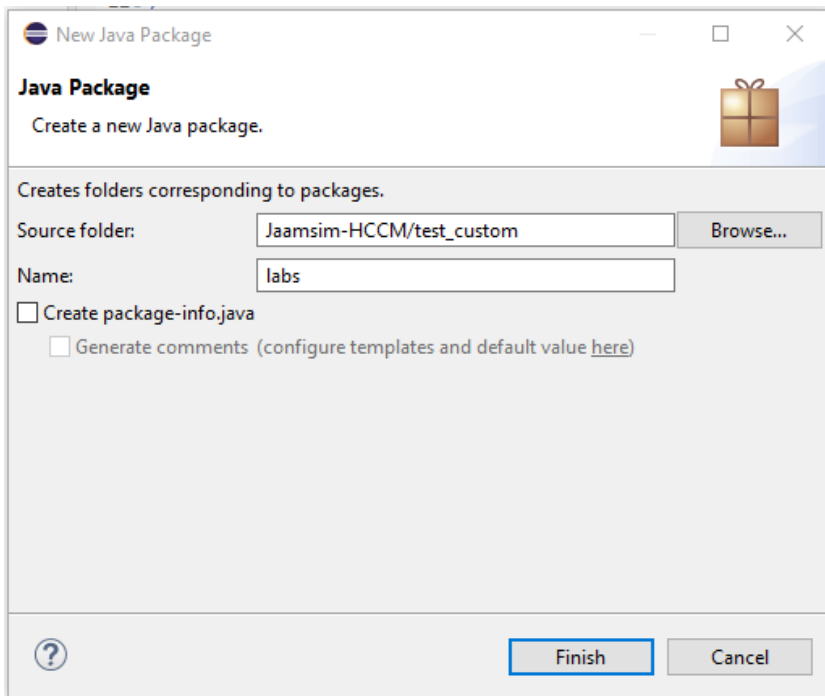Figure 1.3: First Step in Creating a New Package



Figure 1.4: Second Step in Creating a New Package

Right click on the newly created labs package and select New → Class. Name the Class RadiologyControlUnit and enter hccm.controlunits.ControlUnit for the Superclass.

