ENGSCI 355 Labs

Thomas Adams

2025-02-18

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 7

2 HCCM Framework 8
2.1 Understanding of the Problem Situation 8
2.2 Identification of Modelling and General Objectives 9
2.3 Defining Output Responses 9
2.4 Defining Input Factors 9
2.5 Model Content 9
2.5.1 Identifying Entities 10
2.5.2 Drawing Behavioural Paths 10
2.5.3 Define the Data 10
2.5.4 Define the Structure 11
2.5.5 Define the Logic 12
2.6 Assumptions and Simplifications 12

3 Inputs, Outputs, and Behaviour 13
3.1 Understanding of the Problem Situation 13
3.2 Modelling Objectives 13
3.3 General Objectives 14
3.4 Defining Output Responses 14
3.5 Defining Input Factors 15
3.6 Identifying Entities 15
3.7 Drawing Behavioural Paths 16

4 Data, Structure, and Logic 21
4.1 Define the Data 21
4.2 Define the Structure 21
4.3 Define the Logic 25
4.4 Assumptions and Simplifications 28

III Jaamsim 30

5 Setting Up VSCode and Java 31

6 Setting Up JaamSim and HCCM 33
6.1 Prerequisites 33
6.2 Create the Project Folder Structure 33
6.3 Clone HCCM into the project folder 33
6.4 Create files to load HCCM and customised components . . 34
6.5 Create a VSCode Java Project 34

6.6 Configure Source Folders 34
6.7 Configure JDK 35
6.8 Configure Libraries 35
6.9 Integrate with JaamSim 35
6.10 Run Custom JaamSim 35
6.11 Running an HCCM Model 36

7 Radiology Clinic 37
7.1 Experiments 37
7.2 Jaamsim Model 37
7.2.1 Creating Model Objects 37
7.2.2 Configuring Objects 40
7.3 Model Logic – Java 42
7.4 Model Output 46
7.5 Task 49

8 Extended Radiology Clinic 50
8.1 Experiments 50
8.2 Jaamsim Model 50
8.3 Task 55

9 Using Traces and Scenarios 56
9.1 Jaamsim Model 56
9.2 Creating an Executable JAR File 58

IV Missing Data 60

10 Imputation Lab 61
10.1 Complete Data 62
10.2 Mean Imputation 62
10.3 Hotdeck Imputation 62
10.4 Bootstrap Replication 63
10.5 Multiple Imputation 64

V Conceptual Models 66

11 Output Buffering 67
11.1 Data 67
11.2 Components 68
11.3 Activity Diagrams 70
11.4 Logic 71

12 Health Clinic 72
12.1 Data 72
12.2 Components 73
12.3 Activity Diagrams 76
12.4 Logic 77

13 Radiology Clinic 78
13.1 Data 78
13.2 Components 79
13.3 Activity Diagrams 81
13.4 Logic 82

14 Extended Radiology Clinic 83
14.1 Data 83
14.2 Components 84
14.3 Activity Diagrams 87

14.4 Logic 88

Preface

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

Part I

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.

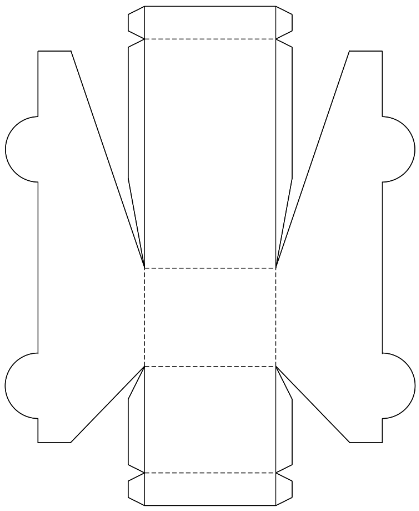

Figure 1.1: The Net Used to Make Paper Cars

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.

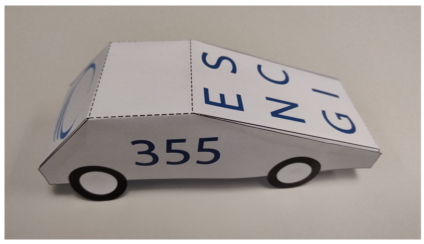

Figure 1.2: A Completed Car

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.

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?

Part II

Conceptual Modelling

2 HCCM Framework

This chapter describes the Hierarchical Control Conceptual Modelling (HCCM) framework which is used 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 and using this chapter as guidance, over the next two labs you will work through the phases for HCCM modelling shown in Figure 2.1 and complete templates for those steps. In the next lab you will complete phases 1, 2, 3, and start phase 4. The remainder of phase 4 will be completed in the lab after that. Chapter 1 provides a partially completed conceptual model of the car making system that you can use as a starting point.

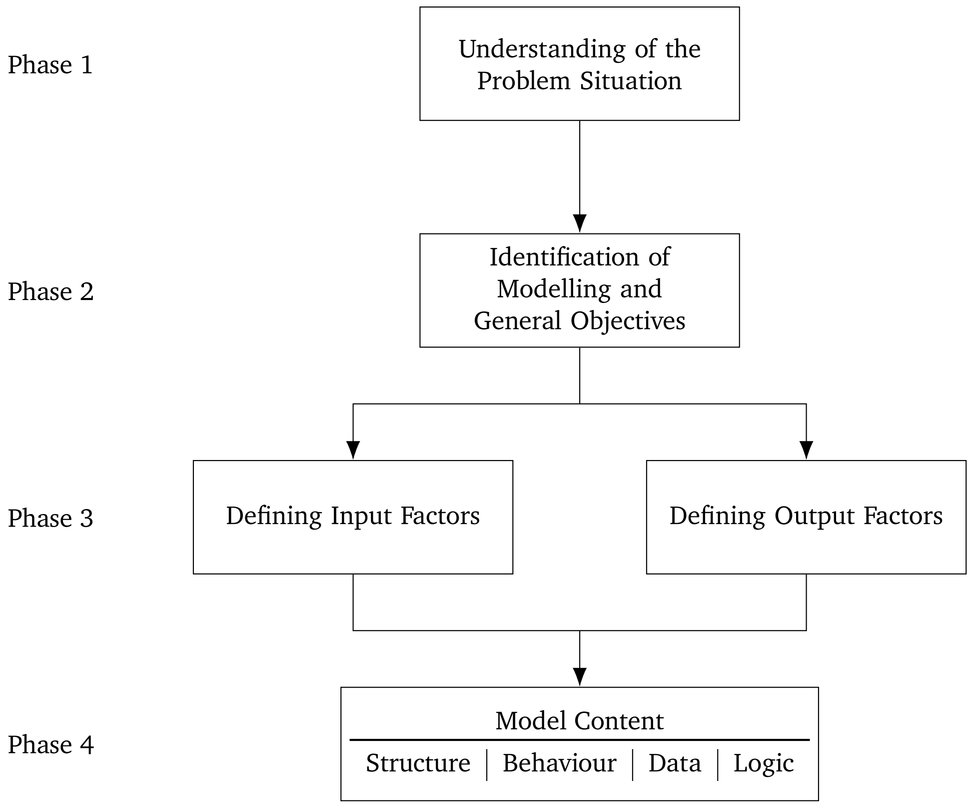

Figure 2.1: Conceptual Modelling Phases

2.1 Understanding of the Problem Situation

In Phase 1, in order 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

For Phase 2, 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

Phase 3 includes defining both the output responses and input factors. You can do these in either order, but it can often be useful to define the output responses first, as it may help you think about what inputs will influence the outputs.

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

In Phase 4 the model content is defined. There is no strict order in which you need to complete the four components (structre, behaviour, data, and logic). A possible approach, that we will take in this lab, is to:

1. Identify the entities;
2. Draw the behavioural paths;
3. Define the data;
4. Define the structure (including the entities again);
5. Define the logic.

Using this approach you may still find yourself deciding to add/remove parts that you have already defined. This is a normal part of the conceptual modelling process, and you need to go back to the part of the process you want to change – for example adding and entity or activity – and then update the rest of the CM.

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 `Modules > 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. Although it is possible to revisit and add/remove entities later, at this stage you want to consider the whole system carefully, as it is easier to include/exclude an entity now than to change it later.

2.5.2 Drawing Behavioural Paths

Once preliminary identification of entities 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.

2.5.3 Define the Data

The data for the conceptual model includes both variables, and data modules. Variables can change their value throughout the simulation and are generally used to store some information temporarily before it is required later in the simulation. Data modules contain the information that is needed to perform the simulation and can be collected beforehand. Data mocules can also represent the input/experimental factors – the things that may change between different simulation scenarios. For each data module the following information should be given:

1. The name of the data module;
2. The source of the data, where the information was obtained;
3. The way the data is modelled, is it represented by a constant, a distribution, etc.
4. Whether the output is deterministic or stochastic;
5. The inputs that the module requires;
6. The quantity that the module outputs.

When presenting a conceptual model is useful to put the data first, as it is often referenced throughout the rest of the conceptual model.

2.5.4 Define the Structure

To define the structure we start with formally defining the entities by listing them along with any attributes that they have. Some common attributes, such as ID number and the activity that the entity is currently participating in, are often omitted to avoid repitition. Attributes are usually included either to assist with the system behaviour – for example record whether a patient has had a test – or to capture the perfomance of the system – how long something has waited for.

Next we define the transitions. Each arrow on a behavioural diagram corresponds to a transition. We can collate these in a table describing: the entity that is performing the transition, and the events that the entity transitions from and to. You can simply number them starting from 1, or adopt a convention of using the entity's initial as a prefix.

Once the transitions have been defined we can define the activities and events. Usually these are presented in two tables, one for the activities and one for the events. For each event (either standalone or as part of an activity) the table should include:

1. The participant(s);
2. The type – either scheduled or controlled;
3. The state changes that occur when the event happens.

The main things that occur in state changes are:

- Schedule an end event – usually in the start event of an activity with a scheduled end event;
- Starting another activity/event – this usually happens in a scheduled end event where an entity is transitioning to another scheduled event;
- Trigger some logic – often in the start event of an activity with a controlled end event.

The simulation start event, and arrive events are often more complicated and involve scheduling the initial events and creating entities.

2.5.5 Define the Logic

The final part of the conceptual model content is the logic. Each trigger that you drew in a behvioural path (the red squares) should correspond first to a trigger statement in the state changes of an event, and a piece of logic defined here. These pieces of logic are used to determine how the system behaves – what activity an entity should do next. It is common to have logic control the behaviour when one entity needs to wait for another, as when the first entity arrives it needs to check whether the other is free to perform an activity with it. The logic is usually presented as pseudocode, alongside the entity that triggers the logic.

2.6 Assumptions and Simplifications

Throughout the four phases of the HCCM framework you should document the assumptions and simplifications that you make. Assumptions are related to uncertainties about the system being modelled, and are used to fill in gaps in the information that is required for the simulation. Simplifications are changes that are made to the model to make it easier to define or implement.

3 Inputs, Outputs, and Behaviour

In this lab you will work with your group to complete the first three Phases of the HCCM framework, and part of the fourth, as outlined in Chapter 2. The goal is for you to gain experience using the HCCM framework, and understand the initial steps in the process. To get the lab signed off your group will need to complete the:

- understanding of the problem situation;
- modelling objectives;
- general objectives;
- output responses;
- input factors;
- entities;
- and behavioural paths.

3.1 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.

3.2 Modelling Objectives

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

3.3 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?

3.4 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?

3.5 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?

3.6 Identifying Entities

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

3.7 Drawing Behavioural Paths

The activity diagrams for the pencil & template, and scissors are given below in Figures 3.1, and 3.2.

