Proceedings of the 2011 Winter Simulation Conference
S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, eds.

INTRODUCTION TO SIMULATION

Ricki G. Ingalls

Oklahoma State University
322 Engineering North
Stillwater, OK, USA

ABSTRACT

Simulation is a powerful tool if understood and used properly. This introduction to simulation tutorial is designed to teach the basics of simulation, including structure, function, data generated, and its proper use. The introduction starts with a definition of simulation, goes through a talk about what makes up a simulation, how the simulation actually works, and how to handle data generated by the simulation. Throughout the paper, there is discussion on issues concerning the use of simulation in industry.

1 DEFINITION OF SIMULATION

Simulation, according to Shannon (1975), is "the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system." According to this definition, a simulation can be a discrete-event simulation, as we will discuss in this paper. Many people who attend this conference will be familiar with the term "MRP simulation." This is a model (actually a copy) of the real system (the MRP system of record) on which experiments (or scenarios) can be run to evaluate various strategies (such as how to respond to a drastic change in the forecast). Although we do not teach courses in our curriculum on "MRP simulation," a MRP simulation is no less a simulation than the type of simulation we will discuss in this tutorial.

The difference, and the power, of discrete-event simulation is the ability to mimic the *dynamics* of a real system. Many models, including high-powered optimization models, cannot take into account the dynamics of a real system. It is the ability to mimic the dynamics of the real system that gives discrete-event simulation its structure, its function, and its unique way to analyze results. So, to take liberties with one of my mentors in this field, we will say that simulation is the process of designing a *dynamic* model of an actual *dynamic* system for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system.

Throughout this paper, we will be referencing ideas and thoughts from Shannon (1975), Law and Kelton (2000), Banks et al. (2000), Kelton, Sadowski, and Sadowski (2001), Ingalls and Kasales (1999), Ingalls (1998), and Ingalls and Eckersley (1992).

2 A DRIVE-THROUGH EXAMPLE

In order to give us a reference model for discussion purposes, let us consider the example of a drive-through window at a fast food restaurant whose logic is shown in Figure 1. As a car enters from the street, the driver, who we will call Fred, decides whether or not to get in line. If Fred decides to leave the restaurant, he leaves as a dissatisfied customer. One of the great things about a simulation is that it is easy to track these type of customers. In most real-world systems, it is usually difficult to track customers who leave dissatisfied.

