Simmer for our DC Simulation

```
library(simmer)
```

Notes for making the DC Simulation in Simmer

I'm going through the documentation for Simmer to see how we can accomplish the things we need to do. Each major "thing" is in a sub-section.

Priority Queing

This example shows how priority (without preemption) can be handled.

Bank 2

```
set.seed(1933)

bank <- simmer()

customer <-
  trajectory("Customer's path") %>%
  set_attribute("start_time", function() {now(bank)}) %>%
  log_(function() {
         paste("Queue is", get_queue_count(bank, "counter"), "on arrival")
         }) %>%
  seize("counter") %>%
  log_(function() {paste("Waited", now(bank) - get_attribute(bank, "start_time"))}) %>%
  timeout(12) %>%
  release("counter") %>%
  log_("Completed")

bank <-
  simmer("bank") %>%
  add_resource("counter") %>%
  add_generator("Customer", customer, function() {c(0, rexp(4, 1/10), -1)}) %>%
  add_generator("Mario", customer, at(22), priority = 1) %>%
  add_generator("Guido", customer, at(22), priority = 2)

bank %>% run(until = 400)
```

```
## 0: Customer0: Queue is 0 on arrival
## 0: Customer0: Waited 0
## 0.177365: Customer1: Queue is 0 on arrival
## 8.64106: Customer2: Queue is 1 on arrival
## 12: Customer0: Completed
## 12: Customer1: Waited 11.8226352687925
## 21.1346: Customer3: Queue is 1 on arrival
## 22: Mario0: Queue is 2 on arrival
## 22: Guido0: Queue is 2 on arrival
## 24: Customer1: Completed
## 24: Guido0: Waited 2
```

```
## 28.0923: Customer4: Queue is 3 on arrival
## 36: Guido0: Completed
## 36: Mario0: Waited 14
## 48: Mario0: Completed
## 48: Customer2: Waited 39.3589401547341
## 60: Customer2: Completed
## 60: Customer3: Waited 38.8654149797765
## 72: Customer3: Completed
## 72: Customer4: Waited 43.9076510670601
## 84: Customer4: Completed
```

```
## simmer environment: bank | now: 84 | next:
## { Monitor: in memory }
## { Resource: counter | monitored: TRUE | server status: 0(1) | queue status: 0(Inf) }
## { Source: Customer | monitored: 1 | n_generated: 5 }
## { Source: Mario | monitored: 1 | n_generated: 1 }
## { Source: Guido | monitored: 1 | n_generated: 1 }
```

```
bank %>%
  get_mon_arrivals() %>%
  transform(waiting_time = end_time - start_time - activity_time)
```

```
##        name start_time end_time activity_time finished replication
## 1 Customer0  0.0000000       12            12     TRUE           1
## 2 Customer1  0.1773647       24            12     TRUE           1
## 3    Guido0 22.0000000       36            12     TRUE           1
## 4    Mario0 22.0000000       48            12     TRUE           1
## 5 Customer2  8.6410598       60            12     TRUE           1
## 6 Customer3 21.1345850       72            12     TRUE           1
## 7 Customer4 28.0923489       84            12     TRUE           1
##   waiting_time
## 1      0.00000
## 2     11.82264
## 3      2.00000
## 4     14.00000
## 5     39.35894
## 6     38.86541
## 7     43.90765
```

Scheduling

I think we can use this to schedule downtime in the DC. That is, when they're running and not. Capacity seems to be contained here too, so maybe that can be used to limit daily capacity.

https://r-simmer.org/reference/schedule.html

```
# Schedule 3 units from 8 to 16 h
#          2 units from 16 to 24 h
#          1 units from 24 to 8 h
capacity_schedule <- schedule(c(8, 16, 24), c(3, 2, 1), period=24)

env <- simmer() %>%
  add_resource("dummy", capacity_schedule)
```