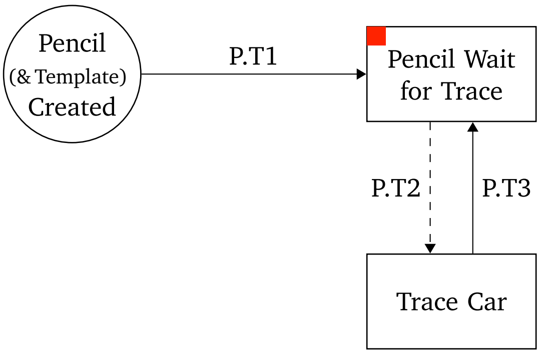

Figure 3.1: Pencil Activity Diagram

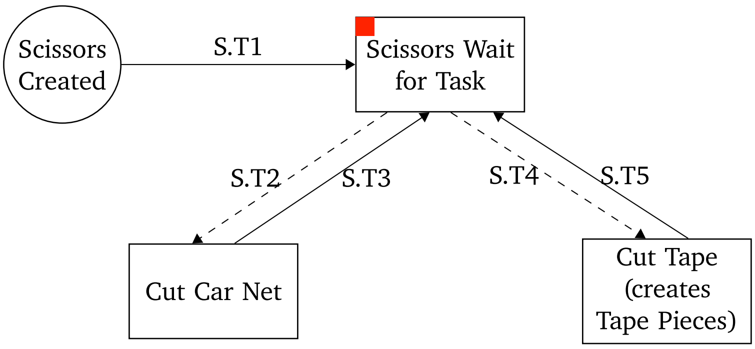

Figure 3.2: Scissors Activity Diagram

In the boxes below draw the activity diagrams for the remaining entities. Once you have completed all of the behavioural diagrams get all of your work checked by a lab tutor.

4 Data, Structure, and Logic

In this lab you will complete the remainder of the fourth phase of the HCCM framework, with your group. Again the goal is to give experience using the HCCM framework, and understand how to formulate the detailed components of a conceptual model. To get the lab signed off your group will need to complete the:

- data modules;
- formal definition of entities;
- transitions;
- activities;
- events;
- logic;
- and both assumptions and simplifications.

4.1 Define the Data

Firstly, you need to give detailed definitions of the data modules. You may not have collected data during car making, but complete the following table that describes the kind of data you would need to collect to simulate car making. Also add a comment on how the entry for CutTapeDuration would change if no person-by-person data was available, but an Exponential distribution that estimated the time it takes for a person to cut tape was available.

Table 4.1: List of Data Modules

Name	Source	Model	Type	Input	Output
NumPencils	System info	Constant	Deterministic	-	The number of pencils available
NumScissors	System info	Constant	Deterministic	-	The number of scissors available
NumTape	System info	Constant	Deterministic	-	The number of rolls of tape available
	System info	Constant	Deterministic	-	
TraceCarDur	Experimental data	Lookup	Deterministic	Person	Time to trace car
CutNetDur	Experimental data	Lookup	Deterministic	Person	Time to cut the net out
CutTapeDur	Experimental data	Lookup	Deterministic	Person	Time to cut a piece of tape

4.2 Define the Structure

The first part of the structure to define is the entities. Table 4.2 lists the entities again, but adds attributes that the entities will need to capture the

Table 4.2: List of Entities

Entity	Attributes
Paper	WaitForTrace[0.0] WaitForCutShape[0.0] WaitForFold[0.0] WaitForTapeCube[0.0]
Pencil	WaitForTrace[0.0]
Scissors	WaitForTask[0.0]
Tape	WaitForCut[0.0]
TapePieces	WaitForTape[0.0] ArrivalTime[0.0] LeavingTime[0.0]
Person	WaitForTask[0.0]

performance of the system, e.g., waiting time until the cube was cut. It is assumed that all entities have the three attributes: ID, CurrentActivity, and CurrentStart. These are omitted in the table to prevent repetition.

The next part of the structure is the transitions, which describe how entities move between activities and events. Table 4.3 lists the transitions for making paper cars. These transitions are prefixed by entity of the behavioural pathway they come from. Complete the transitions for the Scissors pathway.

Table 4.3: List of Transitions

Participant	Name	From Event	To Event
Paper	P.1	Paper Created	Paper Wait for Trace.Start
	P.2	Paper Wait for Trace.End	Trace Car.Start
	P.3	Trace Car.End	Paper Wait for Cut Net.Start
	P.4	Paper Wait for Cut Net.End	Cut Car Net.Start
	P.5	Cut Car Net.End	Paper Wait for Fold.Start
	P.6	Paper Wait for Fold.End	Fold Car.Start
	P.7	Fold Car.End	Paper Wait for Tape.Start
	P.8	Paper Wait for Tape.End	Tape Car.Start
	P.9	Tape Car.End	Car Finished
Pencil	N.1	Pencil/Template Created	Pencil Wait for Trace.Start
	N.2	Pencil Wait for Trace.End	Trace Car.Start
	N.3	Trace Car.End	Pencil Wait for Trace.Start
Scissors	S.1	Scissors Created	
	S.2		
	S.3		
	S.4		
	S.5		
Tape	T.1	Tape Created	Tape Wait for Cut.Start
	T.2	Tape Wait for Cut.End	Cut Tape.Start
	T.3	Cut Tape.End	Tape Wait for Cut.Start
Tape Piece	TP.1	Tape Pieces Created	Tape Pieces Wait for Tape.Start

Table 4.3: List of Transitions

Participant	Name	From Event	To Event
	TP.2	Tape Pieces Wait for Tape.End	Tape Car.Start
	TP.3	Tape Car.End	Tape Pieces Leave
Person	H.1	Person Created	Person Wait for Task.Start
	H.2	Person Wait for Task.End	Trace Car.Start
	H.3	Trace Car.End	Person Wait for Task.Start
	H.4	Person Wait for Task.End	Cut Car Net.Start
	H.5	Cut Car Net.End	Person Wait for Task.Start
	H.6	Person Wait for Task.End	Fold Car.Start
	H.7	Fold Car.End	Person Wait for Task.Start
	H.8	Person Wait for Task.End	Cut Tape.Start
	H.9	Cut Tape.End	Person Wait for Task.Start
	H.10	Person Wait for Task.End	Tape Car.Start
	H.11	Tape Car.End	Person Wait for Task.Start

Table 4.4 lists the activities from the behavioural pathway diagrams along with the state changes for the start and end event of each activity. Complete the activities for:

- Paper Wait for Tape Car
- Tape Car
- Person Wait for Task (*Hint* look at Scissors Wait for Task)

Table 4.4: Activities

Activity	Participants	Event	Type	State Change

Paper Wait for Trace	Paper (P)	Start	Scheduled	`(Default, omitted hereafter) P.CurrentActivity = "this activity"`
				`(Default, omitted hereafter) P.CurrentStart = TIME`
				`TRIGGER OnStartPaperWaitForTrace WITH C`

		End	Controlled	`P.WaitForTrace = TIME - P.CurrentStart` `# TRANSITION P.2 in logic`
Trace Car	Paper (P), Person (H), Pencil (N)	Start	Controlled	`SCHEDULE END at TIME + TraceCube(H)`

		End	Scheduled	`START Paper Wait for Cut Net WITH P # TRANSITION P.3`
				`START Person Wait for Task WITH H # TRANSITION H.3`
				`START Pencil Wait for Trace WITH N # TRANSITION N.3`

Paper Wait for Cut Net		Start		`TRIGGER OnStartPaperWaitForCutNet WITH P`
		End		`P.WaitForCutNet = TIME - P.CurrentStart` `# TRANSITION P.4 in logic`
Cut Car Net	Paper (P), Person (H), Scissors (S)	Start	Controlled	`SCHEDULE END at TIME + CutNet(H)`

		End	Scheduled	`START Paper Wait for Fold WITH P # TRANSITION P.5`
				`START Person Wait for Task WITH H # TRANSITION H.5`
				`START Scissors Wait for Task WITH S # TRANSITION S.3`

Paper Wait for Fold	Paper (P)	Start	Scheduled	`TRIGGER OnStartPaperWaitForFold WITH P`

Table 4.4: Activities

Activity	Participants	Event	Type	State Change
		End	Controlled	`P.WaitForFold = TIME - P.CurrentStart` `# TRANSITION P.6 in logic`
Fold Car	Paper (P), Person (H)	Start	Controlled	`SCHEDULE END at TIME + FoldCar(H)`
		End	Scheduled	`START Paper Wait for Tape Car WITH P # TRANSITION P.7` `START Person Wait for Task WITH H # TRANSITION H.7`
Paper Wait for Tape Car		Start		
		End		
Tape Car		Start		
		End		
Pencil Wait for Trace	Pencil (N)	Start	Scheduled	`TRIGGER OnStartPencilWaitForTrace WITH N`
		End	Controlled	`N.WaitForTrace = N.WaitForTrace + TIME - N.CurrentStart` `# TRANSITION N.2 in logic`
Scissors Wait for Task	Scissors (S)	Start	Scheduled	`TRIGGER OnStartScissorsWaitForTask WITH S`
		End	Controlled	`S.WaitForTask = S.WaitForTask + TIME - S.CurrentStart` `# TRANSITION S.2 or S.4 in logic`
Cut Tape	Tape (T), Person (H), Scissors (S)	Start	Controlled	`SCHEDULE END at TIME + CutTape(H)`

		End	Scheduled	`START Person Wait for Task WITH H # TRANSITION H.9`
				`START Scissors Wait for Task WITH S # TRANSITION S.5`
				`START Tape Wait for Cut WITH T # TRANSITION T.3`
				`CREATE Tape Pieces TP`
				`START Tape Pieces Created WITH TP`

Tape Wait for Cut	Tape (T)	Start	Scheduled	`TRIGGER OnStartTapeWaitForCut WITH T`
		End	Controlled	`T.WaitForCut = T.WaitForCut + TIME - T.CurrentStart` `# TRANSITION T.2 in logic`
Tape Pieces Wait for Tape	Tape Pieces (TP)	Start	Scheduled	`TRIGGER OnStartTapePiecesWaitForTape WITH TP`
		End	Controlled	`TP.WaitForTape = TP.WaitForTape + TIME - TP.CurrentStart` `# TRANSITION TP.2 in logic`
Person Wait for Task		Start		
		End		

Table 4.5 lists the events to start and finish the simulation along with the events from the behavioural pathway diagrams along with the state changes for each event. Complete the events for:

- Tape Pieces Created
- Person Created

Table 4.5: Events

Event	Participants	Type	State Change

Simulation Start	None	Scheduled	`FOR NumPaper DO`
			`  CREATE Paper P`
			`  START Paper Created WITH P`
			`END FOR`
			`FOR NumPencils DO`
			`  CREATE Pencil N`
			`  START Pencil/Template Created WITH N`
			`END FOR`
			`FOR NumScissors DO`
			`  CREATE Scissors S`
			`  START Scissors Created WITH S`
			`END FOR`
			`FOR NumTape DO`
			`  CREATE Tape T`
			`  START Tape Created WITH T`
			`END FOR`
			`FOR NumPeople DO`
			`  CREATE Person H`
			`  START Person Created WITH H`
			`END FOR`

Paper Created	Paper (P)	Scheduled	`START Paper Wait for Trace WITH P # TRANSITION P.1`
Car Finished	Paper (P)	Scheduled	`Calculate statistics for P` `REMOVE Paper P`
Pencil/ Template Created	Pencil (N)	Scheduled	`START Pencil Wait for Trace WITH N # TRANSITION N.1`
Scissors Created	Scissors (S)	Scheduled	`START Scissors Wait for Task WITH S # TRANSITION S.1`
Tape Created	Tape (T)	Scheduled	`START Tape Wait for Cut WITH T # TRANSITION T.1`
Tape Pieces Created			
Tape Pieces Leave	Tape Pieces (TP)	Scheduled	`Calculate statistics for TP` `REMOVE Tape Pieces TP`
Person Created			
Simulation Finish	None	Scheduled	`Calculate statistics as required for Pencils, Scissors, Tape , Person entities`

