Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

WEBGPSS: THE FIRST TWO HOURS OF SIMULATION EDUCATION

Richard G. Born

College of Business
Northern Illinois University
DeKalb, IL 60115, U.S.A.

Ingolf Ståhl

Department of Managerial Economics
Stockholm School of Economics
S-11383 Stockholm, SWEDEN

ABSTRACT

In this paper we present seven short lessons used for introducing management science students to discrete event simulation. It has been used both as the only element of such simulation in courses that devote only two classroom hours to this topic and as the introduction in courses that are devoted almost completely to simulation.

1 INTRODUCTION

The authors have been teaching discrete event simulation for together almost four decades to beginners. Most of the courses have used GPSS in some form or other. During the last few years both of us have used the WebGPSS system, available both at the site <www.webgpss.com> and as a stand-alone system for Windows (Born 2003, Ståhl *et al.* 2003).

One of us has written a conventional text book for this (Ståhl 2003) and one has designed an electronic equivalent with over 500 PowerPoint slides (Born and Ståhl 2003, Schriber *et al.* 2003). Although most of our teaching has been in longer courses, some teaching of simulation has also been as a small part of a general course in e.g. operations research, .e.g as introduction to queueing. The following seven lessons have then proved to be useful in courses when only two classroom hours have been available for discrete event simulation. They have also been used for such purpose by other teachers, e.g. in Latvia. The seven lessons have also proved to be a good start in longer courses.

During these two first hours we gradually build up a model of a simple one server system, where customers arrive at random and wait for service in a first come, first served queue. We shall here start by first studying just how customers arrive at and leave a system. In Lesson 1, we assume that customers arrive 18 minutes apart at a booth to buy tokens for a turnstile. Each customer buys one token. When 50 tokens have been bought, the turnstile is closed down, since there are then no tokens left.

2 LESSON 1

When we open WebGPSS, e.g. at www.webgpss.com, we see that the screen, shown in Figure 1, is divided into two parts, one larger to the right, the *block diagram* field, initially empty, and one to the left, the *Symbol menu,* with 18 GPSS block symbols. We build our programs by clicking on these symbols.

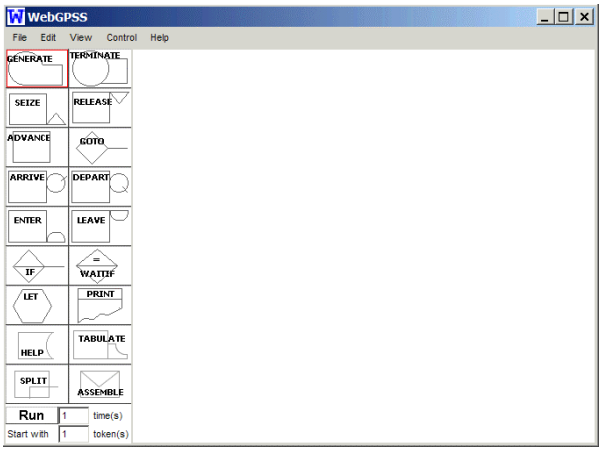

Figure 1: The WebGPSS Application Window

The GENERATE symbol in the upper left-hand corner is the first block used in every GPSS program. With this block we bring customers **into** the simulation program. When we click on the GENERATE symbol in the symbol menu we get a copy of this symbol in the block diagram part to the right. There is then also a cross below this GENERATE block indicating where the next block will be inserted in the block diagram.

For this program we also need a TERMINATE block to get the customers **out of** the system. We hence click on this symbol, to the right of GENERATE in the symbol menu. This TERMINATE block is now placed in the block diagram below the GENERATE block, where we earlier had the cross. This cross has moved down one step.

We have now created our first **segment** of blocks, as shown in Figure 2. A line connects the GENERATE and

TERMINATE blocks. We have now created a simple block diagram, which shows the logic of the program.

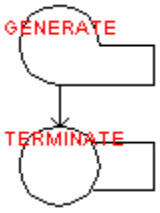

Figure 2: Block Diagram Showing the Program Logic

This gives the general structure of the program. We must next give values to the operands of the blocks. We double-click on the GENERATE block in the block diagram to open the dialog for the operands of GENERATE. Here we see all the operands of the block, as shown in Figure 3. In this first program it is enough to write a value in the first field in this dialog. Even if GENERATE has five operands, we need for this program to determine only the first operand, Average IAT. IAT means InterArrival Time, i.e. Average IAT is the average time between the arrivals of two subsequent customers.

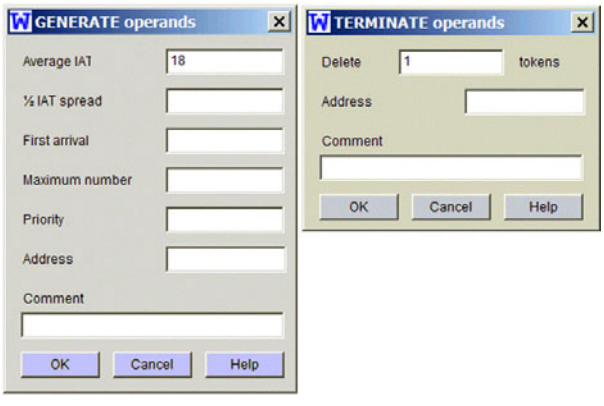

Figure 3: The GENERATE and TERMINATE Operands

Since we want all customers to come 18 minutes apart, we write 18 in the field on the first line. Since this is the only operand that we now want to input, we click on *OK*. Then the dialog is closed and we see 18 written inside the GENERATE block, as shown in the block diagram in Figure 4. Having given a value only to the first operand implies that we for our first programs assume that the time of the arrival of the first customer, T(1), is equal to IAT(1), i.e. the first Inter Arrival Time. This is the standard, i.e. default, assumption. Hence the first customer arrives after 18 minutes.