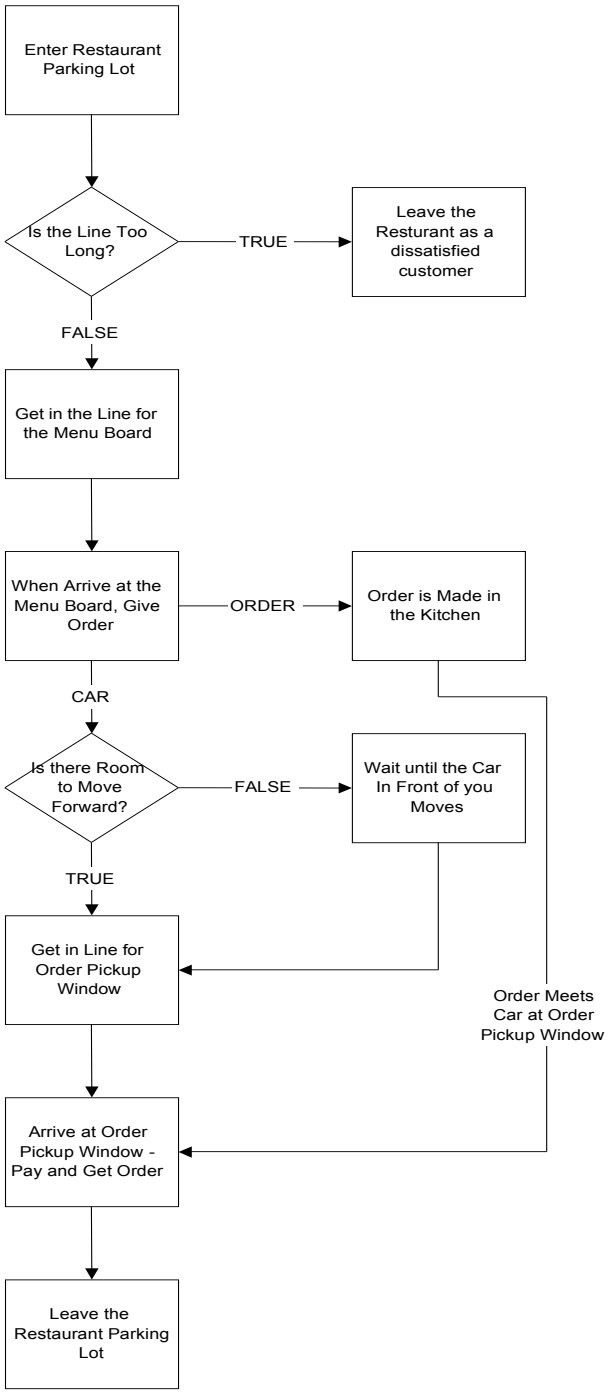

Figure 1: Drive-Through Example Logic Flow

If Fred decides to get in line, then he waits until the menu board (with the speaker no normal human being can understand) is available. At that time, Fred gives the order to the order taker.

After the order is taken, then two things occur simultaneously. (1) Fred moves forward if there is room. If there is no room, then he has to wait at the menu board until there is room to move forward. As

soon as there is room, he moves so the next customer can order. (2) The order is sent electronically back to the kitchen where it is prepared as soon as the cook is available.

As soon as Fred reaches the pickup window, then he pays and picks up his food, if it is ready. If the food is not ready, then Fred has to wait until his order is prepared. In this drive-through, Fred never hears, "Please pull forward and we will bring your order out to you." That is one of the reasons that Fred and I both like this place.

As soon as Fred parts with his money and gets his food, he then leaves the pickup window a satisfied customer.

3 DISCRETE-EVENT SIMULATION STRUCTURE

Although there are various flavors and paradigms in discrete-event simulation, there has evolved a basic structure that is used by most simulation packages. Regardless of how complex a discrete-event simulation package may be, it is likely to contain the basic components that we will describe in this section.

The structural components of a discrete-event simulation include *entities, activities and events, resources, global variables,* a *random number generator, a calendar, system state variables* and *statistics collectors*.

3.1 Entities

The best way to understand the function of an entity is understand that *entities cause changes in the state of the simulation*. Without entities, nothing would happen in a simulation. As a matter of fact, one stopping condition for a simulation model is the condition where there are no active entities in the system.

Entities have *attributes*. Attributes are characteristics of a given entity that are unique to that entity. Attributes are critical to the understanding of the performance and function of entities in the simulation.

In our example, the primary entity type is the cars coming to the restaurant. Also, there is a second entity type created when the order is taken, and that is the order itself. It has a relatively short life in the simulation, lasting only from the time the order is taken until it meets back up with the car at the pickup window.

We also have two attributes in our simulation. The first one is the time of day that the car enters the restaurant parking lot. We will call this attribute *StartTime*. The second is the value of the order. We will call this attribute *OrderValue*. Both of these are unique to the customers who are represented in the simulation.

Another example of an entity is the part that is flowing through a factory. Entities that represent parts in a factory could be created randomly or according to a schedule. A common attribute would be the time that the part started in the factory. Each part would have a unique time that it started in the factory. It may also have other attributes such as priority, the type of part, and cost incurred to produce the part. In a normal factory simulation that is tracking each individual part, it would not be unusual to have thousands of entities active in the simulation simultaneously.

Entities, however, do not need to be parts in a manufacturing facility or cars in a drive-through. It is also a common practice to use entities to represent the flow of *information*. This information could be a customer order, an email alert, a packet in a computer network, etc. It can be anything non-physical that causes a change in the status of system. These entities also have attributes. For example, an entity representing a packet in a computer network would have attributes such as packet size, packet destination, the time that the packet would time-out, etc.

3.2 Activities and Events

Activities are processes and logic in the simulation. *Events* are conditions that occur at a point in time which cause a change in the state of the system. An entity *interacts* with activities. Entities *interacting with* activities *create* events.

There are three major types of activities in a simulation: *delays, queues and logic*. The *delay* activity is when the entity is delayed for a definite period of time. In our example, there are three delays. The first is when Fred is ordering at the menu board, the second is when the order is being cooked in the kitchen, and the third is Fred picks up his order at the pickup window. In general, the length of time for a delay is either constant or is randomly generated. At the point that the entity starts the delay, an event occurs. This event schedules the entity on the *calendar* (which we will get to later). If the delay is for *d* time units, then the entity is scheduled to complete the delay *d* time units after the current time of the simulation. At that time, the delay expires and another event is generated.

