DSA3361_WS3

Hui Teng/Yiqun/Qian Jiang

2022-09-26

Contents

Learning Outcomes

1. Be able to draw activity diagram and code in simmer,
2. Extract performance measures, run replications and compute confidence intervals,
3. Understand how to modify scenario to improve performance measure.

Task 1: Fill in Activity Diagram and Code in Simmer

1-1: Call Centre System Scenario

Suppose we operate a simplified call centre system. The time between incoming calls is exponentially distributed with a mean of 0.857 minutes. The central number has only 26 trunk lines, which means that the maximum number of callers at any one time is 26. If all 26 lines are in use, a caller will get a busy signal and be forced to exit.

If the caller has been answered, a recording describing options will be issued: 62% of callers will choose to transfer to technical support, 22% of them will choose sales information and the remaining 16% will choose order-status inquiry, respectively.

The estimated time for this activity is uniformly distributed from 0.1 to 0.6 minutes. Depending on their chosen individual options, callers experience different activity routes:

a) Technical support

- A second recording describing further options will be issued. This activity requires time which is uniformly distributed from 0.1 to 0.5 minutes. The percentage of requests for product types 1, 2, and 3 are 30%, 35% and 35% respectively. For convenience, we refer to the three types as tech-type-1, tech-type-2 and tech-type-3.
- If a qualified technical support person is available for the selected product type, the call is automatically routed to that person. Otherwise, the call will be placed in an electronic queue, where some Jazz music is played repeatedly.
- Suppose we will arrange some supporting engineers for each product type, and each engineer will be reserved to answer questions related to one specific product type.
- Time taken for the tech-type-1 calls is estimated to be triangularly distributed from 8 to 18 minutes, with a peak at 12 minutes.
- Time taken for the tech-type-2 and the tech-type-3 calls is estimated to be triangularly distributed from 3 to 18 minutes, with a peak at 6 minutes.
- Once completed, the caller exits the system.

b) Sales information

- Automatically routed to the sales staff.
- If staff is not available, the caller is treated to soothing new-age space music (after all, we're hoping for a sale *wink*)
- Sales calls are estimated to be triangularly distributed from 4 to 45 minutes, with a peak at 15 minutes.
- Once completed, happy customers exit the system.

c) Order-status inquiry

- callers will be automatically handled by the phone system, and there is no limit on the number the system can handle (except that there are only 26 trunk lines, which is itself a limit, since an ongoing order-status call occupies one of these lines.)
- The estimated time for these transactions is triangularly distributed from 2 to 4 minutes, with a peak at 3 minutes, with 45% of the callers opting to speak to a real person after they have received their order status.
- These calls are routed to the sales staff, where they wait with a lower priority than sales calls. (This means that if an order-status call is in a queue waiting for a salesperson and a new arriving sales call enters, the sales call will be given priority over the order-status call and answered first.)

- These follow-up order-status calls are estimated to last from 2 minutes to 4 minutes, with a peak at 3 minutes (triangular distribution).
- The other 55% of callers exit the system once completed.