We next want to give an operand to the TERMINATE block. We double-click on the TERMINATE block to open the dialog for its operands. We see that TERMINATE has only one operand: the number of tokens to be taken away.

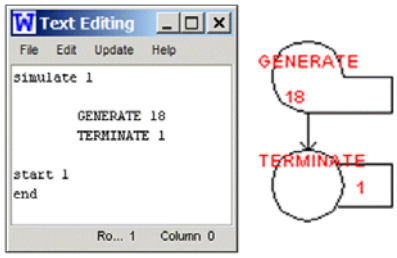

Figure 4: The Full Program and Block Diagram

This refers to the number of tokens that a customer removes going through the TERMINATE block. In the example with the booth each customer buys a certain number of tokens. Since we here assume that each customer buys only one token, we write 1 in the field on the first line, as shown in the right half of Figure 3. After clicking on *OK*, we see 1 inside the TERMINATE block, as shown in the block diagram in Figure 4.

We have now created the complete block diagram for the first program. We have also created a program in text format. We can see this program in its original form by clicking on Edit in the main menu and then on Text Editing. After clicking on *OK* in the box with the question "Do you want to replace this text?", we can in the Text Editing window see the full program, as shown in Figure 4.

Besides the two blocks, GENERATE 18 and TERMINATE 1, we see three control statements. Unlike **blocks** that refer to actions taken by customers and executed **every** time a **customer** enters the block, **control statements** generally refer to actions executed **once**, at the start or end of simulation. Every GPSS program has at least the following three control statements: SIMULATE, START and END.

SIMULATE comes first in each program and has as operand the number of times that one wants to run the program, i.e. the value given to the right of the Run button, close to the bottom of the symbol menu.

START always lies after all the blocks and has as operand the number of tokens that are available at the start of the simulation. This operand determines how the simulation is stopped. On the bottom line in the symbol menu we see that the default value is 1. For this first run, we allow it to remain unchanged. This gives us START 1, which means that we have only one token. Since this one token is removed already by the first customer, the program is terminated already by the first customer.

END, which lacks operands, is inserted automatically at the very end of each program to mark its end.

We next move to the Run button at the bottom of the symbol menu, with the text *time(s)*. In the field to the left of this text we have the default value 1. Since we initially want to run the program only once, we allow this value 1 to remain unchanged. When we click on the Run button we get a new window, a Results window. This contains two

sub-windows, one for the program list and one for the block statistics, as shown in Figure 5.

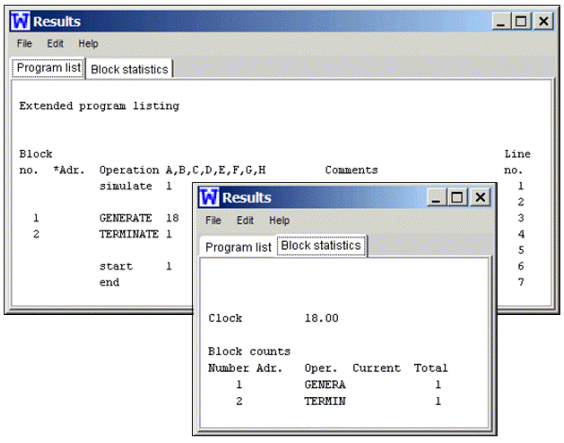

Figure 5: The Results Window with Program List and Block Statistics Tabs, when Starting with 1 Token

Program list contains an extended listing of the program. "Extended" implies that the GPSS system has added some features to the original program, which we saw in the Text Editing window. The program is listed in a neater format than the "free format" of the original file. Furthermore, at the right hand side, the statements are numbered. At the left hand side we have other numbers, referring to blocks. In this program there are only two blocks.

We next click on the tab with the text *Block statistics*, to get the only statistics provided by this program. We here first get information about the value of the simulation clock when the simulation is finished, namely 18. GENERATE 18 implies, as mentioned, that customer 1 comes at time 18. Since this customer removes the only token, the simulation is ended at time 18. We see furthermore, under *Total*, that one customer has gone through the GENERATE block and one customer through the TERMINATE block, i.e., exactly as expected.

We next change the program so that 50 customers go through the system. We write 50 in the field to the right of **Start** at the bottom in the symbol menu. We get the program we call **PRO01**. We click again on Run to get the Results window, shown in Figure 6. We see in the program listing that we now have START 50, i.e. that 50 tokens have to be removed before the simulation is stopped.

As mentioned, customer 1 comes after 18 minutes. When will the other customers arrive? The GENERATE block can schedule any desired number of customers, all with the IATs according to the distribution expressed by the operands, in this case the first operand. Thus, the arrival of customer 2 is at time T(2) = T(1) + IAT(2) = 18 + 18 = 36. Customer 3 will also arrive 18 minutes later, in this case at time 36 + 18 = 54. In this way the GENERATE block will schedule and bring additional customers into the system. As customer 1 becomes active at time 18, GENERATE schedules the arrival of customer 2 at time 36. When customer 2 gets into the system at time 36, GENERATE schedules the arrival of customer 3.