4.3 Define the Logic

The last part of the structure to define is the logic. You need to define the logic for each of the triggers (the red squares in the behavioural paths, and TRIGGER statements in the state changes). Tables 4.6, to 4.14 show the logic for some of the triggers. Complete the logic for: - OnStartTapeWaitForCut - OnStartCubeWaitForFold - the last condition of OnStartPersonWaitForTask

Table 4.6: OnStartPencilWaitForTrace

Triggered by: Pencil N

```
IF (any Paper P with P.CurrentActivity = Paper Wait for Trace) AND
   (any Person H with H.CurrentActivity = Person Wait for Task) THEN
  SELECT valid Paper P
  SELECT valid Person H
  START Trace Car WITH P, H, N
END IF
```

Table 4.7: OnStartScissorsWaitForTask

Triggered by: Scissors S

```
IF (any Paper P with P.CurrentActivity = Paper Wait for Cut Net) AND
   (any Person H with H.CurrentActivity = Person Wait for Task) THEN
  SELECT valid Paper P
  SELECT valid Person H
  START Cut Net WITH P, H, S
ELSE IF (any Tape T with T.CurrentActivity = Tape Wait for Cut) AND
        (any Person H with H.CurrentActivity = Person Wait for Task) THEN
  SELECT valid Tape T
  SELECT valid Person H
  START Cut Tape WITH T, H, S
END IF
```

Table 4.8: OnStartTapeWaitForCut

Triggered by: Tape T

```

```

Table 4.9: OnStartTapePiecesWaitForTape

Triggered by: Tape Pieces TP

```
IF (any Paper P with P.CurrentActivity = Paper Wait for Tape) AND
   (any Person H with H.CurrentActivity = Person Wait for Task) THEN
  SELECT valid Paper P
  SELECT valid Person H
  START Tape Car WITH P, H, TP
END IF
```

Table 4.10: OnStartPaperWaitForTrace

Triggered by: Paper P

```
IF (any Pencil N with N.CurrentActivity = Pencil Wait for Trace) AND
   (any Person H with H.CurrentActivity = Person Wait for Task) THEN
  SELECT valid Pencil N
  SELECT valid Person H
  START Trace Car WITH P, H, N
END IF
```

Table 4.11: OnStartPaperWaitForCut

Triggered by: Paper P

```
IF (any Scissors S with S.CurrentActivity = Scissors Wait for Task) AND
   (any Person H with H.CurrentActivity = Person Wait for Task) THEN
  SELECT valid Paper P
  SELECT valid Person H
  START Cut Car WITH P, H, S
END IF
```

Table 4.12: OnStartPaperWaitForFold

Triggered by: Paper P

```

```

Table 4.13: OnStartPaperWaitForTape

Triggered by: Paper P

```
IF (any Tape Pieces TP with TP.CurrentActivity = Tape Pieces Wait for Tape) AND
   (any Person H with H.CurrentActivity = Person Wait for Task) THEN
  SELECT valid Tape Pieces TP
  SELECT valid Person H
  START Tape Car WITH P, H, TP
END IF
```

Table 4.14: OnStartPersonWaitForTask

Triggered by: Person H

```
# Prioritise taping, then folding, then cutting, then tracing
IF (any Paper P with P.CurrentActivity = Paper Wait for Tape Car) AND
   (any Tape Pieces with TP.CurrentActivity = Tape Pieces Wait for Tape) THEN
  SELECT valid Paper P
  SELECT valid Tape Pieces TP
  START Tape Car WITH P, H, TP
ELSE IF (any Paper P with P.CurrentActivity = Paper Wait for Tape Car) AND
        (any Tape T with T.CurrentActivity = Tape Wait for Cut) AND
        (any Scissors S with S.CurrentActivity = Scissors Wait for Task) THEN
  # There is a car waiting to be taped, but no tape pieces
  SELECT valid Tape T
  SELECT valid Scissors S
  START Cut Tape WITH T, H, S
ELSE IF (any Paper P with P.CurrentActivity = Paper Wait for Fold)
  SELECT valid Paper P
  START Fold Car WITH P, H
ELSE IF (any Paper P with P.CurrentActivity = Paper Wait for Cut Net) AND
        (any Scissors with S.CurrentActivity = Scissors Wait for Task) THEN
  SELECT valid Paper P
  SELECT valid Scissors S
  START Cut Car Net WITH P, H, S






END IF
```

4.4 Assumptions and Simplifications

What assumptions did you make when defining your model? What simplifications did you make? These details are important when communicating your model to others. List them in the box below. Remember, assumptions are beliefs about the system that we take to be true, and inform part of the model. Assumptions enable us to resolve uncertainties about the actual system. We use assumptions complete parts of the model that we do not have the information about. Simplifications are intentional changes to the model that make it different from the system being modelled. Simplifications are usually used to replace a complex part (or parts) with a simpler one, that is either more easily: defined in the conceptual model; implemented in the simulation; or communicated.

You have now completed the conceptual model of the cube making activity and you could use this model as the staring point for implementing a simulation model in JaamSim using the HCCM library. Show your work to a lab tutor to get it signed off.

Part III

Jaamsim

5 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. You don't need to get this lab signed off, but you will need to have VSCode and the extensions installed to complete the next few labs.

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 25. You can download it from here, and then install it. If you are using a lab computer we will use a version of Java that is already installed, which is Amazon Corretto 25.

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

First we will set up the folder structure that you will use to contain the files, not just for this lab but all the ENGSCI 355 labs. Create a new folder called **ENGSCI355**, if you are using a lab computer create this folder in the **Documents** folder on your University OneDrive. By keeping all of the 355 files on your OneDrive you should be able to access them on any University computer. If you are using your own computer, then create it within Documents or wherever you usually keep University related work. Inside the **ENGSCI355** folder create another folder called **Java_Example**.

Once you have created the folders start VSCode. The first thing we will do in VSCode is create a profile for 355. A VSCode profile allows you to save settings and transfer them when you use VSCode on a different computer. This is useful if you use a different lab computer, or use a lab computer then your personal computer. We will use it mainly to save the extensions we need to run Java programs.

To create a profile click on the Settings icon in the bottom left corner of the screen, then select Profiles. In the window that opens click on the **New Profile** button. Enter **ENGSCI355** for the profile name, then scroll down and click Create. Once the profile has been created we need to activate it by clicking on the tick next to its name. With the profile active we need to install the extensions that we need for Java. On the left-hand side click on the Extensions tab. Search for the 'Extension Pack for Java' and install it. Once it is installed, go back to the Profiles page by clicking on the name of the tab or by clicking on the Settings icon again and selecting Profiles. Then click on the three dots to the right of the **ENGSCI355** profile, an select **Export** from the pop-up menu. From the export options provided select "Local file". Then browse to the ENGSCI355 folder on your OneDrive you create earlier and click Save. You should now have a ENGSCI355.code-profile file saved on your OneDrive that includes the extensions needed to run Java programs (like Jaamsim). This means that if you work on a different computer in the future you can load this profile by clicking on the Settings icon, selecting Profiles, clicking the arrow next to the **New Profile** button, selecting **Import Profile**, and browsing to this ENGSCI355.code-profile file.

Then in VSCode open the folder you created earlier 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 25 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:

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

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

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

6 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. The goal is for you to be able to run an HCCM Jaamsim model and begin to understand how the Java code, autoload and include files, and the cfg file are related. To get this lab signed off you will need to show the demonstration model running.

6.1 Prerequisites

These instructions were prepared using:

1. Git – 2.47.0.2;
2. GitHub Desktop – 3.4.8 (x64);
3. Java – Amazon Corretto 25;
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 25 is available on the University of Auckland's Software Centre.

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

6.2 Create the Project Folder Structure

Create a new folder on your University OneDrive called ENGSCI355, if you have not done so already. Then create two folders within this one, called **sim**, and **labs**. 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.

6.3 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, and click Clone. 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
```

6.4 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 6.1.

```
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 6.1: 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**.

6.5 Create a VSCode Java Project

In VSCode use File → Open Folder to open the **sim** folder. In the File Explorer open some folders 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.

6.6 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

6.7 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-17 is selected and then click 'Apply Settings'.

6.8 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'.

6.9 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'.

6.10 Run Custom JaamSim

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

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

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

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

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

6.11 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, ssq.zip, from Canvas under Lab 4 in the Java, Jaamsim, and HCCM module. Then, 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.

Show this model running to a lab tutor to get your lab signed off.

7 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 13.

To have your lab signed off you need to show that you have built the simulation model, can run it, and the output matches the provided results.

7.1 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.

7.2 Jaamsim Model

7.2.1 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 7.1. Then in the Object Selector window select ActiveEntity1, press F2, and rename it **PatientEntity**. Alternatively, you can change the name by editing the value of the 'Name' Keyword, which is the top entry in the 'Key Inputs' tab.

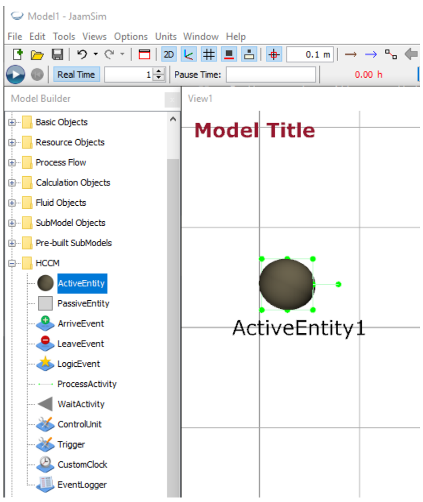

Table 7.1: Arrival Distribution Inputs

Object	Keyword	Value
ArrivalDistribution	UnitType	TimeUnit
	RandomSeed	1
	Mean	0.125 h

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 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 7.1 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 7.2:

Table 7.2: 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

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 7.3 in your simulation.

Once you have created all of these objects lay them out similarly to as shown in Figure 7.2.