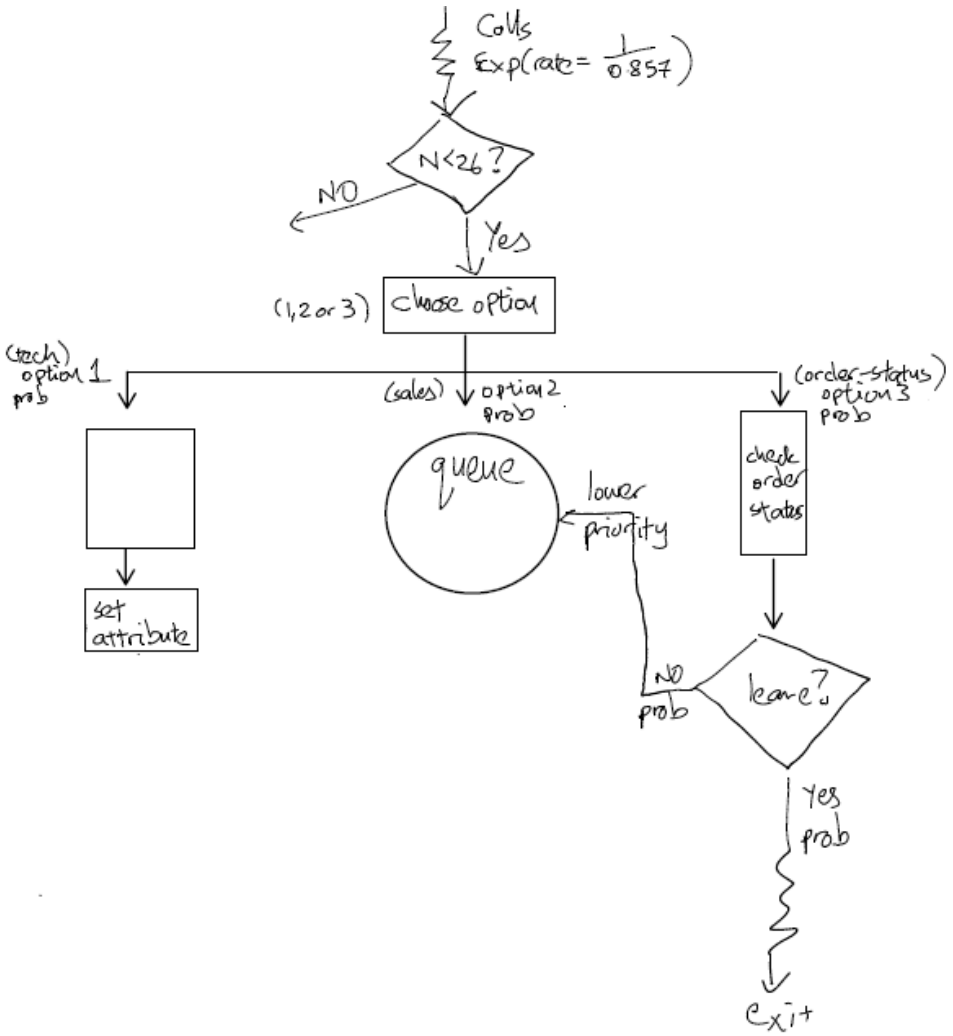
The call centre hours are from 9 am until 6 pm. Over the course of a day there are 8 technical support employees to answer technical support calls, with

- 2 employees devoted to the tech-type-1 calls;
- 3 employees to the tech-type-2 calls; and
- 3 employees to the tech-type-3 calls calls.

There are 4 sales employees to answer the sales calls and those order-status calls that opt to speak to a real person.

1-2: Fill Up the Activity Diagram

The diagram below shows an incomplete activity diagram. Please complete the diagram using the A4 paper provided.