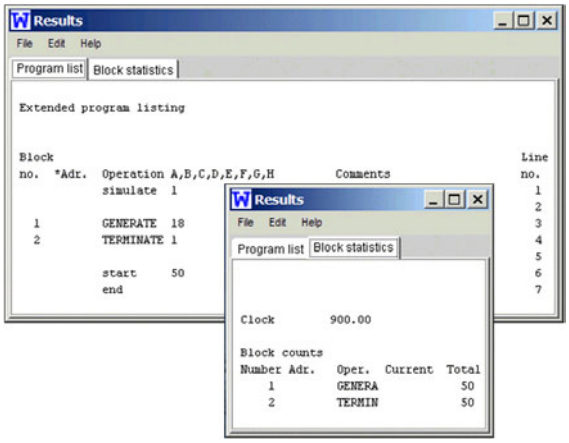

Figure 6: Results Window when Starting with 50 Tokens

In this way simulation could continue "forever" if it were not for the TERMINATE block. Each customer takes away one token. After the first customer 49 tokens remain; after the second 48 tokens and after 49 customers only one token remains. The 50th customer, arriving at time 50*18 = 900, takes the last token, The simulation then stops, with no tokens left. This is seen in the Block statistics, where the clock value at simulation stop is 900. Under *Total* we can see that 50 customers have gone through both GENERATE and TERMINATE.

3 LESSON 2

In program PRO01 above there was no uncertainty. There was always a distance in time of 18 minutes between each customer arriving at the turnstile. Since there is no uncertainty, the same output appears no matter how many times the program is run. We shall now modify program PRO01 by inserting **uncertainty**. We introduce randomness into the Inter Arrival Times of the customers, so that they do not always come 18 minutes apart, but instead between 12 and 24 minutes apart, as in Figure 7.

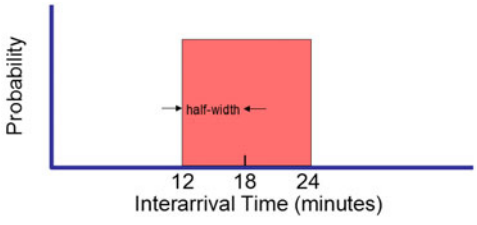

Figure 7: Uniformly Distributed Inter Arrival Times

The average is still 18 minutes, but the customers arrive 18 $\pm$ 6 minutes apart. The "half spread" is 6 minutes, since the total spread is 24 - 12 = 12 minutes. All times are

hence **uniformly** distributed, i.e. all times between the two extreme values of 12 and 24 are equally possible. This is the simplest form of assumption regarding randomness, but there are many other distributions in GPSS.

We double-click on GENERATE in the block diagram to open its dialog and we write 6 in the field for *1/2 IAT spread*, as shown in Figure 8. As regards this second operand the following should be noted. Since, e.g., customer 2 cannot come before customer 1, the time, IAT(2), between the arrival of the second and first customer cannot be negative. Hence, the lower limit of the possible IATs must not be negative and the **second** operand, called the **B operand**, must not be larger than the **first** operand, the **A operand**. If there is no B operand, like in PRO01, then there is an implicit, "default", value of 0, giving a constant IAT.

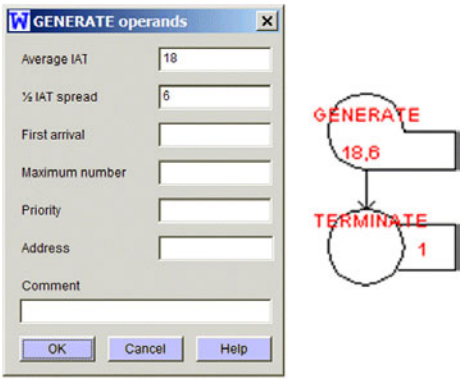

Figure 8: Introducing Random Inter Arrival Times

We next click on *OK* and see 18,6 as text in the GENERATE block with a **comma** between the operands, as shown in Figure 8. The comma **separates** the operands. Customers are generated with a time distance of between 12 and 24, with all times in between equally probable.

Since we now have randomness in the arrival times, it is no longer enough to run the new program, **PRO02**, only once. Every run will use a different sequence of random numbers. We write 3 for *Run 3 times* in the field at the Run-button and click on Run to run the program three times. In the results window, we first see in the program listing that we have SIMULATE 3 and GENERATE 18,6. We see in the block statistics, in Figure 9, the results from all three runs. The first one was finished when the simulation clock is 892.24; the second at 859.92 and the third time 900.63. Two observations can be made here:

Everyone who runs this program will get the same results. This has to do with the fact that GPSS, like all other simulation systems, does not use true random numbers, but instead *pseudo-random* numbers, i.e., artificial random numbers, which look random, but are determined by a computer program. A starting number called a seed initiates the sequence. Each number in the sequence is then obtained from the preceding number by a process of multiplication and division with certain constants. The use of this type of numbers has great advantages. For every run, a specific sequence of random numbers is used.