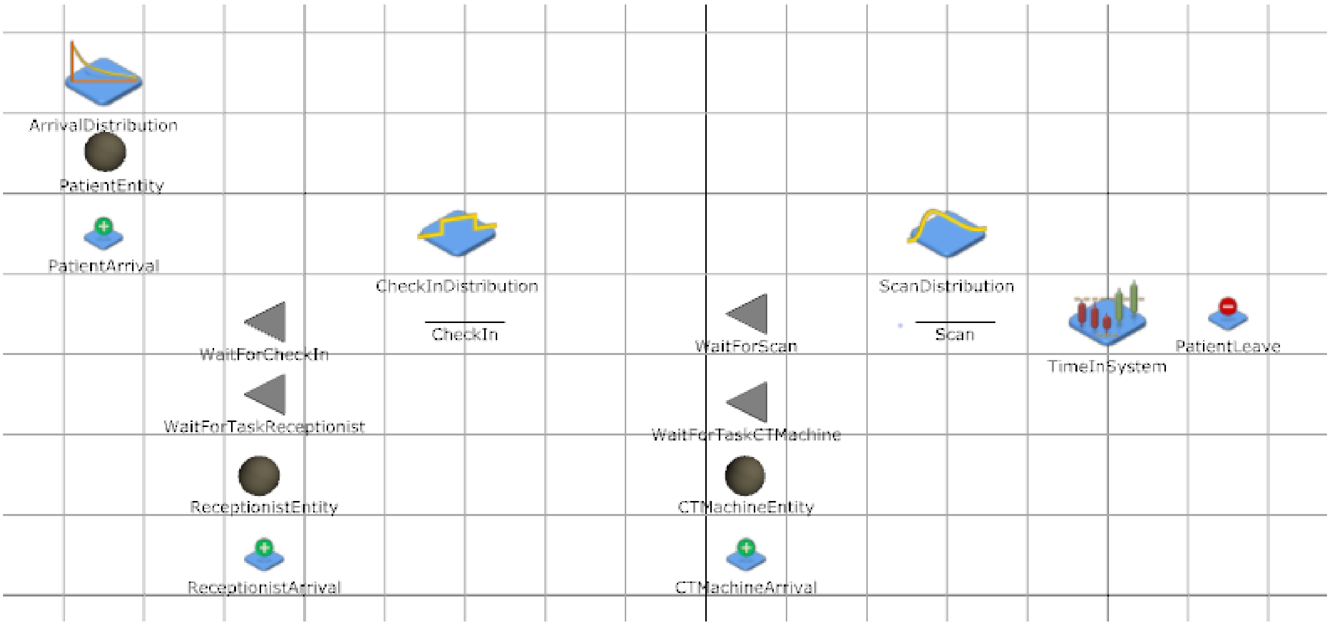

Figure 7.2: Screenshot of Simulation Model Layout

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 in the Lab 5: Radiology Clinic Assignment 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.

Table 7.3: 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

7.2.2 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 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

Table 7.4: 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 7.5: 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

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 8.7, 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.

Table 7.6: Collecting Statistics

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

7.3 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.

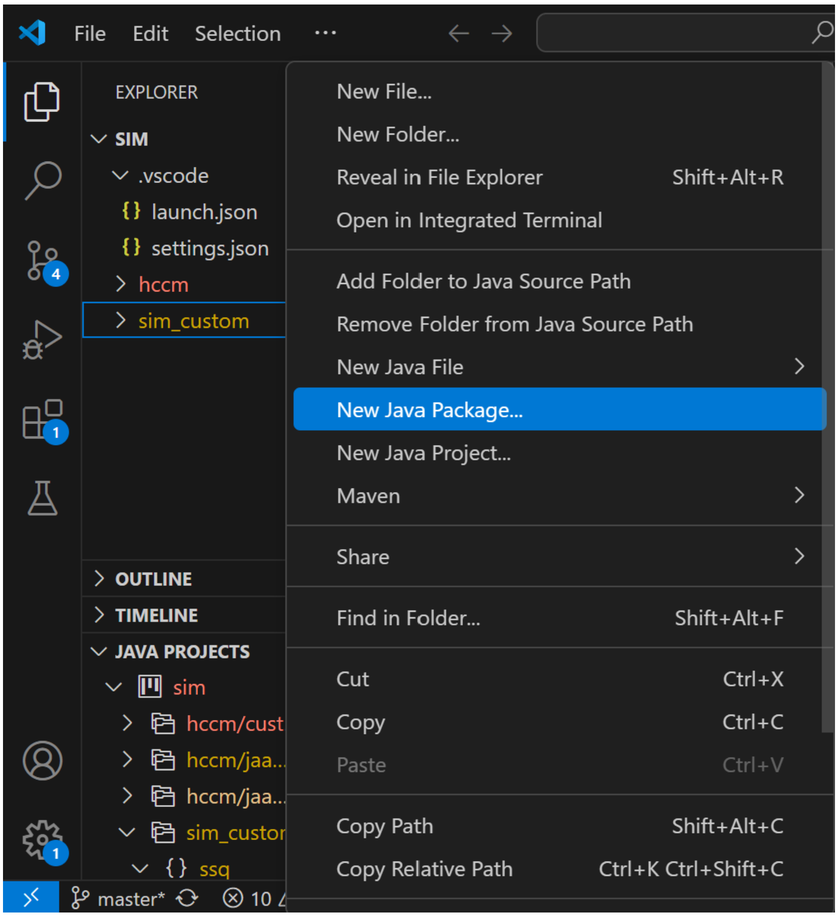

Figure 7.3: First Step in Creating a New Package

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.

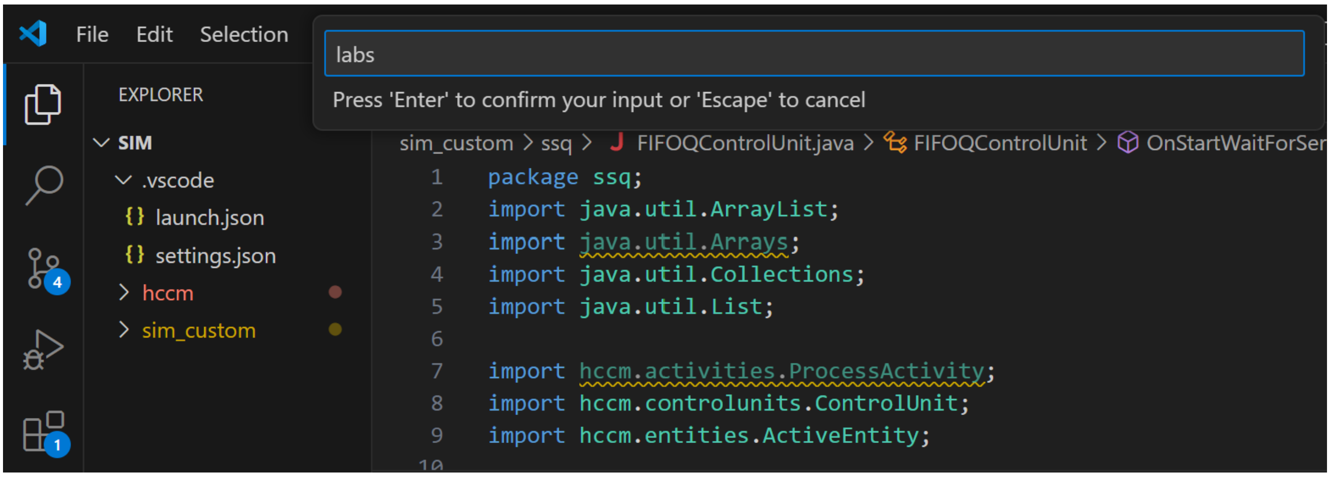

Figure 7.4: Second Step in Creating a New Package

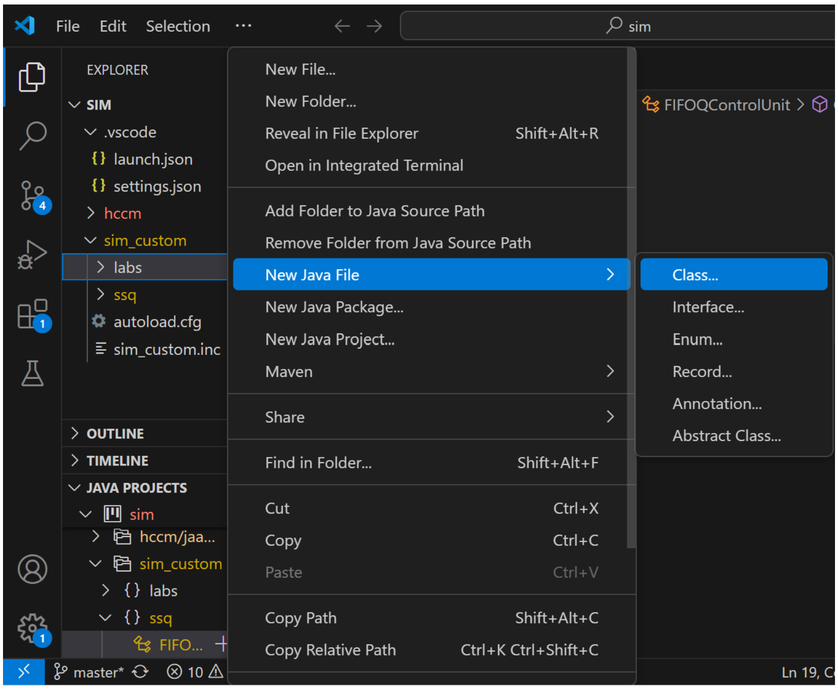

Figure 7.5: First Step in Creating a New Class

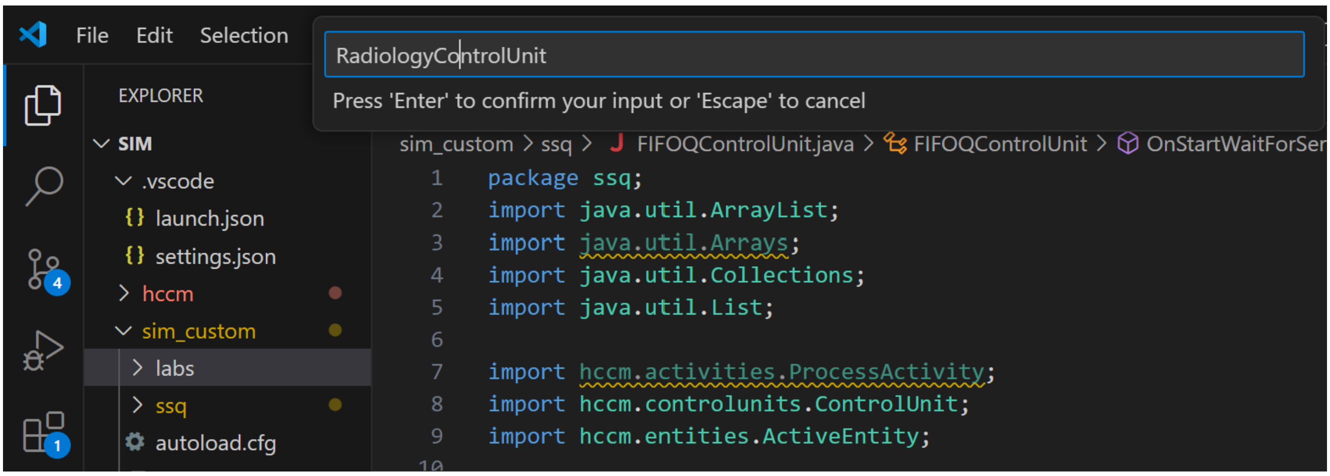

Figure 7.6: Second Step in Creating a New Class

Once you have created the new class, an almost-empty file called Radiology-ControlUnit.java should be created. You need to add `extends ControlUnit` to the line that declares the class, and also import the ControlUnit class with `import hccm.controlunits.ControlUnit;`. At this stage the file should look like this:

```java
package labs;

import hccm.controlunits.ControlUnit;

public class RadiologyControlUnit extends ControlUnit{

}
```

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 on Canvas in the Lab 5: Radiology Clinic Assignment 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 {
    ↪   ControllerIconModel } IconFile {
    ↪   '<res>/images/Controller-24.png' } DefaultSize { 0.5 0.5
    ↪   0.5 m }
10 RadiologyControlUnit JavaClass { labs.RadiologyControlUnit }
    ↪   Palette { 'Custom Logic' } DefaultDisplayModel {
    ↪   AssembleIconModel } IconFile {
    ↪   '<res>/images/Assemble-24.png' } DefaultSize { 0.5 0.5
    ↪   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:

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 on Canvas in the Lab 5: Radiology Clinic Assignment.

