BAREKENG: Journal of Mathematics and Its Applications
March 2023 Volume 17 Issue 1 Page 0581–0592
P-ISSN: 1978-7227 E-ISSN: 2615-3017
https://doi.org/10.30598/barekengvol17iss1pp0581-0592

OUTPUT VISUALIZATION FROM RESULT OF DISCRETE EVENT SYSTEM SIMULATION WITH 'simmer' R PACKAGE

I Gusti Agung Anom Yudistira[1*], Rinda Nariswari [2], Samsul Arifin [3]

[1,2,3] *Statistics Department, School of Computer Science, Bina Nusantara University, Jl. Kebon Jeruk Raya No. 27, Kebon Jeruk, Jakarta Barat 11530, Indonesia*

*Corresponding author's e-mail: ** ***i.yudistira@binus.ac.id***

ABSTRACT

Article History:

Received: 29th November 2022
Revised: 11th February 2023
Accepted: 20th February 2023

Keywords:

Discrete Event System Simulation;
R programming;
Simmer;
Simulation;
Visualization.

This study aims to describe the various capabilities of the simmer package on R, especially in running a discrete event simulation model of a circular system, then develop a DES simulation model building technique, which is effective and can represent real systems well, and explore the simulation output on this simmer, both in statistical summary form and parameter estimation. The method used in this research is the literature study with descriptive and exploratory approaches. Model development is more effective when it is carried out starting from simple models, to more complex forms step by step, and describing the system using a flow chart. Replication for simulations is easy to perform so as to get standard error values for model parameter estimators. The stages in developing a discrete event simulation model with a simmer, start with compiling a simple flowchart to a more complex form, and replication is carried out. The simmer output in the form of a data frame makes it very easy to process the output further. The simple R API on Simmer will also make it easier to simulate.

How to cite this article:

I. G. A. A. Yudistira, R. Nariswari and S. Arifin., "OUTPUT VISUALIZATION FROM RESULT OF DISCRETE EVENT SYSTEM SIMULATION WITH 'simmer' R PACKAGE," *BAREKENG: J. Math. & App.*, vol. 17, iss. 1, pp. 0581-0592, March 2023.

1. INTRODUCTION

Simulation models are used to study a system, especially if it is difficult / expensive or dangerous to study the system directly [1]. Besides that, real systems are complex, almost impossible to study with analytical models, for that simulation is the last option that can't be bargained anymore. Furthermore, Shannon [2] provides a definition of simulation, which is a process for designing a model of a real system and conducting experiments on the model, for the purpose of understanding the behavior of the system or evaluating various strategies or a set of criteria. Simulations are run with various scenarios, to study various obstacles and find solutions to the studied system. The simulation results are usually in the form of a data set, which need to be analyzed further in order to obtain the important information, Stark [3] states that: "Data visualization gives us a clear interpretation of what information means by providing a visual context through maps or graphs". This makes the data more natural for the human mind to understand and therefore makes it easier to identify the presence of trends, patterns and outliers in larger data sets. Data visualization is needed because the human brain is not capable of taking so much raw, unorganized information and turning it into something usable and understandable. Visualization is usually in the form of graphs, charts and summary tables [4].

R is a functional programming language, which is based on open source (open source), so it can be accessed for free. The R language is very well known among scientists and statisticians, even reaching its use in various applied fields. Among academics, R is widely used to assist in the computational aspects of solving research problems. In terms of simulation studies, R has the power of static stochastic simulation techniques. The main weakness of R before 2017, was the unavailability of a package to solve reliable discrete event simulation (DES) problems. So programming DES using R becomes very difficult. Since 2017 developed package `simmer` which is a DES package for R that allows high-level process-oriented modeling [5], in line with other modern simulators. The simmer package makes it easy to build discrete-event simulation models in R [6]. The package is designed as a generic yet reliable process-oriented framework, written in C++, so execution is relatively fast and robust. The simmer package also has automatic monitoring capabilities, with the default output in tabular form. This package also provides a rich and flexible API for R, centered around the concept of paths, common in simulation models for entity paths. Function $trajectory()$ used to generate an object with class *trajectory*. However, the technique for building simmer-based DES models has not yet been developed, so it is effective for representing complex systems [7], [8].

The R program has excellent visualization capabilities [9]. There are four main R packages for graphics processing, namely (1) *graphics* packages, which are automatically loaded during standard R installation. The graphics system produced by this graph is called the *base graphics* system; (2) the *grid* package provides low-level graphics functions, so this package does not completely plot graphs, but only provides additional graphic features; In addition to the basic graphics system, there are two important packages that provide high-level graphics functions and are based on the *grid* package, namely (3) the *lattice* package created by Deepayan Sarkar's, this package provides a better view than the basic graphics system, it just has a more complicated syntax; (4) the *ggplot2* package, an R chart system based on the *grid* package as well, created by Hadley Wickham, is similar in some respects to *lattice* charts, but with a fundamentally different basis and structure. In this study, the visualization of discrete event simulation results will be discussed, using both the basic graphics system (*graphics*), *grid* and *ggplot2*. The *dplyr* package will be used to generate statistics summary tables, to support visualization results, by utilizing both the basic graphics system (*graphics*), *grid* and *ggplot2*. The *dplyr* package will be used to generate statistics summary tables, to support visualization results, by utilizing both the basic graphics system (*graphics*), *grid* and *ggplot2* [10].