What this has done is created a new Java class, based off the ControlUnit class, which we will use to implement the logic required for the clinic. 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 return (arrow) on line 10 is removed.
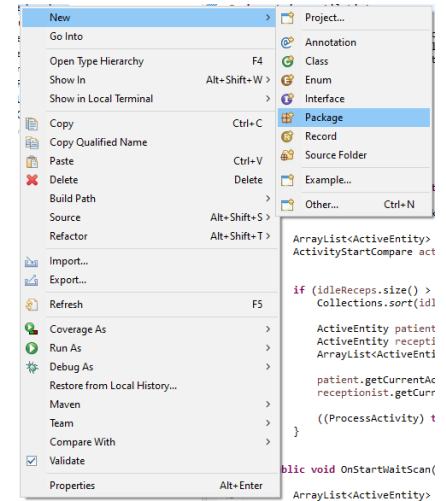
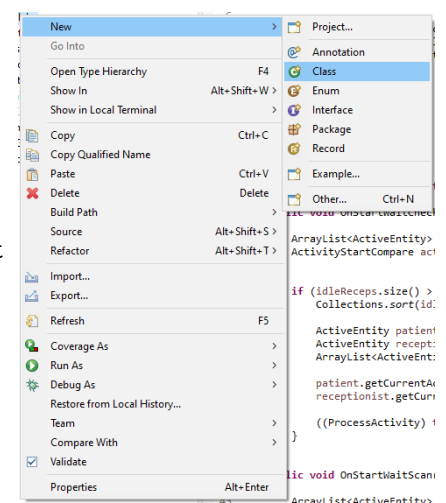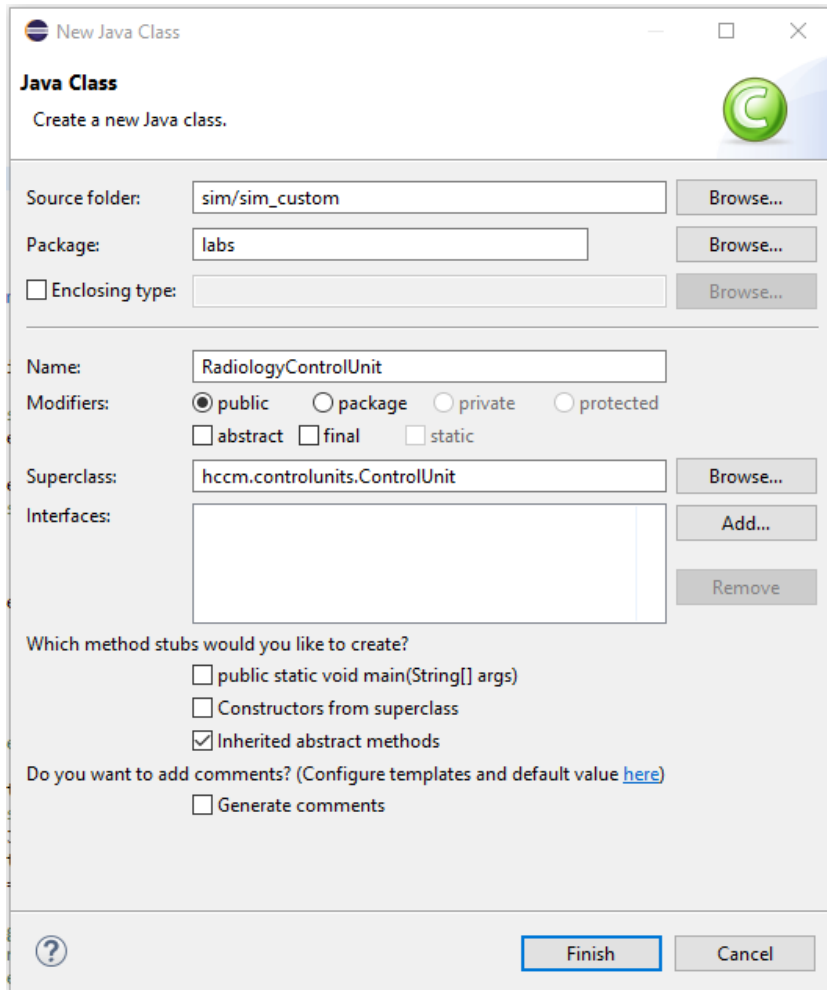

Figure 1.5: First Step in Creating a New Class

Figure 1.6: Second Step in Creating a New Class

Alternatively there is a new sim_custom.inc file on Canvas under Jaamsim Lab 1 that you can use directly.

```
1  Define ObjectType {
2      FIFOQControlUnit
3      RadiologyControlUnit
4  }
5
6  ControllerIconModel ImageFile { '<res>/images/Controller-256.png' }  Transparent { TRUE }
7  AssembleIconModel ImageFile { '<res>/images/Assemble-256.png' }  Transparent { TRUE }
8
9  FIFOQControlUnit JavaClass { ssq.FIFOQControlUnit }  Palette { 'Single Server Queue' } DefaultDisplayModel
10 RadiologyControlUnit JavaClass { labs.RadiologyControlUnit }  Palette { 'Custom Logic' } DefaultDisplayMod
```

Once you have updated the sim_custom.inc file, clean your project and 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:
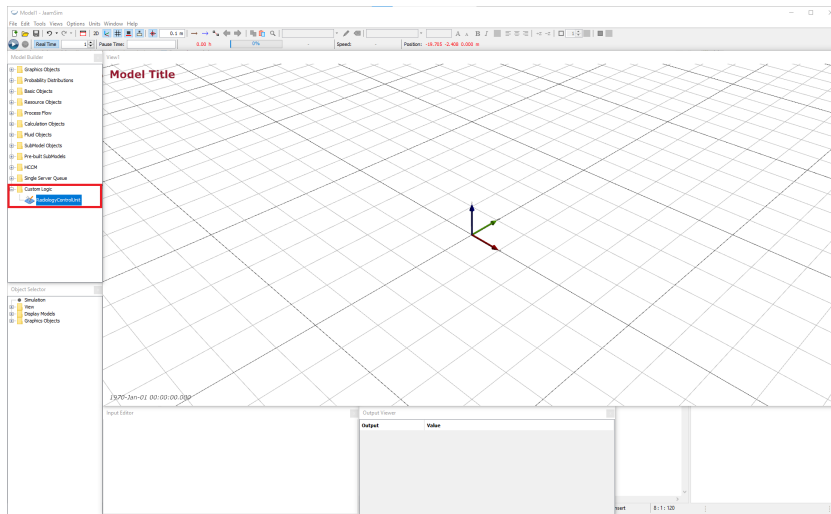


