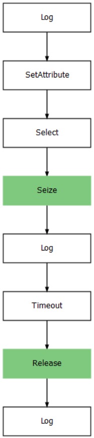
Complicated bank example coded



Outline

- Simulations with `simmer` package
  - Scenario 1
  - Scenario 2
  - Scenario 3a
  - Scenario 3b
- Data-informed decision-making

Learning Objectives

1. Use `simmer` package to perform complicated simulations.
   - Learn the different queuing systems and policies.
2. Create one replication of a finite horizon simulation of 1000 time units.

Simulations with `simmer` package

Simulations with `simmer` package

Recall the following scenarios: Many customers arriving at a random simulated time, then..

Scenario 1

They each look around for a random simulated time and then leave.

Scenario 2

They each join the main queue to be served by the only counter (server) for a random simulated time and then leave.

Scenario 3a

They each join the main queue to be served by the next available counter (there are two) for a random simulated time and then leaves.

Scenario 3b

They each join an individual queue in front of a particular counter (based on a chosen queue selection policy) to be served for a random simulated time and then leaves.

Scenario 1

Many customers arriving at a random simulated time, look around for a random simulated time and then leave.

- There is no queuing.

Scenario 1

cont'd

- For the random arrival, we may simulate this using draws from an exponential distribution [eg. by using `rexp(n, rate=1/10)`]
- The rate (β value) in this exponential distribution is something that we have to figure out before running the simulation.
  - Done by observing the average inter-arrival time of customers on a normal day at the bank over a period of time.
- Each customer also stays in the bank for a different amount of time, and this can be simulated by another exponential distribution.

Scenario 1

cont'd

```
task_duration <- function()
                    {rexp(n = 1, rate = 1/12)}

customer <-
  trajectory("Customer's path") %>%
  timeout(task = task_duration)

gen_arrivals <- function()
             {c(rexp(n = 50, rate = 1/10), -1)}

bank <-
  simmer("bank") %>%
  add_generator(name_prefix = "Customer",
                trajectory = customer,
                distribution = gen_arrivals)

bank %>% run(until = 1000)
```

- The ‘customer’ object is a template for creating new customers.
- This object still needs to be called within the bank environment as a *trajectory* to be *generated*.

Scenario 1

cont'd

Note: for detailed output, `log_` functions were added to the code. Please refer to the accompanying Rmd file for full code.

```
11.2789: Customer0: I arrived!
22.4243: Customer1: I arrived!
24.1159: Customer0: I finished at 24.116
32.7907: Customer1: I finished at 32.791
36.6729: Customer2: I arrived!

...

503.572: Customer48: I arrived!
506.064: Customer49: I arrived!
511.044: Customer47: I finished at 511.044
533.761: Customer48: I finished at 533.760
543.156: Customer49: I finished at 543.156
simmer environment: bank | now: 543.155776969034 | next:
{ Monitor: in memory }
{ Source: Customer | monitored: 1 | n_generated: 50 }
```

Scenario 2

Many customers arriving at a random simulated time, each joining the main queue to be served by the only server for a random simulated time and then leave.

- We cannot expect customers to just enter and do nothing.
  - Customers are going to require service from the bank teller.
  - Extend the previous simulation to include a service counter (resource).

Scenario 2

cont'd

```
task_duration <- function() {rexp(n = 1, rate = 1/12)}
curr_time <- function() {now(bank)}

customer <-
  trajectory("Customer's path") %>%
  log_("I arrived!") %>%
  set_attribute("start_time", curr_time) %>%
  seize("Counter") %>%
  log_(function() {paste("I waited for",
   now(bank) - get_attribute(bank, "start_time"))}) %>%
  timeout(task = task_duration) %>%
  release("Counter") %>%
  log_(function() {paste("I finished at", now(bank))})
```

- A Counter is seized before the `timeout` function and released after.
- The `seize` function checks if the resource is available before performing the specified actions.

Scenario 2

cont'd

```
gen_arrivals <- function()
              {c(rexp(n = 50, rate = 1/10), -1)}

bank <-
  simmer("bank") %>%
  add_resource("Counter", capacity = 1) %>%
  add_generator(name_prefix = "Customer",
                trajectory = customer,
                distribution = gen_arrivals)

bank %>% run(until = 1000)
```

- The Counter must be added as a resource in the bank environment.

Scenario 2

cont'd

```
11.2789: Customer0: I arrived!
11.2789: Customer0: I waited for 0
22.4243: Customer1: I arrived!
24.1159: Customer0: I finished at 24.116
24.1159: Customer1: I waited for 1.692

...

524.588: Customer47: I finished at 524.588
524.588: Customer48: I waited for 21.016
554.777: Customer48: I finished at 554.777
554.777: Customer49: I waited for 48.712
591.868: Customer49: I finished at 591.868
simmer environment: bank | now: 591.868140104537 | next:
{ Monitor: in memory }
{ Resource: Counter | monitored: TRUE | server status: 0(1) | queue status: 0(Inf) }
{ Source: Customer | monitored: 1 | n_generated: 50 }
```

Scenario 3a

Many customers arriving at a random simulated time, each joining the main queue to be served by the next available counter (there are two) for a random simulated time and then leaves.

- Many different service counters so that customers may be managed more efficiently.
- Are customers going to make one queue or are they going to form separate queues in front of each counter?
  - Scenario 3a: There is only ONE queue for the two counters.