This study aims to 1) describe the various capabilities of R packages for visualization of discrete event simulation outputs, 2) develop user defined R functions, for visualizing simulation outputs using the *simmer* package. which is effective and can provide complete information in the form of trends, patterns or outliers. And 3) explore and interpret simulation outputs based on visualizations generated using homemade functions (objective 2). R, which has strength in statistical and graphical analysis, will be very interesting when combined with simmer and synergize with packages for visualization, especially *ggplot2* and *dplyr*. The *simmer* main competitors are *SimPy* and *SimJulia*, which were developed for Python and Julia, respectively. [11], [12].

2. RESEARCH METHODS

The method used in this study is the library method, with a descriptive and exploratory approach [7]. The steps in this research are:

1) Review the library to get a complete understanding of the R API (application programming interface). The R APIs for simmer, ggplot2 and dplyr are in the form of R functions;
2) Review the literature to get a complete understanding of the environmental structure of simmer, ggplot2 and dplyr classes;
3) Build a chart or flowchart that describes the system, and examine the results (output) of the simulation to be studied;
4) Build an R script that starts in its simplest form;
5) Increase the complexity of the system one level and beyond;
6) Is it sufficient to represent the system, if not go back to step 5);
7) Run the simulation and generate the desired output for further analysis;
8) Done

The modeling and simulation process presented in this paper starts from problem formulation to decision making. This research is limited only to the aspect of model development, especially the development of discrete event simulation models. Aspects of data collection, testing model assumptions and carrying out experiments on the model are outside the scope of this research [13].

3. RESULTS AND DISCUSSION

The simplest Discrete Event System is a queuing system with a single server (single queue). The description of the system is as follows: A system has a single server that serves the entity for a transaction. Yudistira introduces a chart to illustrate the system, which is as follows [7]:

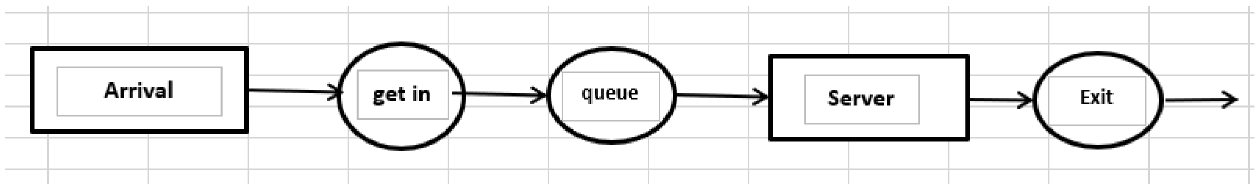

Figure 1. A Simple System (Single Server)

Figure 1. This chart depicts a trajectory of entities originating from an environment, with a certain interarrival time and generally a random variable with a certain probability distribution, for example exponential, with $rate = \lambda$ (arrivals / unit time) [14]. The entity enters the system to get service from the "Server", but if the "Server" is in a "busy" status, then the entity enters the queue, so there will be a change in the status of the queuing subsystem, namely the queue length is increased by 1 unit. The entity will remain in the queue, until it has a turn to be served. When the entity gets service (enters the service subsystem), then at that point in time there will also be a change in the status of the queue and "Server" subsystem, namely the queue length is reduced by 1 unit, and the "Server" status is "busy". Entity will get service for time t, which is also a random variable and as long as “Server” is running, no other entity will be served. So “Server” is controlled by that entity during time t. After the service is finished, the entity will leave the system and return to its environment. The exit of the entity from the system will trigger status changes in the system, that is, if there is no other entity that will be served next, then the "Server" status becomes "Idle", and the status of the number of entities that are still in the system, decreases by 1 unit [15].

3.1 Simulation Without Replication (Single Replication)

The previously discussed system is modeled with an R script. Then by using the *simmer* package, the generic form is as follows:

```
# R packages needed in this research
require(simmer); require(ggplot2; require(dplyr)
require(grid)
set.seed(seed)
env <- simmer("single_server")

# generate interarrival time (AK)
AK <- function() round(rexp(n = 1, rate = rate1),3)
# generate duration of server activity time (Lama)
Lama <- function() round(rexp(n = 1, rate = rate2),3)

# Trajectory
lintas <- trajectory() %>%
   seize("Server") %>%
   timeout(Lama) %>%
   release("Server")

# resource and entity generation
env %>%
   add_resource("Server", 1) %>%
   add_generator("ent", lintas, AK, mon = 2) %>%
   run(RUN) %>% invisible
```

There is in the last line there, is a $run(RUN)$ command, the RUN object in that command provides a simulation time limit. The simulation process is run only for $time \leq RUN$. However, entities that enter the system at $time \leq RUN$, but exit the system greater than RUN, are not considered by the script. In the case of real systems, what generally applies is the time limit for the entry of new entities into the system. Meanwhile, entities that are already in the system are guaranteed to get "Server" services until all of them leave the system.