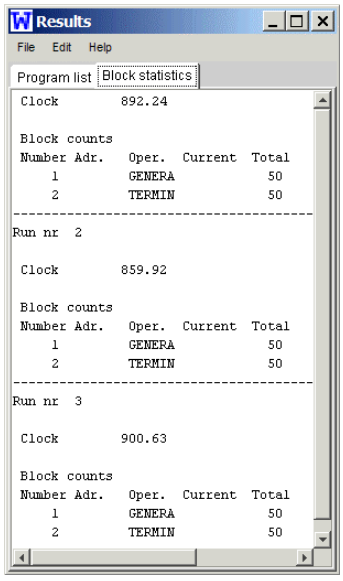

Figure 9: Block Statistics for Three Runs

Secondly, the different results in the three runs are connected with the fact that we in the three runs use different sequences of random numbers for determining the time between the arrivals of the 50 customers. By writing a number larger than 1 in the field after the Run-button, we can run the program several times with different results. We can regard each such result as a sample from a large number of possible results. If we want to draw conclusions from a simulation, the simulation program should **never** run just once, since one can then not estimate the variation of results due to differences in the random numbers.

4 LESSON 3

In the two earlier programs, the number of customers decided when the simulation should stop. In the next program, which we call **PRO04**, to keep the numbering in accordance with what is on the Web, the simulation will instead stop at a specific time, namely after 8 hours, but customers will still arrive 18 $\pm$ 6 minutes apart. We build on program PRO02 above. The first change is that no customer determines when the simulation is stopped. A customer hence no longer takes away any token. We open the dialog of TERMINATE and delete 1 in the field regarding the number of tokens to take away. After *OK* to close the dialog, there is no number in the TERMINATE block.

Furthermore, we have in this program not only a customer segment, but also a stop segment, as shown in Figure 10. If we now did not have the stop segment, the simulation would go on "forever", since the customers would not decrease the number of tokens and each customer schedules

the arrival of the next customer. We can think of a janitor who comes after 8 hours and closes the turnstile. In order to have a block generating a janitor to the right of the customer segment, we must first insert a cross for the next block under the TERMINATE block by clicking below this block. We next click on the GENERATE symbol in the symbol menu, so that the new GENERATE block is placed to the right of the old one in the block diagram. The janitor must also leave the system. We hence also click on the TERMINATE symbol in the symbol menu. We now also have a stop segment.

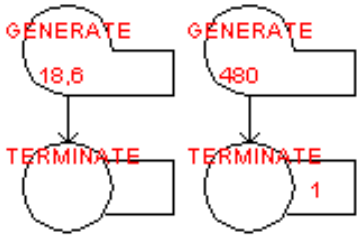

Figure 10: Customer and Stop Segments

Since it is confusing to talk about a customer in the case of the stop segment, we shall use the more general term **transaction** to denote the **temporary** entities generated in the GENERATE blocks. We have two kinds of transactions in this program: customers and a janitor.

We next give operands to the blocks in this stop segment. Since we used minutes in the customer segment, we must use minutes also in the stop segment. In GPSS no specific unit of time is specified. All units of time are possible, but one must use the **same** unit of time consistently throughout the whole program. 8 hours must hence be stated as 480 minutes. We click on the new GENERATE block to the right and write 480 as Average IAT. In this case, when the janitor comes after exactly 480 minutes, there is no spread and there is no B operand in this GENERATE block. If nothing else is stated, the janitor arrives at the first sampled or determined IAT-time, in this case exactly 480 minutes after simulation start.

Since the janitor closes the system after 480 minutes, i.e. already the first time that he arrives, the janitor's TERMINATE block must remove tokens. If we start with only one token, it will be enough to have the janitor remove this only token to stop the simulation. We hence double-click on the TERMINATE block and write 1 in the field on the line for number of tokens to be removed and click on *OK*. As the last step, we change 50 in the field for Start to 1, since it is enough to start with one token, which will then be removed by the janitor to stop the simulation.

We shall next run this program 3 times. We then see in the block statistics that the simulation is stopped after 480 minutes in each of the three runs and that in the first two runs 27 customers arrived, but only 26 in the third run.

5 LESSON 4

We shall now change the turnstile of program PRO04 above into a museum. In the turnstile, the customers left the system the same moment as they arrived. In the museum of **PRO05**, we allow them to stay in the simulated system for a time that varies at random between 20 and 30 minutes, as shown in Figure 11. We call this a museum, since we do not have any limit on the number of visitors who can be inside the museum system at the same time.

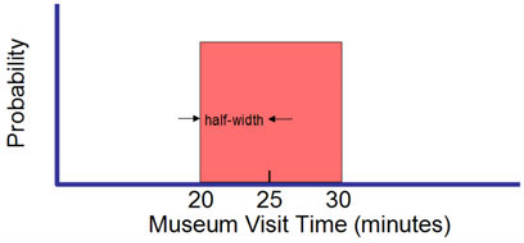

Figure 11: Uniformly Distributed Museum Visit Times

We use a new block, the ADVANCE block, for delaying the visitors between 20 and 30 minutes, i.e. with an average time of 25 minutes and a half spread of 5 minutes. We insert such an ADVANCE block between the first GENERATE and TERMINATE blocks. We click on the line connecting these two blocks in the customer segment. The cross of where the next block shall be inserted is then marked on this line. We click on ADVANCE in the symbol menu and a rectangular ADVANCE block is inserted between the GENERATE and the TERMINATE blocks in the customer segment, as shown in Figure 12.