1-3: Set Up Customer Trajectory and Build the Simulation Model

- Suppose N represents the number of calls being handled. Let us consider the following simplified scenario: the call centre is running from 9am to 12pm, and there is no new incoming calls after 12pm. We will run the simulation until we have finished all the calls that come before 12pm.

```
call_centre <- simmer("Call centre simulation") %>%
  add_global("N", 0)

call_traj <- trajectory() %>%
  branch(function() get_global(call_centre, "N") >=26,
         continue = ____,
         trajectory() %>%
           set_attribute("call-type", 0) %>%
```

```
          log_("Got a busy signal.")
        ) %>%
set_global(__________) %>%   # Hint: Update `N` by adding one
log_(function() paste("No. in system:", get_global(call_centre, "N"))) %>%
timeout(function() runif(1, 0.1, 0.6)) %>%
branch(function() sample(c(1,2,3), size=1, prob=__________),
       continue = ____,
       trajectory() %>% #tech call
         set_attribute("call-type", 1) %>%
         timeout(function() runif(1, 0.1, 0.5)) %>%
         branch(function() sample(c(1,2,3), size=1, prob=c(0.3, 0.35, 0.35)),
            continue = ____,
            trajectory() %>%
              seize("tech-1") %>%
              timeout(function() rtriangle(1, 8, 18, 12)) %>%
              release("tech-1"),

            trajectory() %>%
              seize("tech-2") %>%
              timeout(function() rtriangle(1, 3, 18, 6)) %>%
              release("tech-2"),

            trajectory() %>%
              seize("tech-3") %>%
              timeout(function() rtriangle(1, 3, 18, 6)) %>%
              release("tech-3")
         ) %>%
         set_global("N", ___, mod="+") %>%
         log_("Tech question answered, leaving now."),

       trajectory() %>% #sales call
         set_attribute("call-type", 2) %>%
         set_prioritization(c(1, 1, 0)) %>%
         seize("sales") %>%
         timeout(function() rtriangle(1, 4, 45, 15)) %>%
         release("sales") %>%
         set_global("N", ___, mod="+") %>%
         log_("Sales question answered, leaving now."),

       trajectory() %>% #order-status
         timeout(function(){rtriangle(1, 2, 4, 3)}) %>%
         branch(function() runif(1) < ____,
                continue = ____,
                trajectory() %>%
                  set_attribute("call-type", 3) %>%
                  set_global("N", -1, mod="+")  %>%
                  log_("Order-status answered, leaving now.")
                ) %>%
         log_("Order-status answered, need to speak to sales though..") %>%
         set_attribute("call-type", 4) %>%
         seize("sales") %>%
         timeout(function(){rtriangle(1, 2, 4, 3)}) %>%
         release("sales") %>%
```

```
          set_global("N", -1, mod="+") %>%
          log_("Order_status answered, spoken to sales, leaving now."))

set.seed(123)
call_centre %>%
  add_generator("Call", call_traj,
                distribution = to(180, function() rexp(1, rate= 1/0.857)),
                mon=2) %>%
  add_resource("tech-1", capacity = __) %>%
  add_resource("tech-2", capacity = __) %>%
  add_resource("tech-3", capacity = __) %>%
  add_resource("sales", capacity = 4, preemptive = FALSE) %>%
  run
```

To assist you understand `set_prioritization` better, we provide another example at the appendix.

1-4: The Simplified Trajectory and Model

```
  inter_arr_distr <- function() rexp(1, rate= 1/0.857)
  task_duration_options <- function() runif(1, 0.1, 0.6)
  task_duration_tech_types <- function() runif(1, 0.1, 0.5)
  task_duration_tech1 <- function() rtriangle(1, 8, 18, 12)
  task_duration_tech2_tech3 <- function() rtriangle(1, 3, 18, 6)
  task_duration_sales <- function() rtriangle(1, 4, 45, 15)
  task_duration_order <- function() rtriangle(1, 2, 4, 3)
  task_duration_order_sales <- function() rtriangle(1, 2, 4, 3)

  branching_options <- function() {
    sample(c(1,2,3), size=1, prob=c(0.62, 0.22, 0.16))
  }
  branching_tech_types <- function() {
    sample(c(1,2,3), size=1, prob=c(0.3, 0.35, 0.35))
  }
  branching_order <- function() runif(1) < 0.55

  call_centre <- simmer("Call centre simulation") %>%
    add_global("N", 0)

  call_traj <- trajectory() %>%
    branch(function() get_global(call_centre, "N") >=26,
         continue = FALSE,
         trajectory() %>%
           set_attribute("call-type", 0)  # type 0 leaving
         ) %>%
    set_global("N", 1, mod="+") %>%   # Update N by adding one
    timeout(task_duration_options) %>%
    branch(branching_options,
         continue=FALSE,
         trajectory() %>% #tech call
           set_attribute("call-type", 1) %>%
           timeout(task_duration_tech_types) %>%
```

```
          branch(branching_tech_types,
             continue = TRUE,
             trajectory() %>%
               seize("tech-1") %>%
               timeout(task_duration_tech1) %>%
               release("tech-1"),

             trajectory() %>%
               seize("tech-2") %>%
               timeout(task_duration_tech2_tech3) %>%
               release("tech-2"),

             trajectory() %>%
               seize("tech-3") %>%
               timeout(task_duration_tech2_tech3) %>%
               release("tech-3")
          ) %>%
          set_global("N", -1, mod="+"), # type 1 leaving

        trajectory() %>% #sales call
          set_attribute("call-type", 2) %>%
          set_prioritization(c(1, 1, 0)) %>%
          seize("sales") %>%
          timeout(task_duration_sales) %>%
          release("sales") %>%
          set_global("N", -1, mod="+"),  # type 2 leaving

        trajectory() %>% #order-status
          timeout(task_duration_order) %>%
          branch(branching_order,
                 continue=FALSE,
                 trajectory() %>%
                   set_attribute("call-type", 3) %>%
                   set_global("N", -1, mod="+")  # Type 3 Leaving
                 ) %>%
          # some change from order-status to sales
          set_attribute("call-type", 4) %>%
          seize("sales") %>%
          timeout(task_duration_order_sales) %>%
          release("sales") %>%
          set_global("N", -1, mod="+")   # Type 4 Leaving
  )

set.seed(123)
call_centre %>%
  add_generator("Call", call_traj,
                distribution = to(180, inter_arr_distr),
                mon=2) %>%
  add_resource("tech-1", capacity = 2) %>%
  add_resource("tech-2", capacity = 3) %>%
  add_resource("tech-3", capacity = 3) %>%
  add_resource("sales", capacity = 4, preemptive = FALSE) %>%
  run
```