Figure 1.7: Screenshot of Control Unit Object

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

```
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

11

(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.

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

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.

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.

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

    // A //
    ActivityStartCompare actSartComp = this.new ActivityStartCompare();

    if (idleCTs.size() > 0) {
      Collections.sort(idleCTs, actSartComp);

      ActiveEntity patient = ents.get(0);
      ActiveEntity ct = idleCTs.get(0);

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

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

  ArrayList<ActiveEntity> waitPats = this.getEntitiesInActivity("PatientEntity", "WaitF
  ActivityStartCompare actSartComp = this.new ActivityStartCompare();

  if (waitPats.size() > 0) {
    Collections.sort(waitPats, actSartComp);

    // B //
    ActiveEntity receptionist = ents.get(0);

    transitionTo("CheckIn", patient, receptionist);
  }
}

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

  ArrayList<ActiveEntity> waitPats = this.getEntitiesInActivity("PatientEntity", "WaitF
  ActivityStartCompare actSartComp = this.new ActivityStartCompare();

  if (waitPats.size() > 0) {
    Collections.sort(waitPats, actSartComp);

    ActiveEntity patient = waitPats.get(0);
    ActiveEntity ct = ents.get(0);

    // C //
  }
}
```

Table 1.7: 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 1.8: 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 |

Table 1.9: 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 |

Table 1.10: R Output

| lower95CI | mean | upper95CI |
|---|---|---|
| 0.442 | 0.449 | 0.457 |

Table 1.11: Task Output

| lower95CI | mean | upper95CI |
|---|---|---|
| 0.724 | 0.731 | 0.737 |

## 1.4 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. 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 RunOutput-List, 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 R. 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 R to analyse them. First download the R analysis file provided, then change the working directory and name of the .dat and .log files to match yours, then run the R file. The following table should be printed:

## 1.5 Task

Construct a 95% confidence interval for the average utilisation of the three CT machines in each experiment. You should get the following output:

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?

# 2 Radiology Clinic Lab – Conceptual Model

## 2.1 Data

Table 2.1: 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 2.2: 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 |

## 2.2 Components

Table 2.3: List of Entities

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

Table 2.4: 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 2.5: 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 2 | TRANSITION 3 WITH p<br>TRANSITION 8 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 2 | TRANSITION 5 WITH p<br>TRANSITION 12 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 2.6: Events

| Activity | Participants | Type | State Change |
|---|---|---|---|
| Simulation Start | - | Scheduled | 1 SCHEDULE Arrival (R) at TIME<br>2 SCHEDULE Arrival (CT) at TIME<br>3 SCHEDULE Arrival (P) at TIME + PatientInterArrival() |
| Arrival (P) | Patient (p) | Scheduled | 1 p.ID = NextPatIDNum<br>2 p.Priority = PatientPriority()<br>3 NextPatIDNum = NextPatIDNum + 1<br>4 SCHEDULE Arrival (P) at TIME + PatientInterArrival()<br>5 TRANSITION 1 WITH p |
| Leave (P) | Patient (p) | Scheduled | 1 Calculate statistics for p |
| Arrival (R) | Receptionist (r) | Scheduled | 1 r.ID = NextReceptionistIDNum<br>2 NextReceptionistIDNum = NextReceptionistIDNum + 1<br>3 IF NextReceptionistIDNum <= NumReceptionists THEN<br>4    SCHEDULE Arrival (R) at TIME<br>5 END IF<br>6 TRANSITION 6 WITH r |
| Leave (R) | Receptionist (r) | Scheduled | 1 Calculate statistics for r |
| Arrival (CT) | CTMachine (c) | Scheduled | 1 c.ID = NextCTMachineIDNum<br>2 NextCTMachineIDNum = NextCTMachineIDNum + 1<br>3 IF NextCTMachineIDNum <= NumCTMachines THEN<br>4    SCHEDULE Arrival (CT) at TIME<br>5 END IF<br>6 TRANSITION 10 WITH c |
| Leave (P) | CTMachine (c) | Scheduled | 1 Calculate statistics for c |