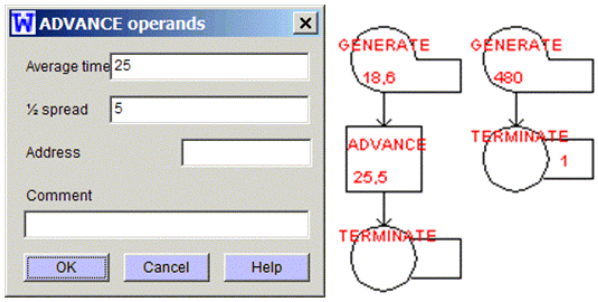

Figure 12: Introducing Random Museum Visit Times

We next click on this ADVANCE block to open its dialog, as shown in Figure 12. We see two operands here: Average time and 1/2 spread. We write 25 as *Average time* and 5 as *1/2 spread*. After *OK*, we see 25,5 inside the ADVANCE block. This implies that all times between 20 and 30 minutes are equally likely, since the two operands, the A and B operands, have the same function as GENERATE. The default values for the A and B operands are likewise 0.

We run the program twice. The program list contains no surprises. The new thing is in the block statistics, shown in Figure 13. Earlier we had only data in the left-most column, *Total*, on how many transactions in total passed through each block. Now we also have data in the middle column with the heading *Current*. For the third run we see in block 1 (GENERATE) under *Total* that 27 visitors have been created, but in block 3 (TERMINATE) that only 25 visitors have left the museum. We see in block 2 (ADVANCE), under *Current*, that 2 visitors are still inside the museum at closing time. They are visitors who planned to leave after time 480, but who are now forced out earlier. This shows that there is no specific limit on how many visitors can be in the ADVANCE block at the same time.

```
Clock        480.00

Block counts
Number Adr.   Oper.  Current  Total
    1         GENERA            27
    2         ADVANC     2      27
    3         TERMIN            25
    4         GENERA             1
    5         TERMIN             1
```

Figure 13: Part of Block Statistics of Museum Program

6 LESSON 5

We have up to now only dealt with the temporary entities, transactions, such as customers. We now introduce the other important type of entities, the permanent **servers**. We here study the simplest kind of server, called a **facility**, which can give service to only **one transaction** at a time.

We shall remake PRO05, our museum program, to a program that deals with a small barbershop with one barber, Joe, who can cut the hair of obviously only one customer at a time. Customers arrive according to the same time pattern as before, i.e. with a time distance of between 12 and 24 minutes. The time for a haircut varies between 20 and 30 minutes, with 25 minutes on average, the same times as the visitors were in the museum in PRO05.

We first insert a new block, a SEIZE block, before the ADVANCE block. We insert the SEIZE block in the same way that we inserted the ADVANCE block earlier. We next insert a RELEASE block after the ADVANCE block. The RELEASE block implies that the customer makes the barber free to serve other customers. We obtain the diagram for this new program, **PRO07**, as shown in Figure 14.

The SEIZE and RELEASE blocks can be seen as mirror pictures of each other. SEIZE has, to its right, a triangle pointing upwards, while RELEASE has a triangle pointing downwards. As seen in the symbol menu, there are in GPSS several other symbols that come in such pairs. This makes it simpler to follow the logic in block diagrams.

We next add operands to the two new blocks, as shown in the operand windows in Figure 15.

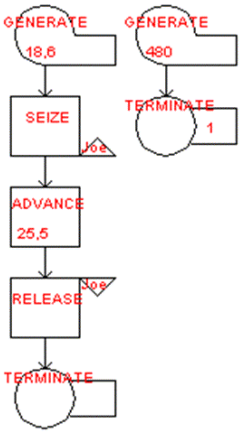

Figure 14: Block Diagram of Program PRO07

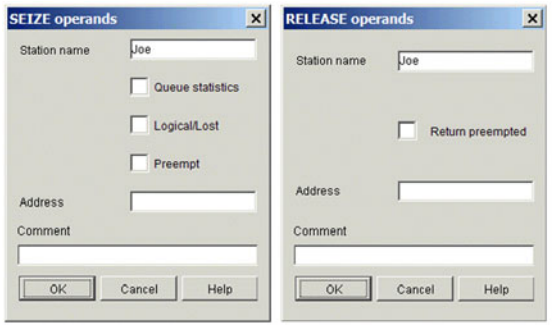

Figure 15: Dialogs of SEIZE and RELEASE

We open the dialog of the SEIZE block. At the top, we have *Station name*. This refers in this case to the name of the barber. We write Joe and click for *OK*. We see Joe written at the bottom of the triangle of the SEIZE block. A server name must have 3 – 6 characters, the first three letters and the remaining characters, if any, letters or digits. We also open the dialog of RELEASE, where we also find a field *Station name*. We write the same name as in the SEIZE block, i.e. Joe. After *OK,* we see Joe written at the top of the downward pointing triangle of RELEASE.

In spite of its simplicity, this program is not trivial, since the results are not self-evident. Since customers arrive on average 18 minutes apart and a haircut takes 25 minutes on average, the waiting line will obviously increase during the day. It is, however, not clear what the longest waiting line during the day is. Suppose our barber has four chairs for waiting customers and wants to find out if this number of chairs is sufficient. We run the program once with the results shown in Figure 16.