Task 2: Extracting Performance Measures, Running Replications and Computing Confidence Intervals.

Performance measures:

(1) the number of balked customers
(2) the time-averaged and maximum queue size
(3) the utilisation of resources
(4) the mean waiting time

2-1: Measure 1 – Number of Balked Customers

Like WS2, we use the `get_mon_***` families to fetch the monitor data.

```
mon_resources <-  get_mon_resources(call_centre)
mon_arrivals  <-  get_mon_arrivals(call_centre)
mon_attributes <- get_mon_attributes(call_centre)
```

Hands-on Practice

Note that we have put one line of `set_attribute("call-type", *)` for each type of calls. In fact, the balked customers (calls), who are forced to leave at the beginning, are labelled as "type 0". Therefore, we will utilise the attributes data to calculate the first performance measure.

```
head(mon_attributes)

mon_attributes1 <- mon_attributes[mon_attributes$key != "N",]
tail(mon_attributes1)

# number of customers
nrow(mon_attributes1)

# check for duplicates
sum(duplicated(mon_attributes1$name))
```

Hands-on Practice

> Q: Pls use `nrow` to calculate the number of customers (calls) for each type. In particular, how many balked customers are there?

```
nrow(______________________)
```

Alternatively, we can use the `dplyr` package.

```
library(dplyr)

# the breakdown of customers by type
mon_attributes1 %>% group_by(______) %>%
                    summarise(______)
```

```
# Task: add a new column calculating the proportion


# Task: removing the type 0 and recalculate the proportion
```

Q: Pls compare the prop column of the above table with the probability of the three options (tech, sales or order-status). What do you observe?

the change of the global parameter, `N`

```
mon_attributes2 <- mon_attributes[mon_attributes$key == "N",]
head(mon_attributes2)

# Task: Using `mon_attributes2`, calculate the number of rows with N value = 26.
```

Q: Why the above number is less than the number of balked customers?

2-2: Measure 2 – the Time-averaged and the Maximum Queue Size

Recall that at WS2, we can read off the time-averaged queue length and the instantaneous queue size from the visualisation of the resources data.

```
head(mon_resources)

plot(mon_resources, items=c("queue", "server"))

# the instantaneous resource usage
plot(mon_resources, items=c("queue", "server"), step = TRUE)
```

Q: How do we only generate the above plots for one particular resource?