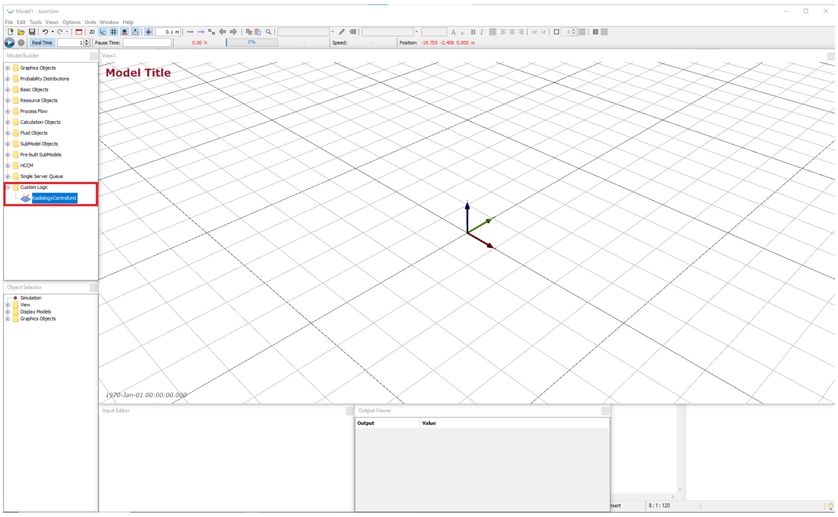

Figure 7.7: Screenshot of Control Unit Object

```
package labs;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

import hccm.activities.ProcessActivity;
import hccm.controlunits.ControlUnit;
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.

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

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

    ArrayList<ActiveEntity> idleReceps = this.getEntitiesInActivity("ReceptionistEntity",
     ↪  "WaitForTaskReceptionist", simTime);
    ActivityStartCompare actSartComp = this.new ActivityStartCompare();

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

      ActiveEntity patient = ents.get(0);
      ActiveEntity receptionist = idleReceps.get(0);

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

Table 7.7: 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

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.

7.4 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

```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", "WaitForCheckIn", simTime);
  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", "WaitForScan", simTime);
  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 7.8: 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 7.9: 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

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 (on Canvas in the Lab 5: Radiology Clinic Assignment), 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
```

7.5 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?

Once your simulation is working and you are getting the correct results, you can get your lab signed off.

8 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 Chapter 14. To have your lab signed off you need to show that you have built the simulation model, can run it, and the output matches the provided results.

8.1 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 8.1:

Table 8.1: Patient Priority Proportions

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

8.2 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 Table 8.2 shows how to create the priority attribute and make the default value 0.

Table 8.2: 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 occuring. Create a DiscreteDistribution object called PriorityDistribution with the values shown in Table 8.3.

Table 8.3: 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 8.4: Assign Priority

Object	Keyword	Value
PatientArrival	AssignmentList	{ 'this.CurrentParticipants(1).priority = [PriorityDistribution].Value' }

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

Table 8.5: 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 8.6: 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. Also set the participant.

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 RadiologyExtended-ControlUnit. 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.

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

  ArrayList<ActiveEntity> waitPats = this.getEntitiesInActivity("PatientEntity", "WaitForScan",
  ↪  simTime);
  AttributeCompare priorityComp = new AttributeCompare("priority");
  ActivityStartCompare actStartComp = this.new ActivityStartCompare();
  ActiveEntity ct = ents.get(0);
  int reqMaintenance = (int) getNumAttribute(ct, "NeedMaintenance", simTime, -1);

  if (reqMaintenance == 1) {
    // A //
  }

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

    int highestPriority = // B //

    ArrayList<ActiveEntity> priorityPatients = new ArrayList<ActiveEntity>();
    for (ActiveEntity wP : waitPats) {
      int patPri = // C //
      if (patPri == highestPriority) {
        priorityPatients.add(wP);
      }
    }

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

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

Then create two new methods called OnCTArrival and OnRequireMaintenance:

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

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

  le.scheduleEvent(ents, maintenanceTimeGap);
}

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

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

  le.scheduleEvent(ents, simTime + maintenanceTimeGap);

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

To get the new logic used in the simulation we need to add two new triggers: **StartCTArrival**, and **StartRequireMaintenance**. Set the control unit for both the triggers to be the 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 8.7, note that **TotalTime** is an output on the entity which stores the total time that the entity has been in the simulation for:

Table 8.7: 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 on Canvas under the Lab 6: Extended Radiology Clinic Assignment. Then, 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
```

8.3 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
```

Once your simulation is working and you are getting the correct results, you can get your lab signed off.

9 Using Traces and Scenarios

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

We are not considering any changes to the system, so the conceptual model is the same as for the previous lab. To have your lab signed off you need to show that you have added the data from the file to the model, exported the model as a .far file, and can run the .jar file from the command line.

9.1 Jaamsim Model

To run the simulation from a data trace we need to make some changes to the Jaamsim model. Once again create a new folder called **RC3** and copy your .cfg file (and the .png files so that the graphics work) from the previous lab folder into this folder and rename it to **radiology_lab_trace.cfg**. First, download the **RC_50_week_data.txt** file from Canvas under the Lab 7: Using Traces and Scenarios Assignment. This file contains 50 weeks of data of patients at the radiology clinic including: the time the patient arrived, the priority of the patient, the time the patient took to check in, and the time the patient took to have their scan.

Before we load the data in we will first change the starting date of the simulation, which defaults to 1970, to instead be 2024, so that the data read from the file is interpreted correctly. To do this go to the Simulation object and under the Options tab enter **2024-01-01** for the StartDate.

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

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

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

Table 9.1: Update PatientArrival

Object	Keyword	Value
PatientArrival	CustomOutputList	{ thisPatientData '[PatientData].Value(this.NumberAdded + 1)' }
		{ nextPatientData '[PatientData].Value(this.NumberAdded + 2)' }
	FirstArrivalTime	[PatientData].Value(2)(2)
	InterArrivalTime	'this.nextPatientData(2) - this.thisPatientData(2)'
	AssignmentList	{ 'this.obj.priority= this.thisPatientData(3)' }
		{ 'this.obj.checkInTime= this.thisPatientData(4)' }
		{ 'this.obj.scanTime= this.thisPatientData(5)' }

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

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

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

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

Now, suppose we are interested in the time that patients spend waiting for check in and for the scan. We can't use the current PatientLogger as it only records the total time that patients are in the system for. We could add attributes for each time that we are interested in, and assign the value when the entity gets to the relevant stage, and then use the PatientLogger to log these attributes. We can instead use an EventLogger from the HCCM palette. The EventLogger records the time that an entity starts each of the activities that it participates in. So, create an EventLogger and call it PatientEventLogger.

Then, to get the events recorded go to the PatientLeave object and under the HCCM tab enter PatientEventLogger for the EventLogger keyword. Now any entities that are sent to the patient leave will have the start times of any activities that they participated in recorded.

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

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

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

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

Now when you run the model, the number of CT Machines and Receptionists will change in each scenario.

Download and run the **RC3_Analysis.R** file, from Canvas under the Lab 7: Using Traces and Scenarios Assignment. You will have to update the directory that it reads the data from and the name of the data file used. The script splits each replication into 50 batches, each one week long, and calculates the mean across the batches and the four scenarios of the 90th percentile waiting time for both check in and scan within each of batch. No warm-up period is used, so this assumes that being empty and idle is a typical state of the system. Splitting into batches by week assumes that each week is not correlated to the preceding and following weeks. You should get the following output:

```
                              CheckInWaitTime  ScanWaitTime
CTMachineScenario ReceptionistScenario
1                 1                  0.059406      0.536305
                  2                  0.000000      0.548324
2                 1                  0.059396      0.154899
                  2                  0.000000      0.170225
```

9.2 Creating an Executable JAR File

We can package the custom Java code in the extended control unit class alongside the base Jaamsim and HCCM code into a jar file that can be run without having to set up Java/VSCode. To do this click on the right arrow that appears when you hover over the **Java Projects** tab, with the tooltip 'Export Jar'. Then in the menu that appears, select GUIFrame as the main class, then click OK with all of the options selected.

This should create a file called **sim.jar** inside the sim folder, alongside the hccm and sim_custom folders. Copy the **sim.jar** file to your folder for this lab (if you are following the structure here it is called RC3 and is in the labs folder), and rename it to RC3.jar. It doesn't really matter where you put it, but it is much easier if the .jar file is in the same directory as the .cfg file for this lab (radiology_lab_trace.cfg).

Then open a terminal in VSCode. There might be one already open, otherwise you can open one by going to **Terminal** in the top left, and selecting **New Terminal**. By default this will open a terminal in the **sim** folder. You will need to navigate to the folder with the .jar and .cfg files in it using the cd (change directory) command. If you are following the folder structure described then should go up one level (cd ..), then into labs (cd labs), then into RC3 (cd RC3).

Once you have a terminal open in the correct folder you can run the following command `java -jar RC3.jar radiology_lab_trace.cfg -h`.

This command gets java to run the RC3.jar file which then opens and runs the radiology_lab_trace.cfg model in headerless mode (denoted by the -h). Headerless mode means that the visualisation of the simulation is not shown, and allows the model to run more quickly. If it runs correctly nothing will be printed, as seen in Figure 9.1.

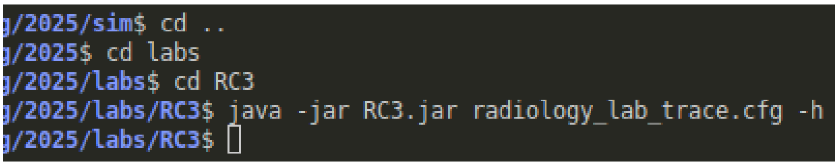

Figure 9.1: Screenshot of Running the Jar File

Once you can run the model from the command line using the .jar file (you can confirm that the model is actually running by checking the date/time modified of the output files), you can get your lab signed off.

Part IV

Missing Data

10 Imputation Lab

In this lab, we will implement some of the techniques we have looked at in class to deal with missing data. You will be using a dataset on paua, or abalone. The data set includes the following variables:

Table 10.1: Abalone Dataset Variables

Name	Data Type	Measurement Unit	Description
Type	nominal	–	M, F, and I (infant)
Length	continuous	mm	Longest shell measurement
Diameter	continuous	mm	perpendicular to length
Height	continuous	mm	with meat in shell
Whole.weight	continuous	grams	whole abalone
Shucked.weight	continuous	grams	weight of meat
Viscera.weight	continuous	grams	gut weight (after bleeding)
Shell.weight	continuous	grams	after being dried
Rings	integer	–	+1.5 gives the age in years

Start by downloading the dataset.

```
import pandas as pd
import numpy as np
from scipy import stats

# URL of the Abalone dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data"

# Column names for the dataset
column_names = ["Type", "Length", "Diameter", "Height", "Whole weight", "Shucked weight",
↪  "Viscera weight", "Shell weight", "Rings"]

# Load the dataset into a Pandas DataFrame
abalone_data = pd.read_csv(url, header=None, names=column_names)

# Display the first few rows of the dataset
print(abalone_data.head())
```

```
  Type  Length  Diameter  Height  Whole weight  Shucked weight  \
0    M   0.455     0.365   0.095        0.5140          0.2245
1    M   0.350     0.265   0.090        0.2255          0.0995
2    F   0.530     0.420   0.135        0.6770          0.2565
3    M   0.440     0.365   0.125        0.5160          0.2155
4    I   0.330     0.255   0.080        0.2050          0.0895

   Viscera weight  Shell weight  Rings
0          0.1010         0.150     15
1          0.0485         0.070      7
2          0.1415         0.210      9
3          0.1140         0.155     10
4          0.0395         0.055      7
```

We will investigate the age of the paua, so let's start by randomly dropping about 10% of the data. We will set a random seed, so that our output is repeatable for debugging purposes, and make a copy of the original data with **NAs** for 400 of the recorded values of the **Rings** variable.