## 2.3 Activity Diagrams
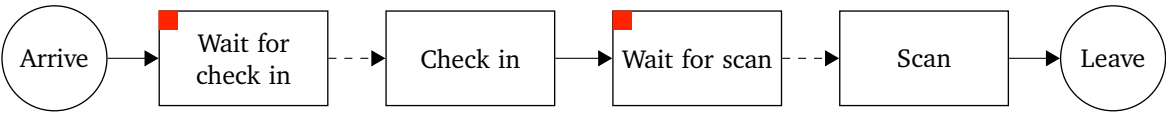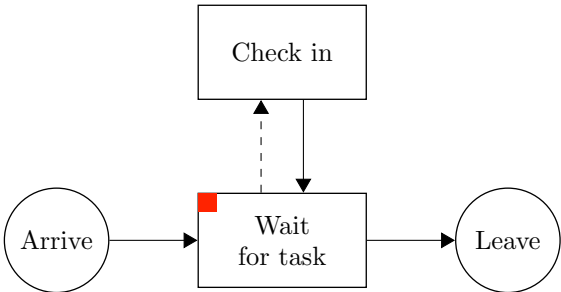


Figure 2.1: Patient Activity Diagram
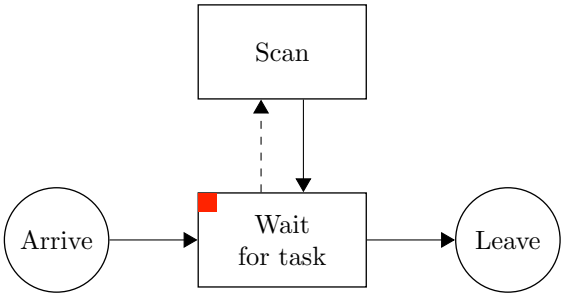


Figure 2.2: Receptionist Activity Diagram



Figure 2.3: CT Activity Diagram

## 2.4 Control Policies

Table 2.7: OnStartWaitForCheckIn

| Triggered by: Patient p |
| --- |

```
1  receps = {r FOR r IN R IF r.State = "Wait for task (R)"}
2  IF receps != {} THEN
3      r_hat = argmin{r.CurrentStart FOR r in receps}
4      TRANSITION 2 with p
5      TRANSITION 7 with r_hat
6  END IF
```

Table 2.8: OnStartWaitForScan

| Triggered by: Patient p |
| --- |

```
1  cts = {c FOR c IN C IF c.State = "Wait for task (C)"}
2  IF cts != {} THEN
3      c_hat = argmin{c.CurrentStart FOR c in cts}
4      TRANSITION 4 with p
5      TRANSITION 11 with c_hat
6  END IF
```

Table 2.9: OnStartWaitForTaskR

| Triggered by: Receptionist r |
| --- |

```
1  patients = {p FOR p IN P IF p.State = "Wait for Check In"}
2  IF patients != {} THEN
3      p_hat = argmin{p.CurrentStart FOR p in patients}
4      TRANSITION 2 with p_hat
5      TRANSITION 7 with r
6  END IF
```

Table 2.10: OnStartWaitForTaskCT

| Triggered by: CTMachine c |
| --- |

```
1  patients = {p FOR p IN P IF p.State = "Wait for Scan"}
2  IF patients != {} THEN
3      p_hat = argmin{p.CurrentStart FOR p in patients}
4      TRANSITION 4 with p_hat
5      TRANSITION 11 with c
6  END IF
```

# 3 Summary

In summary, this book has no content whatsoever.