Suppose the input parameters in the model are as follows, $rate1 = 0.40$; $rate2 = 0.45$; $RUN = 100$; $seed = NULL$. Then the simulation is run and the output of the arrival simulation of the entity is as follows:

```
# Simulation output for entity arrival state, with ongoing = TRUE
   out <- env %>%
   get_mon_arrivals(ongoing = TRUE)%>%
   subset(start_time > 0) %>%
      transform(serv_start_time = end_time - activity_time)
```

The function $transform(serv\_start\_time = end\_time - activity\_time)$ will add a new column with the name $serv\_start\_time$, which is the time the entity was served. The parameter of $ongoing = TRUE$ in $get\_mon\_arrival$, will record the last entities outside the RUN simulation range time. In this study, the RUN value is equal to 100 units of time is used, so that the simulation is only processed for the time value (in this case the $end\_time\ value$) is ≤ 100. So that the simulation output stored in the out object, which is written in the last lines, is as follows:

```
> out
    name start_time end_time activity_time finished replication serv_start_time
1   ent0      0.486   13.018        12.532     TRUE           1           0.486
2   ent1      4.096   15.688         2.670     TRUE           1          13.018
3   ent2      5.366   15.824         0.136     TRUE           1          15.688
                                            ...... dst ......
37 ent36     80.021   80.348         0.327     TRUE           1          80.021
38 ent37     80.948   91.062        10.114     TRUE           1          80.948
39 ent38     83.131   95.830         4.768     TRUE           1          91.062
40 ent39     85.112       NA            NA    FALSE           1              NA
41 ent40     85.295       NA            NA    FALSE           1              NA
42 ent41     92.779       NA            NA    FALSE           1              NA
```

Entities "$ent39$" to "$ent41$" (lines 40 to 42) have a value of $finished = FALSE$, meaning that in the simulation time range $RUN = 100$, the entity has not finished processing, so end_time (time out of the system) and $activity\_time$ (the length of time the entity is served) all three have NA values, in this simulation it happens that there are only three entities that have not been processed. It is possible that in the next simulations, more or less than three entities will have not been processed, or none of the entities have been processed. Note that the entities "$ent39$", "$ent40$" and "$ent41$", enter the system before 100 units of time, i.e., their $start\_time$ values are 85,112, 85,295 and 92,779, respectively. So the three entities are already in the system, so that this research will get the values of end_time, $activity\_time$, and $serv\_start\_time$, the three entities. The following script will get the value of end_time for the entity "$ent39$".

$row.names(out) <- 1: dim(out)[1]$
$px <- env \% > \% peek(Inf, verbose = TRUE)$
px

```
      time process
1 100.337   ent39
2 102.474   ent42
3 102.474     ent
```

The result of the script above is, the line numbers are sorted more regularly and most importantly we get the end_time for the entity "$ent39$", which is 100,337. It is also shown that "$ent42$" entered the system ($start\_time$) at the 102.474th time, so we do not count it because its arrival exceeds 100. The service start time for the "$ent39$" ($serv\_start\_time$) entity is when the previous entity "$ent38$" left the system, i.e. at end_time 95,830. This last time value will be the service start time ($serv\_start\_time$) "$ent39$", by obtaining the end_time and $serv\_start\_time$ values, the $activity\_time$ value can be obtained. The process for entities "$ent40$" and "$ent41$", is carried out in the same way so that all NA values are filled in. Click on the following link to see the full script, Appendix single.pdf . The final result is given as follows (only the last 7 lines are displayed).

```
    name start_time end_time activity_time finished replication serv_start_time
36 ent35     78.456   79.370         0.058     TRUE           1          79.312
37 ent36     80.021   80.348         0.327     TRUE           1          80.021
38 ent37     80.948   91.062        10.114     TRUE           1          80.948
39 ent38     83.131   95.830         4.768     TRUE           1          91.062
40 ent39     85.112  100.337         4.507    FALSE           1          95.830
41 ent40     85.295  104.864         4.527    FALSE           1         100.337
42 ent41     92.779  106.827         1.963    FALSE           1         104.864
```

3.2 Visualization of Simulation Results Without Replication

The $single$ function is a user-defined function, the full script for this function is in this link Appendix single.pdf. This function produces four graphs, namely "Utility", "Queue", "System", and "All". The "Utility" graph depicts the relationship between simulation time and server status ($0 = \text{“}idle\text{”}\ and\ 1 = \text{“}busy\text{”}$). The server status "$idle$" means the server is not serving, while the server status "$busy$" means the server is working. The graph is shown in **Figure 2** below **[16], [17]**.

.

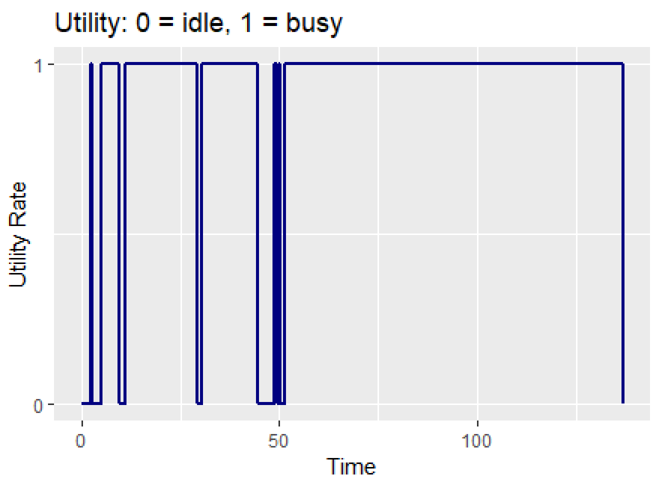

Figure 2. Utility Graph from Result of Function $single(graph = "Utility")$

Based on **Figure 2**, The area of this graph shows the utility level of the server. The second graph "Queue" depicts a *step* function, which relates the simulation time and queue length, the area under the curve for this graph is the queue rate. An image of the "Queue" graph generated by a *single* function, is given in **Figure 3**.