We can fetch the queue sizes from the plot, as follows.

```
plot1 <- plot(mon_resources, items=c("queue", "server"))
head(plot1$data[,-5])

# to get the time-weighted queue size for sales, by end of the model
df_resource <- plot1$data
tail(df_resource[_____________], n = 1)

# to get the max queue size for sales
max(df_resource[_____________])
```

Hands-on Practice

Alternatively, we can use the `dplyr` package functions to uniformly show the queue size per resource.

```
# time-averaged queue size
plot1$data %>% group_by(resource, item) %>%
               filter(item == "queue") %>%
               summarise(time_weighted_size = tail(mean, n = 1) )

# Task: max queue size
```

2-3: Measure 3 – Utilisation of Resources

```
plot2 <- plot(mon_resources, metric="utilization")
plot2

plot2$data
```

Q: What do you think is the ideal utilisation, or optimum utilisation range?

2-4: Measure 4 – Mean Waiting Time, per Resource Type

```
mon_arrivals_sub <- get_mon_arrivals(call_centre, per_resource = TRUE)
tail(mon_arrivals_sub)

# check whether any call involves at least two resources
sum(duplicated(mon_arrivals_sub$name))

# a function to calculate the waiting time
ave_wait_time <- function(df) {
    mean(df$end_time - df$start_time - df$activity_time)
}

# the mean waiting time for customers who talked to sales
ave_wait_time(mon_arrivals_sub[mon_arrivals_sub$resource ==
                                 "sales",])
```

Q: Pls calculate the mean waiting time for other type of customers(tech-1, tech-2, etc)?

Hands-on Practice

Task: use the `dplyr` package functions to calculate the mean waiting time per resource.

The following is to compare the number of customers per resource VS. that per types. The former one utilises the arrivals data while the latter one utilises the attributes data.

```
# the number of customers (calls) per resource
mon_arrivals_sub %>% group_by(resource) %>%
                     summarise(n = n())

# the number of customers (calls) per types
mon_attributes1 %>% group_by(value) %>%
                    summarise(n = n())
```

Q: What do you observe?

Hands-on Practice (Optional)

As the type 2 and type 4 customers (calls) talked to sales, is there any way to calculate the mean waiting time for type 2 and type 4 customers, respectively?

A left join in R is a merge operation between two data frames where the merge returns all of the rows from the left table and attach the info of the right table next to the matching rows, at which the two tables share the same content at the identifier column.

A left join in R will NOT return values of the second table which do not already exist in the first table.

```
head(mon_arrivals_sub)
head(mon_attributes1)

arrivals_attri <- left_join(mon_arrivals_sub, mon_attributes1, by = _____)
head(arrivals_attri)

# task: check the mean waiting time of each type. (for type 1, 2, 4)
```

```
arrivals_attri %>% mutate(waiting_time = end_time - start_time - activity_time) %>%
               group_by(resource, value) %>%
               summarise(mean_waiting = mean(waiting_time))

# let us compare it with the earlier table that we got
mon_arrivals_sub %>% mutate(waiting_time = end_time - start_time - activity_time) %>%
                group_by(resource) %>%
                summarise(mean_waiting = mean(waiting_time))
```

Q: Comparing the above two tables, what do you observe?

2-5: Replications, Estimates & Confidence Intervals

We just introduced 4 performance measures. For each measure/statistic, we could wrap up the whole model inside a function, `single_rep`. Then we run the model multiple times, derive the estimate of the performance measure, and the confidence interval of it.