In the block statistics, we see under *Total* in blocks 1 and 2, (GENERATE and SEIZE), that 27 customers have arrived at the barbershop, but that only 19 have started getting a haircut by closing time. Under *Current* in the GENERATE block (block 1), there are 8 persons waiting for a haircut. Hence four chairs are not enough.

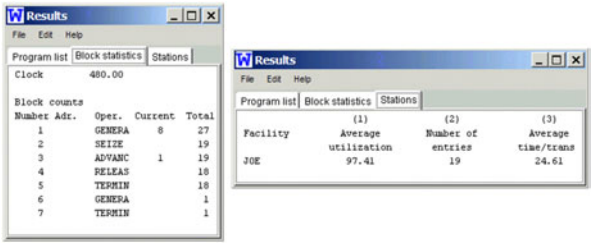

Figure 16: Barbershop Program Results

The persons who are waiting to get served are in the block statistics waiting in the block **preceding** the SEIZE block, but are also put on an internal waiting list, invisible to the user. While customer 1 can get a haircut without waiting, the barber is busy when customer 2 arrives and tries to "seize" Joe. Customer 2 will then have to wait. In the block statistics, he will not be admitted into the SEIZE block, but must wait in the block preceding SEIZE, here GENERATE. When customer 1 has got a haircut and "releases" Joe, Joe turns to the waiting customer 2, who will now be permitted to go through the SEIZE block into the ADVANCE block to get his haircut. He then goes through the RELEASE block, which never refuses entry. GPSS automatically determines who waits and who gets into service, on a first come, first served basis, unless another discipline is indicated. Figure 16 shows that we also get a tab with statistics for *Stations* with three data items:

1. *Average utilization,* 97.41, is the percent of the total simulation time that Joe was busy.
2. *Number of entries* is the number of times that customers have started to get service from the station.
3. *Average time/trans*, 24.61, the average time that each customer is served, is the total time that Joe was busy, divided by the number of entries, 19.

7 LESSON 6

The results of PRO07 showed the length of the waiting line at the **end** of the simulation, but there was no information on the waiting line during the simulation. To get more detailed queuing statistics, we open the dialog of the SEIZE block. In Figure 15, the line under *Station* name in the Seize operands dialog reads *Queue statistics*, followed by a small box. By clicking to get a "mark" here, we "switch on" the collection of queue statistics. After *OK*, we see a Q in the lower right-hand corner of the SEIZE rectangle. We run this program **PRO08** once.

In the program list, we now see that the only difference to PRO07 is that the second block now reads SEIZE Joe,Q. The block statistics and the stations statistics are exactly the same as for program PRO07. The adding of Q to the SEIZE block thus does not influence the actual simulation process. The Q only gathers statistics. We also find a new subwindow named *Queue/AD*, as shown in Figure 17.

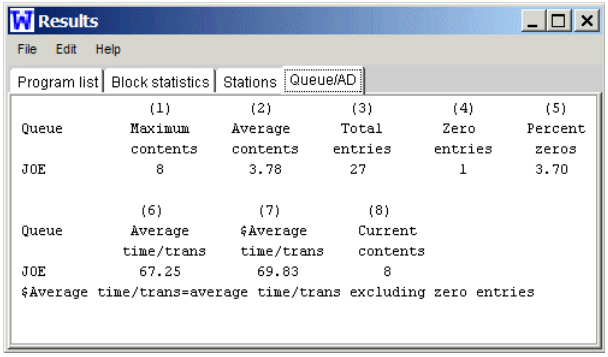

Figure 17: Barbershop Program Results Queue Statistics

We here find eight information items:

1. *Maximum contents*, 8, which is the highest number of customers that ever waited during the simulation.
2. *Average contents*, 3.78, is total time that customers waited, divided by the total simulation time, 480.
3. *Total entries*, 27, is the number of customers who have joined the queue.
4. *Zero entries*, 1, is the number of customers who did not have to wait at all, in this case customer 1.
5. *Percent zeros*, 3.70, is the number of zero entries, here 1, as a percent of total entries (point 3).
6. *Average time/trans*, 67.25, is total time customers have been waiting, divided by the number of entries.
7. *$Average time/trans*, 69.83, is the average time for those customers that really had to wait. We here refer to the 26 (=27-1) customers who had to wait.
8. *Current contents*, 8, is the number of customers, who were waiting when the simulation ended.

Since customers arrive with an average IAT of 18 minutes, but a haircut takes on average 25 minutes, the waiting line will grow. The question is, if it is only because of this time discrepancy that we get a waiting line. To answer this question, we open the dialog of the ADVANCE block and change the average time for a haircut to 18 minutes, i.e. the same time as the average IAT, and run the program once. We see already in the block statistics, under *Current* for block 1, a waiting line, but it is much shorter, of just 1 person. We see in the queue statistics that average waiting time has decreased, but still amounts to almost a quarter of an hour. The number of customers who did not wait at all has only increased to two. 93 percent of the customers still had to wait.

The question is then if it is the random variation in arrival and service times that causes the remaining queues. To answer this, we change our program so that there are no random variations. In the GENERATE block of the customer segment, we delete 6 as the 1/2 IAT spread and in the

ADVANCE block we delete 5 as the 1/2 spread to get GENERATE 18 and ADVANCE 18. Running this program once, we see in the block statistics that no one waits in block 1 and in the queue statistics that the waiting time is 0, and that the percentage of customers who did not wait reached 100. The waiting line in the preceding program was hence only caused by the random variations. This stresses the importance of using random variables when studying systems subject to noticeable random variations.