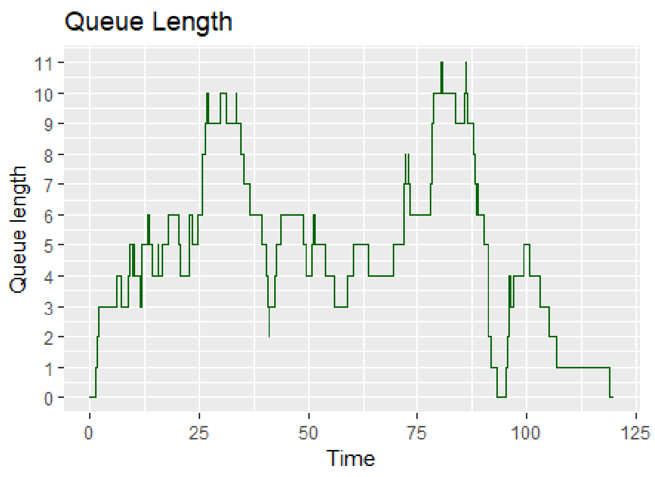

Figure 3. Graph from Result of Function `single(graph="Queue")`

Figure 4 is the relationship between the simulation time, and the number of entities in the current system. Just as before that the area under the curve, is the *flow rate*, or the flow rate of entities in the system. The "System" graph at a glance looks similar to the "Queue" graph, it is because the number of entities in the system is equal to the number of entities in the queue subsystem (shown by the "Queue" graph) plus one more entity being served by the server. The graph is shown by **Figure 4** below

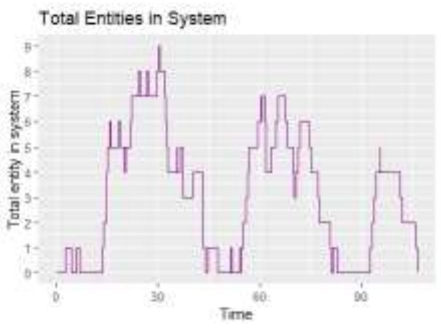

Figure 4. Graph from Result of Function $single(graph = "System")$

Figure 5 is the resulting fourth graph is named "All". The $single(graph = "All")$ command will generate all three graphs at once, in one field. The result is given in **Figure 5** below.