```
data_missing = abalone_data.copy()
n_missing = 400
rng = np.random.default_rng(345678)
missing_ind = rng.choice(data_missing.index, size=n_missing,
  replace=False)
data_missing.loc[missing_ind, 'Rings'] = np.nan
```

10.1 Complete Data

Start by computing a 95% confidence interval for the mean age of the paua in the population using only the complete (non-missing) data (be careful to account for the relationship between age and number of rings).

```
compl_mean = data_missing['Rings'].mean() + 1.5
compl_sem = stats.sem(data_missing['Rings'].dropna())
ci = stats.norm.interval(0.95, loc=compl_mean,
  scale=compl_sem)

print(f"Complete data CI: {ci[0]:.3f}, {ci[1]:.3f}")
```

```
Complete data CI: 11.332, 11.539
```

10.2 Mean Imputation

Next, implement simple imputation to estimate the mean age, by making a copy of the Ring column in **data_missing**, replacing the **NAs** with the mean, and recomputing the 95% confidence interval. My output is below.

```
Mean imputation CI: 11.342, 11.529
```

10.3 Hotdeck Imputation

Now try using random draws, conditioning on **Type**. We can do this by looping on the paua type, and replacing the missing values for each type by an appropriately sized draw with replacement from the non-missing values for that type. Reset the seed so that you can get consistent output when re-running just this part.

```
rng = np.random.default_rng(678910)
imp_data = data_missing['Rings'].copy()
for t in data_missing['Type'].unique():
  this_missing = ***A***
  num_draws = ***B***
  non_missing = data_missing.loc[imp_data.notna() &
↪  (data_missing['Type'] == t), 'Rings']
  hot_deck = ***C***
```

```
  imp_data[this_missing] = hot_deck

hd_imp_mean = imp_data.mean() + 1.5
hd_imp_sem = stats.sem(imp_data)
ci = stats.norm.interval(0.95, loc=hd_imp_mean,
  scale=hd_imp_sem)

print(f"Hot deck: {ci[0]:.3f}, {ci[1]:.3f}")
```

Note, code fragment ***A*** produces True if the corresponding row is of type t and has missing data, fragment ***B*** returns the number of missing data for type t, and fragment ***C*** performs the appropriate draw from the non-missing data (using the function rng.choice). The output is shown below:

```
Hot deck CI: 11.346, 11.542
```

10.4 Bootstrap Replication

After implementing a couple of standard imputation techniques, let's try bootstrap replication. We will generate 200 bootstrap replicates, impute missing data via random draws conditioning on Type, and thus include the effect of missing data in our 95% CI for mean age of the population. In the code that follows, missing code fragment ***A*** samples the appropriate number of row indices with replacement, for using to build the bootstrap replicate dataframe. Fragments ***B***, ***C*** and ***D*** then repeat the hotdeck imputation steps as performed in the previous section.

```
rng = np.random.default_rng(891011)
imp_data = data_missing['Rings'].copy()
num_boots = 200
boot_reps = np.zeros(num_boots)

for i in range(num_boots):
  this_boot_inds = ***A***
  this_boot = data_missing.loc[this_boot_inds].reset_index()

  for t in this_boot['Type'].unique():
    this_missing = ***B***
    num_draws = ***C***
    non_missing = this_boot.loc[this_boot['Rings'].notna() &
 ↪  (this_boot['Type'] == t), 'Rings']
    hot_deck = ***D***
    this_boot.loc[this_missing, 'Rings'] = hot_deck

  boot_reps[i] = this_boot['Rings'].mean() + 1.5
```

To compute our confidence interval, we will use the mean of the bootstrapped estimates as our **actual estimate**, and build our confidence interval from that value using the quantiles of the bootstrap replicates, as shown in class.

```
bs_imp_mean = boot_reps.mean()

emp_ci = bs_imp_mean * 2 - np.quantile(boot_reps, [0.975,
↪  0.025])
print(f"Empirical bootstrap CI: {emp_ci[0]:.3f},
↪  {emp_ci[1]:.3f}")
```

```
Empirical bootstrap CI: 11.347, 11.562
```

Alterntaively we can build a confidence interval directly from our bootstrap replicates based on the normal distribution:

```
bs_imp_sem = stats.sem(boot_reps)
norm_ci = stats.norm.interval(0.95, loc=bs_imp_mean,
↪  scale=boot_reps.std())

print(f"Normal bootstrap CI: {norm_ci[0]:.3f},
↪  {norm_ci[1]:.3f}")
```

```
Normal bootstrap CI: 11.336, 11.561
```

10.5 Multiple Imputation

Our final approach for building a confidence interval for the mean of the variable with missing data is to implement multiple imputation. We will condition on Type, and impute 500 values for each missing value, to build a 95% CI for mean age of the population that reflects the uncertainty due to missing data.

```
rng = np.random.default_rng(101112)
n_imps = 500
mi_est = np.zeros(n_imps)
mi_var = np.zeros(n_imps)

for i in range(n_imps):
  imp_data = data_missing['Rings'].copy()
  for t in data_missing['Type'].unique():
    this_missing = ***A***
    num_draws = ***B***
    non_missing = data_missing.loc[imp_data.notna() &
↪  (data_missing['Type'] == t), 'Rings']
    hot_deck = ***C***
    imp_data[this_missing] = hot_deck

  mi_est[i] = ***D***
  mi_var[i] = ***E***
```

Note that code fragments `***A***`, `***B***` and `***C***` again repeat the hotdeck imputation steps as performed in the previous sections. Fragment `***D***` computes the mean for iteration `i`. Fragment `***E***` computes the variance of our estimator (the mean). To compute this note that:

$$\begin{aligned} Var[\overline{X}] &= Var\left[\frac{1}{n}\sum_1^n X_i\right] \\ &= \frac{1}{n^2}Var\left[\sum_1^n X_i\right] \\ &= \frac{1}{n^2}\sum_1^n Var\left[X_i\right] \\ &= \frac{1}{n^2}\sum_1^n \sigma \\ &= \frac{1}{n}\sigma^2 \end{aligned}$$

Of course we don't know σ^2 as this is a population parameter, but an unbiased estimate for it is $\frac{1}{n-1}\sum_1^n (X_i - \overline{X})^2$. We complete the analysis by computing our estimate (`***F***`), the within imputation variance (`***G***`), the between imputation variance (`***H***`), the total variance (`***I***`), the estimate of fraction of information lost due to missing data (`***J***`), and the degrees of freedom (`***K***`). We can then compute the CI width using a t distribution.

```
pt_est = ***F***
```

```
11.44127316255686
```

```
w_est= ***G***
```

```
0.0025160028417431446
```

```
b_est = ***H***
```

```
0.00020210398944415928
```

```
t_est = ***I***
```

```
0.0027185110391636205
```

```
gamma = ***J***
```

```
0.07449232116518455
```

```
df = ***K***
mi_ci = stats.t.interval(0.95, df, pt_est, np.sqrt(t_est))
```

```
Multiple imputation CI: 11.339, 11.544
```

Part V

Conceptual Models

11 Output Buffering

11.1 Data

Table 11.1: List of Global Variables

Name	Description	Initial Value
NextPacketIdNum	The Id number that will be assigned to the next packet	1
P	The set of all packets	$\emptyset$

Table 11.2: List of Data Modules

Name	Source	Model	Type	Input	Output
InterarrivalTime	Problem Description	Poisson Process	Stochastic	Mean (1 min)	Sample from distribution
PacketSize	Problem Description	Triangular Distribution	Stochastic	Min, Mode, Max	Sample from Distribution
TransmitRate	Problem Description	Constant	Deterministic	-	Value
TransmitDuration	Problem Description	Function	Deterministic	Size, Rate	Value (Size / Rate)
BufferSize	Experiment	Constant	Deterministic	-	Value

11.2 Components

Table 11.3: List of Entities

Entity	Attributes
Packet	ID CurrentActivity CurrentStart Size
Link	ID CurrentActivity CurrentStart

Table 11.4: List of Transitions

Participant	Name	From Event	To Event
Packet	P.1	Packet Arrives	Wait for Link.Start
	P.2	Wait for Link.End	Transmit.Start
	P.3	Wait for Link.End	Packet Leaves
	P.4	Transmit.End	Packet Leaves
Link	L.1	Link Created	Wait for Packet.Start
	L.2	Wait for Packet.End	Transmit.Start
	L.3	Transmit.End	Wait for Packet.Start

Table 11.5: Activities

Activity	Participants	Event	Type	State Change

Wait for Link	Packet (p)	Start	Scheduled	`P.CurrentActivity = "Wait for Link" # default`
				`P.CurrentStart = TIME # default`
				`TRIGGER OnStartWaitForLink WITH p`

		End	Controlled	
Transmit	Packet (p), Link (l)	Start	Controlled	`SCHEDULE Transmit.End at TIME + TransmitDuration (p.Size, Rate)`
		End	Scheduled	`START Packet Leaves WITH p # TRANSITION P.4` `START Wait for Packet WITH l # TRANSITION L.3`
Wait for Packet	Link (l)	Start	Scheduled	`TRIGGER OnStartWaitForPacket WITH l`
		End	Controlled	

Table 11.6: Events

Event	Participants	Type	State Change
Simulation Start	-	Scheduled	`SCHEDULE Link Created at TIME` `SCHEDULE Packet Arrives at TIME + InterarrivalTime()`

Table 11.6: Events

Event	Participants	Type	State Change

Packet Arrives	-	Scheduled	`CREATE Packet p`
			`p.ID = NextPatIDNum`
			`NextPatIDNum = NextPatIDNum + 1`
			`p.Size = PacketSize()`
			`SCHEDULE Packet Arrives at TIME + InterarrivalTime()`
			`START Wait for Transmit WITH p # TRANSITION P.1`

Packet Leaves	Packet (p)	Scheduled	`Calculate statistics for p`

Link Created	-	Scheduled	`CREATE Link l`
			`l.Conversion = Conversion`
			`START Wait for Packet WITH l # TRANSITION L.1`

Simulation Finish	-	Scheduled	`Calculate any required statistics`

11.3 Activity Diagrams

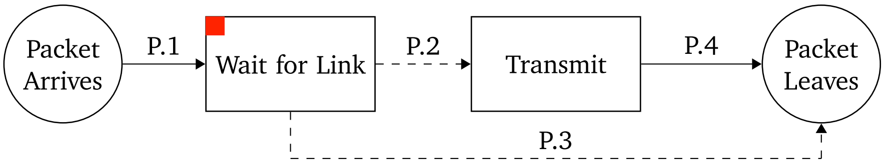

Figure 11.1: Packet Activity Diagram

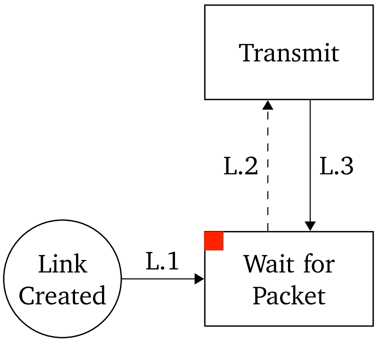

Figure 11.2: Link Activity Diagram

11.4 Logic

Table 11.7: OnStartWaitForLink

Triggered by: Packet p

```
waiting_packets = {p1 FOR p1 IN P IF p1.CurrentActivity = "Wait for Link"}
buffer_used = sum{p1.Size for p1 waiting_packets IF p1 != p}
IF L.CurrentActivity IS "Wait for Packet" THEN
    START Transmit WITH p, L # TRANSITIONS P.2, L.2
ELSE IF buffer_used + p.size > BufferSize THEN
    START Packet Leaves with p # TRANSITION P.3
END IF
```

Table 11.8: OnStartWaitForPacket

Triggered by: Link l