Let us run this analysis for the first performance measure, the number of balked customers.

```
single_rep <- function() {
  inter_arr_distr <- function() rexp(1, rate= 1/0.857)

  task_duration_options <- function() runif(1, 0.1, 0.6)
  task_duration_tech_types <- function() runif(1, 0.1, 0.5)
  task_duration_tech1 <- function() rtriangle(1, 8, 18, 12)
  task_duration_tech2_tech3 <- function() rtriangle(1, 3, 18, 6)
  task_duration_sales <- function() rtriangle(1, 4, 45, 15)
  task_duration_order <- function() rtriangle(1, 2, 4, 3)
  task_duration_order_sales <- function() rtriangle(1, 2, 4, 3)

  branching_options <- function() {
    sample(c(1,2,3), size=1, prob=c(0.62, 0.22, 0.16))
```

```
}
branching_tech_types <- function() {
  sample(c(1,2,3), size=1, prob=c(0.3, 0.35, 0.35))
}
branching_order <- function() runif(1) < 0.55

call_centre <- simmer("Call centre simulation") %>%
  add_global("N", 0)

call_traj <- trajectory() %>%
  branch(function() get_global(call_centre, "N") >=26,
       continue = FALSE,
       trajectory() %>%
         set_attribute("call-type", 0)
       ) %>%
  set_global("N", 1, mod="+") %>%
  timeout(task_duration_options) %>%
  branch(branching_options,
       continue=FALSE,
       trajectory() %>% #tech call
         set_attribute("call-type", 1) %>%
         timeout(task_duration_tech_types) %>%
         branch(branching_tech_types,
            continue = TRUE,
            trajectory() %>%
              seize("tech-1") %>%
              timeout(task_duration_tech1) %>%
              release("tech-1"),

            trajectory() %>%
              seize("tech-2") %>%
              timeout(task_duration_tech2_tech3) %>%
              release("tech-2"),

            trajectory() %>%
              seize("tech-3") %>%
              timeout(task_duration_tech2_tech3) %>%
              release("tech-3")
         ) %>%
         set_global("N", -1, mod="+"),

       trajectory() %>% #sales call
         set_attribute("call-type", 2) %>%
         set_prioritization(c(1, 1, 0)) %>%
         seize("sales") %>%
         timeout(task_duration_sales) %>%
         release("sales") %>%
         set_global("N", -1, mod="+"),

       trajectory() %>% #order-status
         timeout(task_duration_order) %>%
         branch(branching_order,
                continue=FALSE,
```

```
                  trajectory() %>%
                    set_attribute("call-type", 3) %>%
                    set_global("N", -1, mod="+")
                  ) %>%
            set_attribute("call-type", 4) %>%
            seize("sales") %>%
            timeout(task_duration_order_sales) %>%
            release("sales") %>%
            set_global("N", -1, mod="+")
          )

  call_centre %>%
    add_generator("Call", call_traj,
                  distribution = to(180, inter_arr_distr),
                  mon=2) %>%
    add_resource("tech-1", capacity = 2) %>%
    add_resource("tech-2", capacity = 3) %>%
    add_resource("tech-3", capacity = 3) %>%
    add_resource("sales", capacity = 4, preemptive = FALSE) %>%
    run %>% wrap()
}

set.seed(123)
c1.envs <- replicate(20, single_rep())
```

Estimates and Confidence Intervals

You might need to modify the following part if you wish to calculate a different performance measure.

Tips: you can change the `attributes` to `arrivals` or `resources`; you can wrap one of the above performance measures up inside a function, which needs an input of a data frame.

```
c1_attributes <- get_mon_attributes(c1.envs)

head(c1_attributes)

c1_attributes1 %>%
  filter(key != "N") %>%
  group_by(replication) %>%
  summarize(num_balked = sum(value == 0)) %>%
  dplyr::select(num_balked) %>% unlist() -> num_cust_vec

num_cust_vec

# to calculate the Normal-based 95% CI of the measure/statistic
n <- length(num_cust_vec)
sd1  <- sd(num_cust_vec)
mu   <- mean(num_cust_vec)
moe  <- qnorm(0.025)*sd1/sqrt(n)
ci_95 <- c(mu + moe, mu - moe)

mu
ci_95
```

```
ci1 <- formatCI(ci_95)
print(paste0("The normal-based 95% CI for number of balked customers is ", ci1))
```

Among the 20 replications, on average, there are 41.6 balked calls, and the normal-based 95% confidence interval is between 33.7 and 49.4.

If we are interested in the proportion of the calls who were balked, we could revise the code as follows.

```
# the function is to calculate the proportion of balked calls
prop_balked <- function(df) {
  nrow(df[df$value == 0,])/nrow(df)
}

# the estimate and the 95% CI of the proportion measure
n <- nrow(c1_attributes1)
(p <- prop_balked(c1_attributes1))

moe <- qnorm(0.975)*sqrt(p*(1-p)/n)
ci_95 <- c(p - moe, p + moe)
ci_95
```

Hands-on Practice

Likewise, you may pick another one of the performance measures, run the simulation multiple times, and calculate the estimate and the 95% confidence interval of the measure.