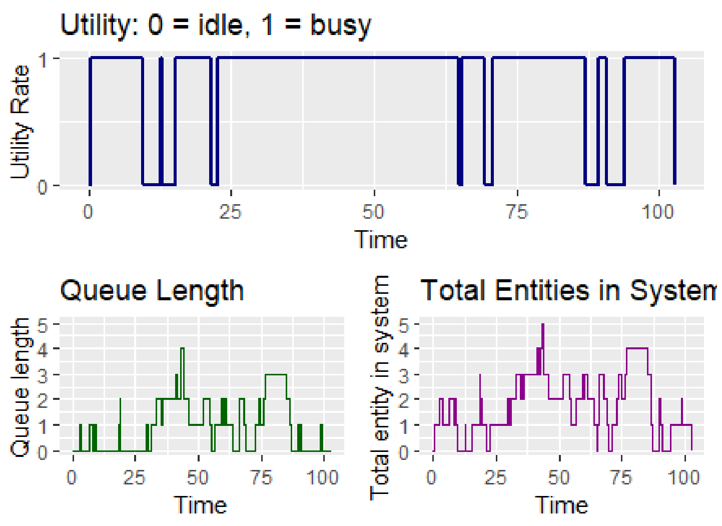

Figure 5. Graph from Result of Function $single(graph = "All")$

Figure 5 presents the single function in addition to generating graphs, also provides tables. There are four tables that are generated from this function, namely: 1) The out_going_output table, which is the output of the function that comes from the simmer package, namely $get\_mon\_arrival(ongoing = TRUE)$, it's just that we have modified the values for end_time, and $activity\_time$ have been assigned values for all entities with the variable $finished = FALSE$, i.e. $end\_time > RUN$ but $start\_time < RUN$. In addition, this table also produces three additional variables (columns), namely: $serv\_start\_time$, $flow\_time$, and $wait\_time$. The variable $serv\_start\_time$ is the time that the entity is start to served by the server, $flow\_time$ is the time the entity is in the system, and $wait\_time$ is the time spent by the entity in the queue. 2) The second table is named $resource\_output$, generated by the simmer $get\_mon\_resources()$ function. This table contains four important variables used to build the graph, namely: $time$, $server$, $queue$, and $system$. The $time$ variable records all simulation times that are less than RUN (beyond that time the entity cannot enter the system). The $server$ variable records the status of the server utility (0 and 1) at any point in time given by time. The $queue$ variable records the status of the queue length of the entity, at each point in the simulation time. The last is the $system$ variable, which records the status of the number of entities in the system at any point in time, given by time. 3) There are ten statistics values generated by this function namely, a) $queue\_rate$ is the queue rate per unit of time, b) statistics $utility\_rate$ measures the level of server utility (usage) per unit of time, c) $system\_last\_time$ measures the server downtime (latest time of time), d) avg_flow_time is the average time spent by the entity in the system, e) $avg\_waiting\_time$ is the average time spent by the entity to be in the queue, f) $longest\_flow\_time$ records the longest time (of all $flow\_time$) spent by an entity in the system, g) $longest\_waiting\_time$ is the longest time (of all $wait\_time$) spent by a entities to be in the queue, h) $average\_queue\_length$ is the average queue length, i) $avg\_entity\_in\_system$ is the average number of entities in the system during the simulation time span, j) $server\_downtime$ is the range between server downtimes ($system\_last\_time$) with RUN. The value of $server\_downtime$ is equal to 0, it means the last entity end_time value is $\leq RUN$. These ten statistics are summarized in tabular form, and there are two tabular forms that are given the first in the form of a width and named $wide\_stat\_value1$, an example of the results are as follows

```
$wide_stat_value1
# A tibble: 5 × 4
  name1              value1 name2                 value2
  <chr>               <dbl> <chr>                  <dbl>
1 queue_rate          3.88  longest_flow_time      22.9
2 utility_rate        0.944 longest_waiting_time   20.4
3 system_last_time  111.    average_queue_length    3.73
4 avg_flow_time      11.7   avg_entity_in_system    4.70
5 avg_waiting_time    9.39  server_downtime        11.2
```

4) The last table given by the $single$ function, is the long form table of the 10 statistics discussed earlier. This table is named $long\_stat\_value2$, an example of the result is given below:

```
$long_stat_value2
# A tibble: 10 × 2
   name                      value
   <chr>                     <dbl>
 1 queue_rate                3.88
 2 utility_rate              0.944
 3 system_last_time        111.
 4 avg_flow_time            11.7
 5 avg_waiting_time          9.39
 6 longest_flow_time        22.9
 7 longest_waiting_time     20.4
 8 average_queue_length      3.73
 9 avg_entity_in_system      4.70
10 server_downtime          11.2
```

3.3 Simulation with n Replications

The statistics generated from the simulation are estimators of system parameters, so to get the margin of error, we need to repeat it several times. This study develops a single server function with $n > 1$ replications. This function is named n_single, which is an expansion of the $single$ function previously discussed. The function arguments are also mostly the same, the use of the function is as follows:

$$n\_single(rate1 = 0.40, rate2 = 0.45, RUN = 100, n = 30, graph = c("None", "Utility", "Wait\_Time", "Flow\_Time", "Total\_Serv", "Avg\_Flow", "Avg\_Waiting", "Queue\_Rate", "Server\_Downtime", "Avg\_Queue\_Length"), seed = NULL)$$

This function provides nine graphical displays namely, from “Utility” to “Avg_Queue_Length”. The full script of the n_single function is in this link Appendix n_single.pdf. The argument value $n = 30$, is the default value for replication.

The n_single function will basically replicate the single function n times (repetitions), and for each replication it will be calculated including utility level ($utility\_rate$), waiting time in the queue ($wait\_time$), number of entities served in each replication (tot_serv), and others can be seen in the output of this function.

3.4 Visualization of Simulation Results with n Replication

The n_single function can generate 9 graphs, with the additional option $graph =$ “$None$” provided if the user does not need the graph display. The nine graphs are $graph =$ “$Utility$”, the choice of this graph argument will produce a line graph that connects the utility level ($utility\_rate$) of each replication with its replication. An example of a graphical demonstration is as follows:

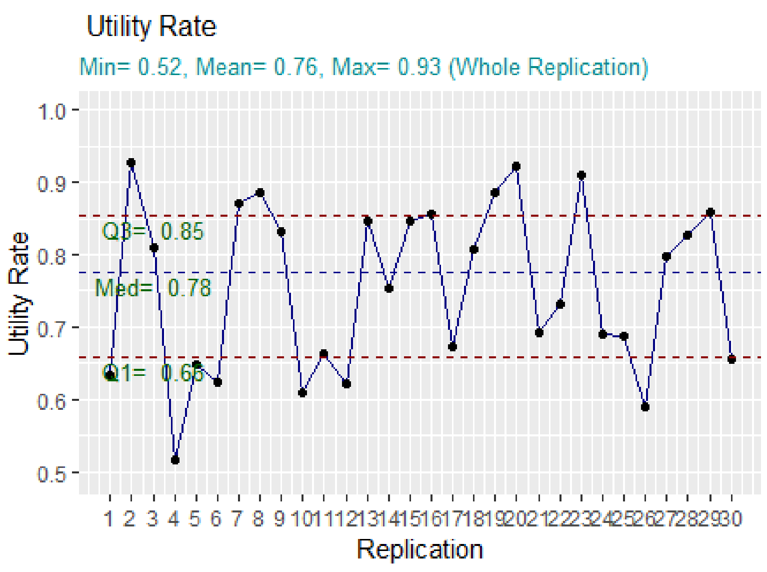

Figure 6. Graph from Result of Function $n\_single(graph = "Utility")$

This graph (**Figure 6**) also shows summary statistics of the $utility\_rate$ for all replications. This statistics summary includes Min (smallest value), $Q1$ (1st quartile), $Median$, $Mean$, $Q3$ (3rd quartile) and Max (largest value).