8 LESSON 7

The queue statistics obtained this far are in many cases insufficient, since they only refer to **averages**. One is often more interested in **extreme** values, like how many waited more than two hours. We might need a table of these waiting times, distributed into different time classes, e.g. of how many waited 10-20 minutes,.., 110-120 minutes, etc. In GPSS we obtain such statistics by using a **queue table**.

We open program PRO08 and go to the top menu and click on *Control*. In the submenu then obtained, we click on *Queue tables* and get a queue tables dialog, where we click on the *Add* button. We get a smaller dialog, which shows for which queues one can get a queue table. In this case there is only one queue, the queue in front of Joe. After a click on *OK,* Joe is written at the top to the left in the main field of the queue tables dialog. At the right-hand side of the dialog we see three fields for data to be input:

Class 1 top. It refers to the upper limit of class 1. We write 0 here. This implies that all who waited 0 minutes, i.e. not at all, are shown in class 1.

Width, i.e. class width, is the number of time units in each class (except the lowest and highest). Here, where we want the classes 0-10, 10-20,.. minutes, we write 10.

# of classes, i.e. number of classes, is the maximum number of classes we wish to have in the table. We set this to 20, which is the maximum number allowed.

When we have written these three values, we click on the *Set* button. The three values are then brought into the first line of the large main field, as shown in Figure 18.

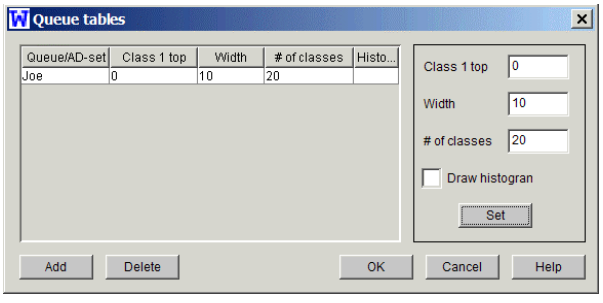

Figure 18: The Queue Tables Dialog

By clicking on OK, we complete the new program and run it once. We then see in the program listing that immediately after SIMULATE we have the statement QTABLE Joe,0,10,20. This is the only difference compared to program PRO08. We also see that the block, station and queue statistics are all the same as for PRO08. The new thing is that we now have a Results window tab called *Queue table* with new statistics, as shown in Figure 19.

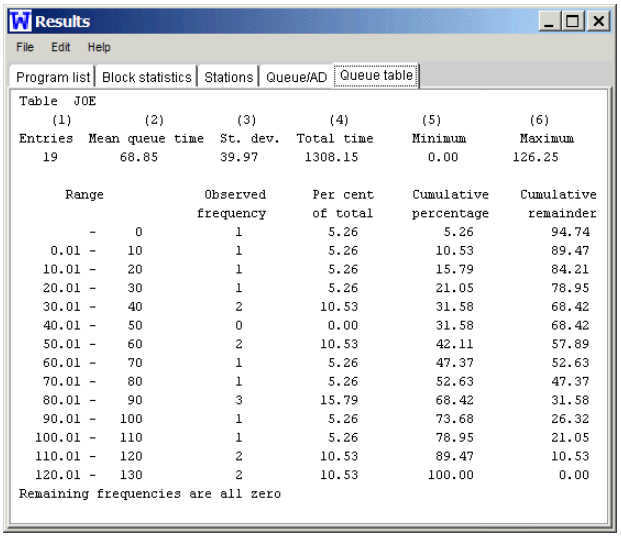

Table JOE

(1) Entries	(2) Mean queue time	(3) St. dev.	(4) Total time	(5) Minimum	(6) Maximum
19	68.85	39.97	1308.15	0.00	126.25

Range	Observed frequency	Per cent of total	Cumulative percentage	Cumulative remainder
- 0	1	5.26	5.26	94.74
0.01 - 10	1	5.26	10.53	89.47
10.01 - 20	1	5.26	15.79	84.21
20.01 - 30	1	5.26	21.05	78.95
30.01 - 40	2	10.53	31.58	68.42
40.01 - 50	0	0.00	31.58	68.42
50.01 - 60	2	10.53	42.11	57.89
60.01 - 70	1	5.26	47.37	52.63
70.01 - 80	1	5.26	52.63	47.37
80.01 - 90	3	15.79	68.42	31.58
90.01 - 100	1	5.26	73.68	26.32
100.01 - 110	1	5.26	78.95	21.05
110.01 - 120	2	10.53	89.47	10.53
120.01 - 130	2	10.53	100.00	0.00

Remaining frequencies are all zero

Figure 19: The Queue Table Tab of the Program Results

At the top we have a line with six data:

1. *Entries*, 19. Since a table produced by QTABLE refers only to those customers that have **finished** waiting, this number is different from the number of entries in the ordinary queue statistics, 27, the number of customers that have **started** waiting.
2. *Mean queue time*, 68.85, is the average waiting time for the customers that have finished waiting.
3. *St(andard) dev(iation),* 39.97, is a measure for describing the variations in the waiting times in this table.
4. *Total time,* 1308.15, is the total time that all 19 customers in the table have waited.
5. *Minimum*, 0, is the shortest waiting time, in this case of the first customer.
6. *Maximum*, 126.25, is the longest waiting time.