Task 3: Modifications

As the final task, let us play a game!

The main target is to modify the model in order to improve the performance, say, reduce the number/proportion of balked customers, or reduce the queue size, or reduce the mean waiting time, or improve the utilisation rates.

Discussion Points

1) What can we modify to improve performance?

Possible solution(s):

2) How can we make this model more realistic?

Possible solution(s):

3) How were the distributions estimated by the team?

Possible solution(s):

Summary/Key Takeaways

1. To apply simulations in the workplace, we may need to:
    - simplify the real scenario, and list the working targets
    - draw the activity diagram
    - set up the trajectory/trajectories
    - toggle between the above steps to modify it till a satisfying level
    - build the simulation model
    - modify the trajectory/model by trial and error with the monitor data
    - extract the performance measure(s), run replications, and compute the estimates and the confidence intervals
    - make modifications (to arrival rates, or capacity number, or even change the trajectory for another design)
    - compare the outputs among the models (original vs. modified ones), and make your data-informed decisions
2. R specifics involved:
    - `trajectory()`, `simmer()`, `seize()` and `release()`
    - `get_mon_resources()`,`get_mon_arrivals` and `get_mon_attributes`
    - `set_attribute()` and `set_global()`
    - `branch()` and `set_priorization()`

References

1. The book `Simulation`, by Sheldon M. Ross
2. Bank tutorial II, by Duncan Garmonsway

Appendix

We provided two extra examples to assist you better understand on `set_priorization`.

Example 1 – A Simplified Call Centre System Scenario

In our earlier call centre scenario, we designed that the type 2 and type 4 customers have different priority. The default setting of priority is 0. We set the priority of type 2 calls as 1, by `set_prioritization(c(1, 1, 0))`. Inside the triple, the first number indicates the priority level, while the other two numbers, `(preemptible, restart)`, are only relevant when `add_resource("sales", preemptive = TRUE)`. We will not go into details here. Feel free to check with us.

Now, the type 2 calls have a higher priority than the type 4 calls (with default priority 0). The basic idea is that the customers in the queue typically follow the first-come-first-serve principle, without prioritization. Yet, once there is a difference in the priorities among the customers in the queue. The customer, who comes later but has a higher priority, will be served first.

Let us use the following simple example to check it carefully.

```
call_centre1 <- simmer("Call centre simulation")

call_traj1 <- trajectory("phone-call") %>%
  branch(function() sample(c(1,2), size=1, prob=c(0.4, 0.6)),
         continue = FALSE,
         trajectory() %>%  # sales call
           set_attribute("call-type", 2)  %>%
           set_attribute("start_time", function() {now(call_centre1)}) %>%
           log_(function() {
                paste("Type 2 arriving with queue size",
                      get_queue_count(call_centre1, "sales"))
            }) %>%
           set_prioritization(c(1, 1, 0)) %>%
           seize("sales") %>%
           log_(function() {paste("Now my turn, waited", now(call_centre1) -
                          get_attribute(call_centre1, "start_time"))}) %>%
           timeout(3) %>%
           release("sales") %>%
           log_("Finished, leaving now"),

         trajectory() %>% # order-status
           set_attribute("call-type", 4)  %>%
           set_attribute("start_time", function() {now(call_centre1)}) %>%
           log_(function() {
                paste("Type 4 arriving with queue size",
                      get_queue_count(call_centre1, "sales"))
            }) %>%
           seize("sales") %>%
           log_(function() {paste("Now my turn, waited", now(call_centre1) -
                           get_attribute(call_centre1, "start_time"))}) %>%
           timeout(3) %>%
           release("sales") %>%
           log_("Finished, leaving now."))

set.seed(123)
```

```
call_centre1 %>% add_generator("Call", call_traj1,
                               at(c(0:4, -1)),
                               mon=2) %>%
  add_resource("sales", capacity = 1, preemptive = FALSE) %>%
  run(until = 21)
```

Q: what happened to `call 2`?