The second graph is a box plot, which shows a box and line graph for the $wait\_time$ values for each replication. An example of a graphical demonstration of this simulation output is shown in **Figure 7**.

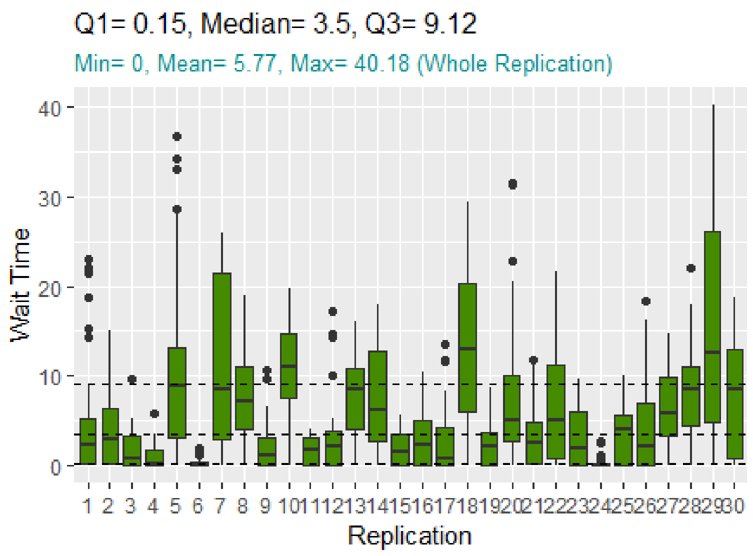

Figure 7. Graph from Result of Function $n\_single(graph = "Wait\_Time")$

Figure 7, It can be seen from the "Wait_Time" graph that the variation of waiting time for each replication is not the same, but most of the waiting times for the entire replication (75%) are less than 9.12 units of time, while the median and average are 3.5 and 5.77 time units, respectively. In addition to “Wait_Time”, another box-line graph generated by this function is the “Flow_Time” graph, which relates the time an entity spends in the system with its replication.

The “Total_Serv” graph generated by the n_single function is a bar graph, which shows the number of entities served in each replication, in the simulation time range ($start\_time < RUN$). An example of the simulation output for this graph is,

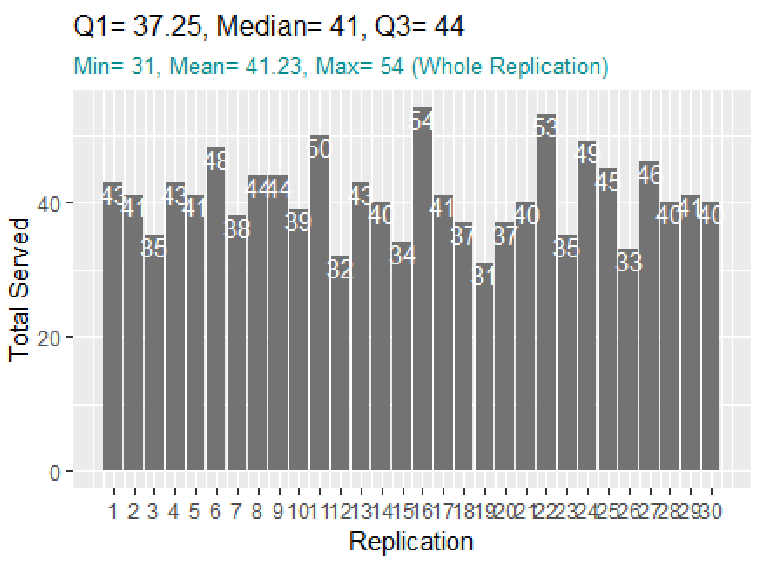

Figure 8. Bar Graph from Result Of Function $n\_single(graph = "Total\_Serv")$

In the graph shown by Figure 8 above, it can be read that there were 43 entities served in replication 1, and the lowest was 31 entities in replication 19, while the highest were 54 entities that occurred in replication 16. Other graphs can be seen directly by running this n_single function.

The n_single function, in addition to generating graphs, also generates four types of tables, which are provided whenever needed for further processing. The four tables are, 1) The $arrival\_out$ table, where the variables are the same as the out_going_output table, which is generated by the $single$ function. An example of the simulation output for this table is:

```
$arrival_out
# A tibble: 1,237 × 9
# Groups:   replication [30]
   name  start_time end_time activity_time finished replication serv_sta…¹ flow_…² wait_…³
   <chr>      <dbl>    <dbl>         <dbl> <lgl>          <int>      <dbl>   <dbl>   <dbl>
 1 ent0       0.232     5.27         5.04  TRUE               1      0.232    5.04   0
 2 ent1       1.87      6.41         1.14  TRUE               1      5.27     4.55   3.41
 3 ent2       6.10      7.33         0.92  TRUE               1      6.41     1.24   0.315
 4 ent3       6.8      13.7          6.38  TRUE               1      7.33     6.91   0.533
 5 ent4      11.8      16.4          2.66  TRUE               1     13.7      4.57   1.91
 6 ent5      18.4      22.1          3.62  TRUE               1     18.4      3.62   0
 7 ent6      20.7      22.9          0.85  TRUE               1     22.1      2.20   1.36
 8 ent7      22.6      24.9          1.96  TRUE               1     22.9      2.24   0.274
 9 ent8      22.9      29.2          4.32  TRUE               1     24.9      6.27   1.95
10 ent9      26.0      29.9          0.751 TRUE               1     29.2      3.98   3.23
# … with 1,227 more rows, and abbreviated variable names ¹serv_start_time, ²flow_time,
#   ³wait_time
# ℹ Use `print(n = ...)` to see more rows
```