Under this line we have the table with five columns:

Column 1, *Range*, shows that the first class refers to a waiting time of 0 minutes, the second one to 0.01 - 10 (i.e. 0.000001 - 10), the third one to 10.01 - 20, etc.

Column 2, *Observed frequency*, shows the number of customers in each class.

Column 3, *Per cent of total*, shows the number in column 2 as a percent of the total number of customers, 19.

In column 4, *Cumulative percentage*, the percentage figures in column 3 are added cumulatively to make it easier to answer a question like "What percent of the customers waited **at most** 20 minutes? (Answer: 15.79 percent).

Column 5, *Cumulative remainder,* is 100 **minus** the value in column 4, to answer e.g. "What percent waited **more than** an hour?" (Answer: 57.89 percent.)

Although we in the queue table dialog asked for 20 classes, the table contains only 14. The highest one shown runs from 120.01 to 130. Since there are no observations above 130, the remaining classes are not displayed.

We can also produce a histogram for the queue table statistics. We click on *Control* and *Queue tables* and in the box for *Draw histogram*, to mark this box. After clicking on *Set*, we see *yes* in the column for Histogram to the right. We click on *OK* to confirm and close the dialog. We now run this program again. We see in the Results window yet another tab, *Histogram 1*. We click on this tab and we see a frequency histogram, as shown in Figure 20. On the scale at the bottom, the **limits** of the classes are written.

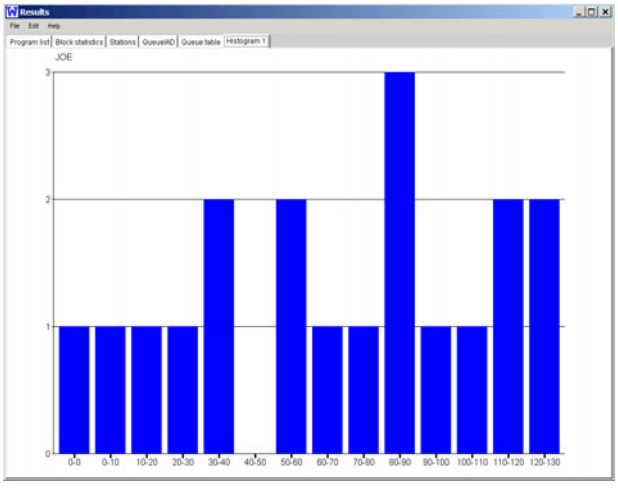

Figure 20: Histogram for the Queue Table Statistics

9 CONCLUSIONS

We have above presented a mini-course, which consists of seven brief lessons. This has been used as the introduction to simulation, either as the first two hours in a course focused on simulation or as the only two hours available for discrete-events simulation in a more general course. We have shown how one with a simple GUI can rapidly build up an easily understood program, which is shown both in a block diagram form and as pure text code and which produces different types of tables and graphs. We have also shown that one with very compact programs (e.g. 7 blocks and 4 control statements) can analyze service problems that are not trivial. We have also shown how one with simple simulation models can exemplify ideas that are fundamental for understanding the formation of waiting lines in service systems.

REFERENCES

Born, R. 2003. Teaching discrete-event simulation to business students: the alpha and omega. In *Proceedings of the 2003 Winter Simulation Conference*, ed. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, 1964-1972. Piscataway, NJ: IEEE.

Born, R. and I Ståhl. 2003. *WebGPSS Slide Presentation*. DeKalb, IL: Available on request from R. Born at <rborn@niu.edu>.

Schriber, T., I. Ståhl, J. Banks, A. Law, A. Seila, and R. Born. 2003. Simulation textbooks – old and new (panel). In *Proceedings of the 2003 Winter Simulation Conference*, ed. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, 1952-1963. Piscataway, NJ: IEEE.

Ståhl, I. 2003. *Simulation made simple with WebGPSS – a tutorial*. Stockholm: Stockholm School of Economics.

Ståhl, I., H. Herper, R. Hill, C. Harmonosky, J. Donohue and D. Kelton. 2003. Teaching the Classics of Simulation to Beginners. In *Proceedings of the 2003 Winter Simulation Conference,* ed. S. Chick, P.J. Sánchez, D. Ferrin, and D.J. Morrice, 1941-1951. Piscataway, NJ: IEEE.

AUTHOR BIOGRAPHIES

RICHARD G. BORN is an Associate Professor of Management Information Systems in the Department of Operations Management and Information Systems in the College of Business at Northern Illinois University. He has taught simulation modeling for the past 13 years to university students at all levels from undergraduate to graduate, including M.S. students in Management Information Systems, M.S. students in Accountancy, and M.B.A. students. His email address is <rborn@niu.edu>.

INGOLF STÅHL is a Professor at the Stockholm School of Economics, Stockholm, and has a chair in Computer Based Applications of Economic Theory. He was visiting Professor, Hofstra University, N.Y., 1983-1985 and leader of a research project on inter-active simulation at the International Institute for Applied Systems Analysis, Vienna, 1979-1982. He has taught GPSS for twenty-five years at universities and colleges in Sweden and the USA. Based on this experience, he has led the development of the micro-GPSS and WebGPSS systems. His email address is <ingolf.stahl@hhs.se> and the web address for his WebGPSS is <http://www.webgpss.com/>.