```
packets = {p FOR p IN P IF p.State = "Wait for Link"}
IF packets IS NOT empty THEN
    p_hat = argmin{p.CurrentStart FOR p IN packets}
    START Transmit WITH p_hat, l # TRANSITIONS P.2, L.2
END IF
```

12 Health Clinic

12.1 Data

Table 12.1: List of Global Variables

Name	Description	Initial Value
NextWalkUpIdNum	The Id number that will be assigned to the next walk-up patient	1
NextApptIdNum	The Id number that will be assigned to the next appointment patient	1
NextDoctorIdNum	The Id number that will be assigned to the next doctor	1
W	The set of all walk-up patients	$\emptyset$
A	The set of all appointment patients	$\emptyset$
D	The set of all doctors	$\emptyset$

Table 12.2: List of Data Modules

Name	Source	Model	Type	Input	Output
InterarrivalTime	Experiment	Poisson Process	Stochastic	Mean	Sample from distribution
AppointmentTimes	Experiment	Lookup	Deterministic	Appointment Number	Appointment Time
ConsultationDuration	Clinic Data	Triangular Distribution	Stochastic	Min, Mode, Max	Sample from Distribution

12.2 Components

Table 12.3: List of Entities

Entity	Attributes
Walk-up Patient	ID CurrentActivity CurrentStart ArrivalTime WaitTime
Appointment Patient	ID CurrentActivity CurrentStart AppointmentNumber AppointmentTime WaitTime
Doctor	ID CurrentActivity CurrentStart Role ConsultTime

Table 12.4: List of Transitions

Participant	Name	From Event	To Event
Walk-up Patient	W.1	Walk-up Arrives	Walk-up Wait for Consultation.Start
	W.2	Walk-up Wait for Consultation.End	Consultation.Start
	W.3	Consultation.End	Walk-up Patient Leaves
Appointment Patient	A.1	Appt. Arrives	Appt. Wait for Consultation.Start
	A.2	Appt. Wait for Consultation.End	Consultation.Start
	A.3	Consultation.End	Appt. Patient Leaves
Doctor	D.1	Doctor Created	Wait for Patient.Start
	D.2	Wait for Patient.End	Consultation.Start
	D.3	Consultation.End	Wait for Patient.Start

Table 12.5: Activities

Activity	Participants	Event	Type	State Change
Walk-up Wait for Consultation	Walk-up Patient (w)	Start	Scheduled	`TRIGGER OnStartWalkupWaitForConsultation WITH w`
		End	Controlled	`w.WaitTime = TIME - w.CurrentStart`
Consultation	Walk-up/Appointment Patient (p), Doctor (d)	Start	Controlled	`SCHEDULE Consultation.End at TIME + ConsultationDuration()`

		End	Scheduled	`d.ConsultTime += TIME - d.CurrentStart`
				`START Walk-up/Appt. Patient Leaves WITH p # TRANSITION P.3`
				`START Wait for Patient WITH d # TRANSITION D.3`

Appt. Wait for Consultation	Appt. Patient (a)	Start	Scheduled	`TRIGGER OnStartApptWaitForConsultation WITH a`
		End	Controlled	`a.WaitTime = TIME - a.CurrentStart`
Wait for Patient	Doctor (d)	Start	Scheduled	`TRIGGER OnStartWaitForPatient WITH d`
		End	Controlled	

Table 12.6: Events

Event	Participants	Type	State Change

Simulation Start	-	Scheduled	`SCHEDULE Create Doctor at TIME`
			`SCHEDULE Walk-up Patient Arrives at TIME + InterarrivalTime()`
			`SCHEDULE Appt. Patient Arrives at AppointmentTimes(1)`

Walk-up Patient Arrives	-	Scheduled	`CREATE Walk-up Patient w`
			`w.ID = NextWalkUpIDNum`
			`NextWalkUpIDNum = NextWalkUpIDNum + 1`
			`w.ArrivalTime = TIME`
			`w.WaitTime = 0`
			`SCHEDULE Walk-up Patient Arrives at TIME + InterarrivalTime()`
			`START Walk-up Wait for Consultation WITH w # TRANSITION W.1`

Appt. Patient Arrives	-	Scheduled	`CREATE Appointment Patient a`
			`a.ID = NextApptIDNum`
			`NextWalkUpIDNum = NextWalkUpIDNum + 1`
			`a.ArrivalTime = a.ID`
			`a.AppointmentTime = TIME`
			`SCHEDULE Appt. Patient Arrives at AppointmentTimes(a.ID + 1)`
			`START Appt. Wait for Consultation WITH a # TRANSITION A.1`

Table 12.6: Events

Event	Participants	Type	State Change

Create Doctor	-	Scheduled	`CREATE Doctor d`
			`d.ID = NextDoctorIDNum`
			`NextDoctorIDNum = NextDoctorIDNum + 1`
			`d.ConsultTime = 0`
			`IF d.ID = 1 THEN`
			`    d.Role = "Walk-up"`
			`ELSE`
			`    d.Role = "Appointment"`
			`END IF`
			`IF d.ID < 2 THEN`
			`    SCHEDULE Create Doctor at TIME`
			`END IF`
			`START Wait for Patient WITH d # TRANSITION D.1`

Walk-up Patient Leaves	Walk-up Patient (w)	Scheduled	`Calculate any required statistics for w`
Appt. Patient Leaves	Appointment Patient (a)	Scheduled	`Calculate any required statistics for a`
Simulation Finish	-	Scheduled	`Calculate any required statistics`

12.3 Activity Diagrams

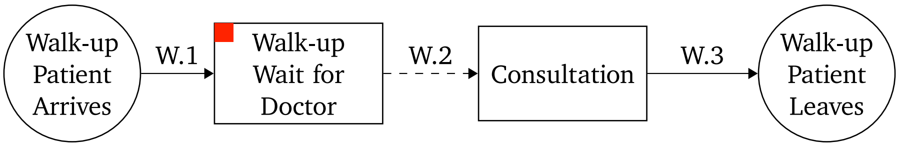

Figure 12.1: Walk-up Patient Activity Diagram

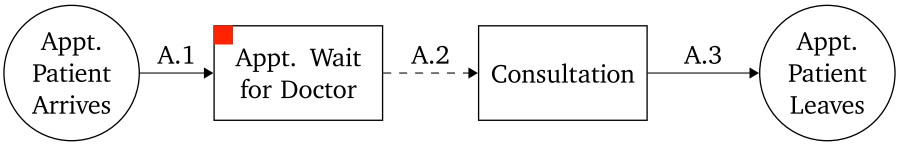

Figure 12.2: Appointment Patient Activity Diagram

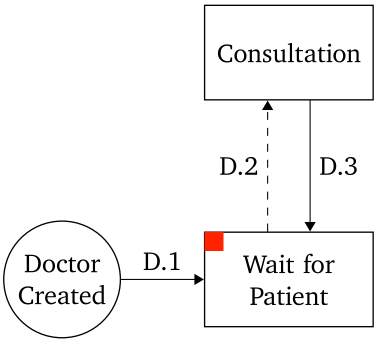

Figure 12.3: Doctor Activity Diagram

12.4 Logic

Table 12.7: OnStartWalkupWaitForConsultation

Triggered by: Walk-up Patient w

```
waiting_walkup_docs = {d FOR d IN D IF d.Role = "Walk-up" AND d.CurrentActivity = "Wait for Patient"}
waiting_appt_docs = {d FOR d IN D IF d.Role = "Appointment" AND d.CurrentActivity = "Wait for Patient"}
IF waiting_walkup_docs IS NOT empty THEN
    d_hat = argmin{d.CurrentStart FOR d IN waiting_walkup_docs}
    START Consultation WITH w, d_hat # TRANSITIONS W.2, D.2
ELSE IF waiting_appt_docs IS NOT empty THEN
    d_hat = argmin{d.CurrentStart FOR d IN waiting_appt_docs}
    START Consultation WITH w, d_hat # TRANSITIONS W.2, D.2
END IF
```

Table 12.8: OnStartApptWaitForConsultation

Triggered by: Appointment Patient a

```
waiting_appt_docs = {d FOR d IN D IF d.Role = "Appointment" AND d.CurrentActivity = "Wait for Patient"}
IF waiting_appt_docs IS NOT empty THEN
    d_hat = argmin{d.CurrentStart FOR d IN waiting_appt_docs}
    START Consultation WITH a, d_hat # TRANSITIONS A.2, D.2
END IF
```

Table 12.9: OnStartWaitForPatient

Triggered by: Doctor d

```
waiting_walkup_pats = {w FOR w IN W IF w.CurrentActivity = "Walk-up Wait for Doctor"}
waiting_appt_pats = {a FOR a IN A IF a.CurrentActivity = "Appt. Wait for Doctor"}
IF d.Role = "Appointment" and waiting_appt_pats IS NOT empty THEN
    p_hat = argmin{p.CurrentStart FOR p IN waiting_appt_pats}
    START Consultation WITH p_hat, d # TRANSITIONS A.2, D.2
ELSE IF waiting_walkup_pats IS NOT empty THEN
    p_hat = argmin{p.CurrentStart FOR p IN waiting_walkup_pats}
    START Consultation WITH p_hat, d # TRANSITIONS W.2, D.2
END IF
```

13 Radiology Clinic

13.1 Data

Table 13.1: 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 13.2: List of Data Modules

Name	Source	Model	Type	Input	Output
PatientInterarrivalTime	Problem Description	Poisson Process	Stochastic	Mean interarrival time	Sample from distribution
NumReceptionists	Problem Description	Constant	Deterministic	-	Value
NumCTMachines	Problem Description	Constant	Deterministic	-	Value
CheckInTime	Problem Description	Uniform Distribution	Stochastic	Min and max time	Sample from distribution
ScanTime	Problem Description	Log-normal Distribution	Stochastic	Mean and std. dev.	Sample from distribution

13.2 Components

Table 13.3: List of Entities

Entity	Attributes
Patient	ID CurrentActivity CurrentStart
Receptionist	ID CurrentActivity CurrentStart
CT Machine	ID CurrentActivity CurrentStart

Table 13.4: List of Transitions

Participant	Name	From Event	To Event
Patient	P.1	Arrive(P)	Wait for check in.Start
	P.2	Wait for check in.End	Check in.Start
	P.3	Check in.End	Wait for scan.Start
	P.4	Wait for scan.End	Scan.Start
	P.5	Scan.End	Leave(P)
Receptionist	R.1	Arrive(R)	Wait for task(R).Start
	R.2	Wait for task(R).End	Check in.Start
	R.3	Check in.End	Wait for task(R).Start
	R.4	Wait for task(R).End	Leave(R)
CT Machine	CT.1	Arrive(CT)	Wait for task(CT).Start
	CT.2	Wait for task(CT).End	Scan.Start
	CT.3	Scan.End	Wait for task(CT).Start
	CT.4	Wait for task(CT).End	Leave(CT)

Table 13.5: Activities

Activity	Participants	Event	Type	State Change
Wait for Check In	Patient (p)	Start	Scheduled	`TRIGGER OnStartWaitForCheckIn WITH p`
		End	Controlled	
Check In	Patient (p), Receptionist (r)	Start	Controlled	`SCHEDULE Check In.End at TIME + CheckInTime()`
		End	Scheduled	`START Wait for Scan WITH p # TRANSITION P.3` `START Wait for Task (R) WITH r # TRANSITION R.3`
Wait for Scan	Patient (p)	Start	Scheduled	`TRIGGER OnStartWaitForScan WITH p`
		End	Controlled	
Scan	Patient (p), CTMachine (c)	Start	Controlled	`SCHEDULE Scan.End at TIME + ScanTime()`

Table 13.5: Activities

Activity	Participants	Event	Type	State Change
		End	Scheduled	`START Leave (P) WITH p # TRANSITION P.5` `START Wait for Task (CT) WITH c # TRANSITION CT.3`
Wait for Task (R)	Receptionist (r)	Start	Scheduled	`TRIGGER OnStartWaitForTaskR WITH r`
		End	Controlled	
Wait for Task (CT)	CTMachine (c)	Start	Scheduled	`TRIGGER OnStartWaitForTaskCT WITH c`
		End	Controlled	

Table 13.6: Events

Event	Participants	Type	State Change

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

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

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

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

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

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

Leave (CT)	CT Machine (c)	Scheduled	`Calculate statistics for c`

13.3 Activity Diagrams

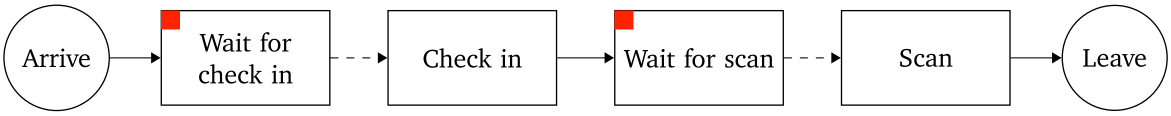

Figure 13.1: Patient Activity Diagram

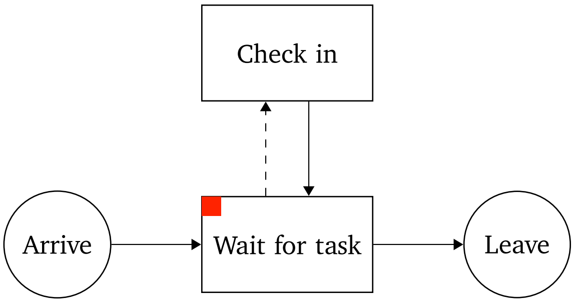

Figure 13.2: Receptionist Activity Diagram

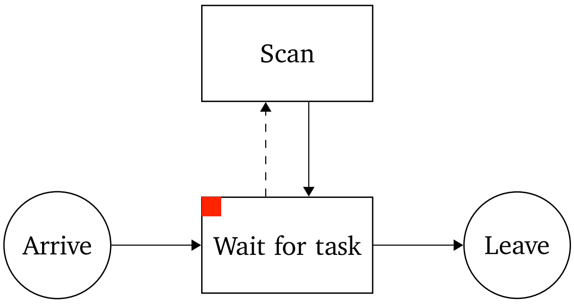

Figure 13.3: CT Activity Diagram

13.4 Logic

Table 13.7: OnStartWaitForCheckIn

Triggered by: Patient p

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

Table 13.8: OnStartWaitForScan

Triggered by: Patient p

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

Table 13.9: OnStartWaitForTaskR

Triggered by: Receptionist r