2) The $resources\_out$ table, which is similar to the $resource\_output$ table that generated by the $single$ function, only the $resources\_out$ table is expanded for n replications.

3) The $statistics\_out$ table that generates statistics values for each replication. There are eleven important statistical values given, among others, $total\_serv$, avg_flow_time (average time spent by entity in the system), avg_queue_length (average queue length). The complete table is given as follows:

```
$statistics_out
# A tibble: 30 × 12
# Groups:   replication [30]
   repli…¹ total…² avg_f…³ avg_w…⁴ longe…⁵ longe…⁶ last_…⁷ serve…⁸ queue…⁹ utili…˟ avg_q…˟
     <int>   <int>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
 1       1      43   20.9    17.9     36.2    35.1    133.    33.4     5.77   0.983    5.62
 2       2      41    8.77    6.54    26.7    25.1    112.    12.0     2.39   0.817    2.22
 3       3      35    5.53    3.36    15.4    11.0    108.     7.76    1.09   0.704    1.43
 4       4      43   17.4    14.9     34.6    32.0    108.     7.96    5.94   0.976    5.99
 5       5      41    7.14    4.86    20.2    19.8    120.    19.7     1.67   0.781    1.98
 6       6      48   37.0    34.2     57.3    52.9    153.    53.3    10.7    0.869   11.7
 7       7      38    4.65    2.38    14.9    13.9    106.     5.53    0.856  0.819    1
 8       8      44   13.5    10.9     31.7    28.3    117.    17.4     4.07   0.987    3.84
 9       9      44   13.9    11.4     26.4    24.6    113.    12.5     4.46   0.965    4.68
10      10      39    9.37    7.29    23.9    22.6    103.     2.51    2.77   0.789    3.24
# … with 20 more rows, 1 more variable: avg_entity_in_system <dbl>, and abbreviated
#   variable names ¹replication, ²total_serv, ³avg_flow_time, ⁴avg_wait_time,
#   ⁵longest_flow_time, ⁶longest_waiting_time, ⁷last_time, ⁸server_downtime, ⁹queue_rate,
#   ˣutility_rate, ˣavg_queue_length
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```

4) The last table is rsc_out, which only contains 5 columns namely, $replication$, $time$, $queue$, $server$, and $system$. This table is provided when necessary to generate a step graph, as examples are shown in Figures 2, 3, and 4. The table looks like this (only 10 rows are shown out of a total of 2505 rows):