Scenario 3a

cont'd

```
task_duration <- function() {rexp(n = 1, rate = 1/12)}
curr_time <- function() {now(bank)}

customer <-
  trajectory("Customer's path") %>%
  log_("I arrived!") %>%
  set_attribute("start_time", curr_time) %>%
  seize("Counter") %>%
  log_(function() {paste("I waited for",
   now(bank) - get_attribute(bank, "start_time"))}) %>%
  timeout(task = task_duration) %>%
  release("Counter") %>%
  log_(function() {paste("I finished at", now(bank))})
```

- This code is identical to Scenario 2.

Scenario 3a

cont'd

```
gen_arrivals <- function()
              {c(rexp(n = 50, rate = 1/10), -1)}

bank <-
  simmer("bank") %>%
  add_resource("Counter", capacity = 2) %>%
  add_generator(name_prefix = "Customer",
                trajectory = customer,
                distribution = gen_arrivals)

bank %>% run(until = 1000)
```

- The only change is that capacity is set as 2 within add_resource.

Scenario 3a

cont'd

```
11.2789: Customer0: I arrived!
11.2789: Customer0: I waited for 0
22.4243: Customer1: I arrived!
22.4243: Customer1: I waited for 0
24.1159: Customer0: I finished at 24.116

...

506.064: Customer49: I arrived!
511.044: Customer47: I finished at 511.047
511.044: Customer49: I waited for 4.979
533.761: Customer48: I finished at 533.761
548.135: Customer49: I finished at 548.135
simmer environment: bank | now: 548.135177892469 | next:
{ Monitor: in memory }
{ Resource: Counter | monitored: TRUE | server status: 0(2) | queue status: 0(Inf) }
{ Source: Customer | monitored: 1 | n_generated: 50 }
```

Scenario 3b

Many customers arriving at a random simulated time, each joining an individual queue in front of a particular counter (based on a chosen queue selection policy) to be served for a random simulated time and then leaves.

- In Scenario 3a, there is only ONE queue for the two counters.
- Consider having an individual queue per counter.
- How to assign customers into the separate queues? Specify via `policy` argument:
  - shortest-queue
  - round-robin
  - random

Scenario 3b

cont'd

```
task_duration <- function() {rexp(n = 1, rate = 1/12)}
curr_time <- function() {now(bank)}

customer <-
  trajectory("Customer's path") %>%
  log_("I arrived!") %>%
  set_attribute("start_time", curr_time) %>%
  select(c("Counter 1", "Counter 2"),
          policy = "random") %>%
  seize_selected() %>%
  log_(function() {paste("I waited for",
   now(bank) - get_attribute(bank, "start_time"))}) %>%
  timeout(task = task_duration) %>%
  release_selected() %>%
  log_(function() {paste("I finished at", now(bank))})
```

- We need to select a counter based on specified policy.
- Then use `seize_selected` to seize the selected counter.

Scenario 3b

cont'd

```
gen_arrivals <- function()
              {c(rexp(n = 50, rate = 1/10), -1)}

bank <-
  simmer("bank") %>%
  add_resource("Counter 1", capacity = 1) %>%
  add_resource("Counter 2", capacity = 1) %>%
  add_generator(name_prefix = "Customer",
                trajectory = customer,
                distribution = gen_arrivals)

bank %>% run(until = 1000)
```

- We add a separate resource for each Counter to make two separate queues.

Scenario 3b

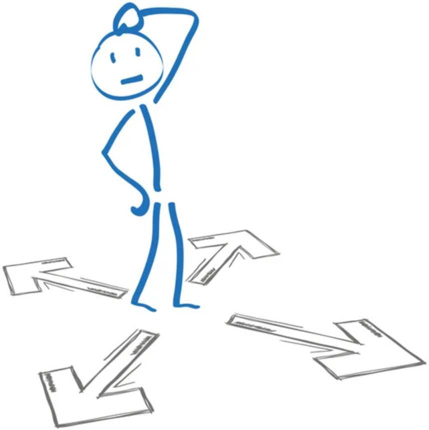
cont'd

```
11.2789: Customer0: I arrived!
11.2789: Customer0: I waited for 0
21.6453: Customer0: I finished at 21.645
22.4243: Customer1: I arrived!
22.4243: Customer1: I waited for 0

...

535.046: Customer45: I finished at 535.046
535.046: Customer46: I waited for 44.084
553.41: Customer46: I finished at 553.410
553.41: Customer48: I waited for 49.838
563.64: Customer48: I finished at 563.640
simmer environment: bank | now: 563.640332143703 | next:
{ Monitor: in memory }
{ Resource: Counter 1 | monitored: TRUE | server status: 0(1) | queue status: 0(Inf)
{ Resource: Counter 2 | monitored: TRUE | server status: 0(1) | queue status: 0(Inf)
{ Source: Customer | monitored: 1 | n_generated: 50 }
```

Data-informed decision-making



Data-informed decision-making

After performing four different simulations, which is the best?

- How do we quantify "best"?
  - Lowest average customer's time spent by customer with a teller (activity time)?
  - Lowest average customer's duration in bank (flow time)?
  - Lowest average customer's waiting time (difference between flow time and activity time)?
  - Lowest elapsed time for the simulation?
- Is one sample of each simulation sufficient to give a conclusion?

Summary

In this video, we have:

1. Used `simmer` package to perform complicated simulations.
   - Learned the different queuing systems and policies.
2. Created one replication of a finite horizon simulation of 1000 time units.