```
patients = {p FOR p IN P IF p.State = "Wait for Check In"}
IF patients IS NOT empty THEN
   p_hat = argmin{p.CurrentStart FOR p IN patients}
   START Check In WITH p_hat, r # TRANSITIONS P.2, R.2
END IF
```

Table 13.10: OnStartWaitForTaskCT

Triggered by: CTMachine c

```
patients = {p FOR p IN P IF p.State = "Wait for Scan"}
IF patients IS NOT empty THEN
   p_hat = argmin{p.CurrentStart FOR p IN patients}
   START Scan WITH p_hat, c # TRANSITIONS P.4, CT.2
END IF
```

14 Extended Radiology Clinic

14.1 Data

Table 14.1: 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 14.2: List of Data Modules

Name	Source	Model	Type	Input	Output
PatientInterarrivalTime	Problem Description	Poisson Process	Stochastic	Mean interarrival time	Sample from distribution
PatientPriority	Problem Description	Discrete Distribution	Stochastic	Priority Probabilities	Sample from distribution
NumReceptionists	Problem Description	Constant	Deterministic	-	Value
NumCTMachines	Problem Description	Constant	Deterministic	-	Value
CheckInTime	Problem Description	Uniform Distribution	Stochastic	Min and max time	Sample from distribution
ScanTime	Problem Description	Log-normal Distribution	Stochastic	Mean and std. dev.	Sample from distribution

14.2 Components

Table 14.3: List of Entities

Entity	Attributes
Patient	ID CurrentActivity CurrentStart Priority[0]
Receptionist	ID CurrentActivity CurrentStart
CT Machine	ID CurrentActivity CurrentStart NeedMaintenance[0]

Table 14.4: List of Transitions

Participant	Name	From Event	To Event
Patient	P.1	Arrive(P)	Wait for check in.Start
	P.2	Wait for check in.End	Check in.Start
	P.3	Check in.End	Wait for scan.Start
	P.4	Wait for scan.End	Scan.Start
	P.5	Scan.End	Leave(P)
	P.6	Arrive(P)	Wait for scan.Start
Receptionist	R.1	Arrive(R)	Wait for task(R).Start
	R.2	Wait for task(R).End	Check in.Start
	R.3	Check in.End	Wait for task(R).Start
	R.4	Wait for task(R).End	Leave(R)
CT Machine	CT.1	Arrive(CT)	Wait for task(CT).Start
	CT.2	Wait for task(CT).End	Scan.Start
	CT.3	Scan.End	Wait for task(CT).Start
	CT.4	Wait for task(CT).End	Leave(CT)
	CT.5	Wait for task(CT).End	Maintenance.Start
	CT.6	Maintenance.End	Wait for task(CT).Start

Table 14.5: Activities

Activity	Participants	Event	Type	State Change
Wait for Check In	Patient (p)	Start	Scheduled	`TRIGGER OnStartWaitForCheckIn WITH p`
		End	Controlled	
Check In	Patient (p), Receptionist (r)	Start	Controlled	`SCHEDULE Check In.End at TIME + CheckInTime()`
		End	Scheduled	`START Wait for Scan WITH p # TRANSITION P.3` `START Wait for Task (R) WITH r # TRANSITION R.3`

Table 14.5: Activities

Activity	Participants	Event	Type	State Change
Wait for Scan	Patient (p)	Start	Scheduled	`TRIGGER OnStartWaitForScan WITH p`
		End	Controlled	
Scan	Patient (p), CTMachine (c)	Start	Controlled	`SCHEDULE Scan.End at TIME + ScanTime()`
		End	Scheduled	`START Leave (P) WITH p # TRANSITION P.5` `START Wait for Task (CT) WITH c # TRANSITION CT.3`
Wait for Task (R)	Receptionist (r)	Start	Scheduled	`TRIGGER OnStartWaitForTaskR WITH r`
		End	Controlled	
Wait for Task (CT)	CTMachine (c)	Start	Scheduled	`TRIGGER OnStartWaitForTaskCT WITH c`
		End	Controlled	
Maintenance	CTMachine (c)	Start	Controlled	`SCHEDULE Maintenance.End at TIME + 30 minutes`
		End	Scheduled	`c.NeedMaintenance = 0` `START Wait for Task (CT) WITH c # TRANSITION CT.6`

Table 14.6: Events

Event	Participants	Type	State Change

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

Arrival (P)	Patient (p)	Scheduled	`p.ID = NextPatIDNum`
			`p.Priority = PatientPriority()`
			`NextPatIDNum = NextPatIDNum + 1`
			`SCHEDULE Arrival (P) at TIME + PatientInterArrival()`
			`IF p.Priority <= 2 THEN`
			`  START Wait for Scan WITH p # TRANSITION P.6`
			`ELSE`
			`  START Wait for Check In WITH p # TRANSITION P.1`
			`END IF`

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

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

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

Table 14.6: Events

Event	Participants	Type	State Change

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

Leave (CT)	CT Machine (c)	Scheduled	`Calculate statistics for c`
Require Maintenance	CT Machine (c)	Scheduled	`c.NeedMaintenance = 1` `TRIGGER OnRequireMaintenance WITH c`

14.3 Activity Diagrams

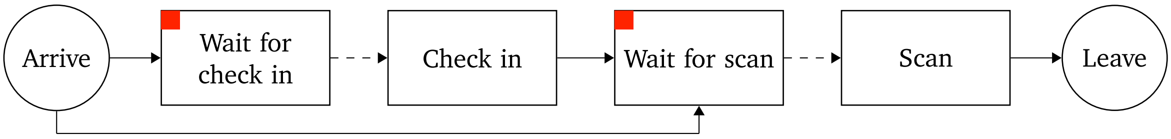

Figure 14.1: Patient Activity Diagram

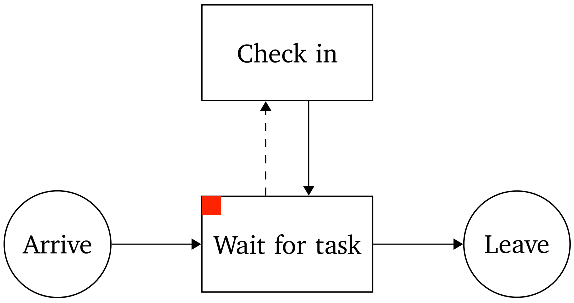

Figure 14.2: Receptionist Activity Diagram

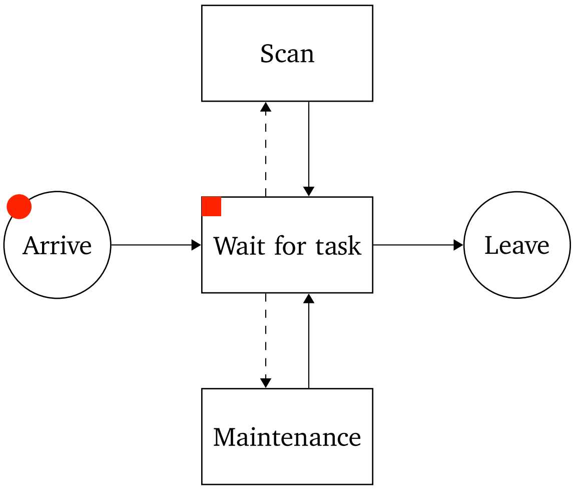

Figure 14.3: CT Activity Diagram

14.4 Logic

Table 14.7: OnStartWaitForCheckIn

Triggered by: Patient p

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

Table 14.8: OnStartWaitForScan

Triggered by: Patient p

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

Table 14.9: OnStartWaitForTaskR

Triggered by: Receptionist r

```
patients = {p FOR p IN P IF p.State = "Wait for Check In"}
IF patients IS NOT empty THEN
    p_hat = argmin{p.CurrentStart FOR p IN patients}
    START Check In WITH p_hat, r # TRANSITIONS P.2, R.2
END IF
```

Table 14.10: OnStartWaitForTaskCT

Triggered by: CTMachine c

```
patients = {p FOR p IN P IF p.State = "Wait for Scan"}
IF c.NeedMaintenance = 1 THEN
  START Maintenance WITH c # TRANSITION CT.5
IF patients IS NOT empty THEN
    top_priority = max{p.Priority FOR p in patients}
    top_patients = {p FOR p IN patients IF p.Priority = top_priority}
    p_hat = argmin{p.CurrentStart FOR p IN top_patients}
    START Scan WITH p_hat, c # TRANSITIONS P.4, CT.2
END IF
```

Table 14.11: OnCTMachineArrive

Triggered by: CT Machine c

```
SCHEDULE Require Maintenance WITH c at TIME + 8 hours
```

Table 14.12: OnRequireMaintenance

Triggered by: CT Machine c

```
SCHEDULE Require Maintenance WITH c at TIME + 8 hours
IF c.CurrentActivity = "Wait for task (C)" THEN
  START Maintenance WITH c # TRANSITION CT.5
END IF
```