```
$rsc_out
# A tibble: 2,505 × 5
   replication   time queue server system
         <int>  <dbl> <dbl>  <dbl>  <dbl>
 1           1  0         0      0      0
 2           1  0.232     0      1      1
 3           1  1.87      1      1      2
 4           1  5.27      0      1      1
 5           1  6.10      1      1      2
 6           1  6.41      0      1      1
 7           1  6.8       1      1      2
 8           1  7.33      0      1      1
 9           1 11.8       1      1      2
10           1 13.7       0      1      1
# … with 2,495 more rows
# ℹ Use `print(n = ...)` to see more rows
```

4. CONCLUSIONS

1. There are two user define R functions resulting from this research, the first one is named $single$, which is stored in the *singleDES*.R file. This function aims to display the dynamics of the status of the queue length of the entity, the number of entities in the system and the status of the "*idle*" or "*busy*" server, throughout the simulation time from the starting point of the simulation (time point 0) to the last entity leaving the system. This graph, of course, only displays the dynamics of the system status for one simulation only. In connection with the graph, this function is also used to display statistics that are used to estimate system parameters, including queue level ($queue\_rate$), server utility level ($utility\_rate$), system downtime ($system\_last\_time$), the time span between the end of the simulation time – in this paper denoted by RUN - and the server downtime (in this case it also means the system downtime), which is then named server downtime ($server\_downtime$), there are six more simulation results statistics generated by this function. Other statistics according to user requirements, can be generated by reprocessing the dataset generated by this function, which is named out_going_output. This dataset is generated from the simmer $get\_mon\_arrival$ function, with a slight modification, namely, continuing simulation for entities that enter the system less than the RUN simulation time. Then the $resource\_output$ dataset, which gives the system state dynamics related to the servers (resources),

2. The second function produced in this study is named n_single, which is an extension of the $single$ function by using replication up to n times. This function aims to visualize the simulation output, from a system with a single server which is repeated up to n times. The graphs generated by this function include; 1) "Utility" graph, which displays a line graph, for the utility level for each replication along with summary statistics including smallest value (*Min*), 1st quartile (*Q1*), *median*, *mean*, 3rd quartile (*Q3*), and largest value (*Max*). This statistics summary is an overview of the overall replication. 2) "Wait_Time" graph, this graph displays a boxplot of the time an entity spends in the queue for each replication, and also displays a summary of those wait time statistics for the entire replication. 3) the "Total_Serv" graph, is a bar graph of the number of entities served in each replication, and like any other graph it also displays summary statistics for the entire replication. There are six more graphs that this function can create, along with summary statistics to give a better picture of the state of the system. As with the $single$ function, the n_single function is also equipped with the ability to generate datasets, which can be further processed to obtain other desired graphs

according to needs. The datasets are $arrival\_out$ and $resources\_out$, which are similar to those generated by a single function, only extended for n replications. The $statistics\_out$ dataset provides statistical values for each replication such as total entities served, average number of entities in the system and queues, utility level and there are seven more statistics displayed in this dataset. The rsc_out dataset is designed to be used to display step graphs, such as graphs generated by the $single$ function, but by selecting the desired replication. The graphs produced by both the $single$ function and the n_single function are more effective with the help of the *ggplot2* package, while the *dplyr* package is used for data manipulation.

ACKNOWLEDGMENT

I would like to thank my institution, Bina Nusantara University, for funding BRUN's internal grant, so that this paper can be created.

REFERENCES

[1] D. N. Banks, Jerry, John Carson II, Barry Nelson, *Discrete-Event System Simulation*, 5th Editio. Edinburgh Gate, England: Pearson Education Limited, 2014.

[2] R. E. Shannon, “Introduction to the art and science of simulation,” in *1998 winter simulation conference. proceedings (cat. no. 98ch36274)*, 1998, vol. 1, pp. 7–14.

[3] M. Stark, “Why Data Visualization Is Important.” 2020.

[4] H. A. A. Jiangjun Tang, George Leu, “Discrete Event Simulation,” *Simulation and Computational Red Teaming for Problem Solving*. pp. 121–142, Oct. 18, 2019. doi: https://doi.org/10.1002/9781119527183.ch7.

[5] S. B. Ucar I, “simmer.plot: Plotting Methods for simmer. R package version 0.1.5,” 2017. http://r-simmer.org/extensions/plot.

[6] I. Ucar, B. Smeets, A. Azcorra, U. Carlos, and I. I. I. De Madrid, “simmer: Discrete-Event Simulation for R,” vol. 90, no. 2, 2019, doi: 10.18637/jss.v090.i02.

[7] I. G. A. A. Yudistira, “Pengembangan Simulasi Kejadian Diskret Berbasis Paket Simmer pada R,” *Eng. Math. Comput. Sci. J.*, vol. 3, no. 2, pp. 79–85, 2021, doi: 10.21512/emacsjournal.v3i2.7386.

[8] J. P. Lander, *R for everyone: Advanced analytics and graphics*. Pearson Education, 2014.

[9] W. Chang, *R graphics cookbook: practical recipes for visualizing data*. O’Reilly Media, 2018.

[10] W. N. Venables, D. M. Smith, and R. C. Team, “An introduction to R, Notes on R: A Programming Environment for Data Analysis and Graphics Version 3.6. 3.” R Foundation for Statistical Computing Vienna, Austria, 2020.

[11] D. Ziniviev, *Discrete Event Simulation. It’s Easywith SimPy!* PragPub, 2018. [Online]. Available: https://www.researchgate.net/publication/322949363_Discrete_Event_Simulation_It’s_Easy_with_SimPy

[12] P. Van Der Paelt, B. Lauwens, and B. Signer, “A Transparent Data Persistence Architecture for the SimJulia Framework”.

[13] M. C. Sachs and E. E. Gabriel, “Event History Regression with Pseudo-Observations: Computational Approaches and an Implementation in R,” *J. Stat. Softw.*, vol. 102, no. 9 SE-Articles, pp. 1–34, May 2022, doi: 10.18637/jss.v102.i09.

[14] A. Ebert, P. Wu, K. Mengersen, and F. Ruggeri, “Computationally Efficient Simulation of Queues: The R Package queuecomputer,” *J. Stat. Softw.*, vol. 95, no. 5 SE-Articles, pp. 1–29, Oct. 2020, doi: 10.18637/jss.v095.i05.

[15] T. A. Syahputri, T. S. Az-zahra, N. A. Setifani, K. P. Ningrum, and D. Rolliawati, “PEMODELAN DAN SIMULASI PROSES PRODUKSI PERALATAN BAYI PADA HOME INDUSTRI PUPPY PUTRA PERDANA,” *JUST IT J. Sist. Informasi, Teknol. Inf. dan Komput.*, vol. 11, no. 1, p. 24, Oct. 2020, doi: 10.24853/justit.11.1.24-31.

[16] M. Tollefson, “Introduction: plot (), qplot (), and ggplot (), Plus Some,” in *Visualizing Data in R 4*, Springer, 2021, pp. 3–7.

[17] X. G. Baqués, A. Mosca, B. Rondelli, and G. R. Fort, “Roman Open Data: A semantic based Data Visualization & Exploratory Interface,” *Arqueol. y Téchne Métodos formales, nuevos enfoques Archaeol. Techne Form. methods, new approaches*, p. 18, 2022.