Here's another example that encompases the idea of schedule and capcity, also from https://cran.r-project.org/web/packages/simmer/vignettes/simmer-04-bank-2.html.

```
library(simmer)

maxTime = 400
set.seed(393937)

bank <- simmer()

customer <-
  trajectory("Customer's path") %>%
  log_(function()
    if (get_capacity(bank, "door") == 0)
      "Here I am but the door is shut."
    else "Here I am and the door is open."
  ) %>%
  seize("door") %>%
  log_("I can go in!") %>%
  release("door") %>%
  seize("counter") %>%
  timeout(function() {rexp(1, 1/10)}) %>%
  release("counter")

openTime <- rexp(1, 1/10)

door_schedule <- schedule(c(0, openTime), c(0, Inf))

doorman <-
  trajectory() %>%
  timeout(openTime) %>%
  log_("Ladies and Gentlemen! You may all enter.")

bank <-
  simmer("bank") %>%
  add_resource("door", capacity = door_schedule) %>%
  add_resource("counter") %>%
  add_generator("Customer",
                customer,
                at(c(0, cumsum(rexp(5 - 1, 0.1))))) %>%
  add_generator("Doorman", doorman, at(0))

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

```
## 0: Customer0: Here I am but the door is shut.
## 6.44076: Customer1: Here I am but the door is shut.
## 8.77564: Customer2: Here I am but the door is shut.
## 19.7241: Doorman0: Ladies and Gentlemen! You may all enter.
## 19.7241: Customer0: I can go in!
## 19.7241: Customer1: I can go in!
## 19.7241: Customer2: I can go in!
## 24.2037: Customer3: Here I am and the door is open.
## 24.2037: Customer3: I can go in!
## 33.3576: Customer4: Here I am and the door is open.
## 33.3576: Customer4: I can go in!
```

```
## simmer environment: bank | now: 79.2542060826083 | next:
```

```
## { Monitor: in memory }
## { Resource: door | monitored: TRUE | server status: 0(Inf) | queue status: 0(Inf) }
## { Resource: counter | monitored: TRUE | server status: 0(1) | queue status: 0(Inf) }
## { Source: Customer | monitored: 1 | n_generated: 5 }
## { Source: Doorman | monitored: 1 | n_generated: 1 }
```

```
bank %>%
  get_mon_arrivals() %>%
  transform(waiting_time = end_time - start_time - activity_time)
```

```
##        name start_time end_time activity_time finished replication
## 1  Doorman0   0.000000 19.72408     19.724085     TRUE           1
## 2 Customer0   0.000000 46.61084     26.886751     TRUE           1
## 3 Customer1   6.440758 64.20811     17.597279     TRUE           1
## 4 Customer2   8.775635 71.50006      7.291950     TRUE           1
## 5 Customer3  24.203687 73.96270      2.462632     TRUE           1
## 6 Customer4  33.357600 79.25421      5.291510     TRUE           1
##   waiting_time
## 1      0.00000
## 2     19.72408
## 3     40.17008
## 4     55.43248
## 5     47.29638
## 6     40.60510
```

Loading Orders from a DataFrame?

https://r-simmer.org/reference/add_dataframe.html

This should work to pre-load all the orders with arrival times and priorities.

Closure to build function that returns repeating values

I think we can use this as a function that takes the capacity array to work as part of a `schedule` object to set a fixed schedule with variable capacity. This is in the middle of https://cran.r-project.org/web/packages/simmer/vignettes/simmer-04-bank-1.html.

```
# Function to specify a series of waiting times, that loop around
loop <- function(...) {
    time_diffs <- c(...)
    i <- 0
    function() {
      if (i < length(time_diffs)) {
        i <<- i+1
      } else {
        i <<- 1
      }
      return(time_diffs[i])
    }
  }

x <- loop(10, 7, 20)
x(); x(); x(); x(); x()
```

```
## [1] 10
## [1] 7
## [1] 20
## [1] 10
## [1] 7
```