Queues are places in the simulation were entities wait for an unspecified period of time. Entities can be waiting on *resources* (which we will get to later) to be available or for a given system condition to occur. *Queues* are most commonly used for waiting in line for a resource or storing material that will be taken out of the queue when the right conditions exist. In our example, there are three queues, the first is the part of the line that waits on the menu board to be available, the second is the order waiting on the kitchen becoming available, and the third is the part of the car line waiting on the order pickup window to become available. All three of these queues are waiting for *resources* to be available.

Logic activities simply allow the entity to effect the state of the system through the manipulation of state variables (which we will get to later) or decision logic. The first of several logic activities in our example is the decision whether or not to get in the order line in the first place. This decision is effected by the length of the line in front of the menu board.

3.3 Resources

In a simulation, resources represent anything that has a restricted (or constrained) capacity. Common examples of resources include workers, machines, nodes in a communication network, traffic intersections, etc. In our example, we have three different resources. The first resource is the menu board, which only one car can use at a time. The menu board is utilized from the time a car moves in front of it until the car moves away from it. In our model, the car does not automatically move away from the menu board after the order has been placed. There must also be room to move forward. So, this resource is occupied for both productive time (when the order is being placed) and unproductive time (when there is not enough room to pull the car forward). The second resource is the kitchen. It is utilized from the time an order arrives until the order is finished cooking. The third resource is the order pickup window. This resource can also be unproductive. That occurs when the car has arrived at the window, but the order is not ready from the kitchen yet. In the course of the simulation, we can track the key statistics of each of these resources, including utilization and costs.

It should also be noted that very complex resources can be utilized in a simulation. In a manufacturing simulation, conveyors are a very complex resource that many simulation packages offer. Also, transportation options such as trucks are offered as resources. A third complex resource is a vat or container that has a (continuous) flow of material both in an out of the resource. Depending on the target market of the simulation package, many complex resources are available to use.

3.4 Global Variables

If you are a programmer, then the idea of having *global variables* is nothing new. A *global variable* is a variable that is available to the entire model at all times. A global variable can track just about anything that is of interest to the entire simulation. In our model we have four global variables, two of which help us configure the problem and two of which collect revenue information. The two variables that help us configure the problem are the length of the line allowed at the restaurant. The first is *MenuBoardLineSize*, which gives the maximum line size for the menu board, including the car who is at the menu board. The second is *OrderPickupLineSize*, which gives the maximum line size after the menu board, including the car at the pickup window. By changing these two variables, the simulator can analyze the effectiveness of different line configurations.

The two variables that help use collect revenue information are *Revenue* and *LostRevenue*. *Revenue* is the total amount of revenue collected in the simulation, and *LostRevenue* is the total amount of revenue lost because the customer thought that the line was too long.

3.5 Random Number Generator

Every simulation package has a random number generator. The random number generator (technically called a pseudo-random number generator) is a software routine that generates a random number between 0 and 1 that is used in sampling random distributions. For example, let us assume that you have determined that a given process delay is uniformly distributed between 10 minutes and 20 minutes. Then every time an entity went through that process, the random number generator would a generate a number between 0 and 1 and evaluate the uniform distribution formula that has a minimum of 10 and a maximum of 20. As an example, let us assume that the generated random number is 0.7312, then the delay would be 10+(0.7312)*(20-10) = 17.312. So the entity would delay for 17.312 time units in the simulation. Everything that is random in the simulation uses the random number generator as an input to determine values.

In our example model, we will use a physical random number generator, two dice. The probability distribution for the roll of two dice is seen in Figure 2. In our model, the roll of two dice becomes the basis for every random delay and randomly assigned value in the model. We have five randomly assigned delays and values in the model, (1) the time between arrivals of cars to the restaurant, (2) the value of the order for the car, (3) the delay at the menu board, (4) the delay at the kitchen, and (5) the delay at the order pickup window. The formulas for these random values are as follows:

1. Time between arrivals of cars to the restaurant = (Dice * 10) seconds
2. The value of the order for the car = (Dice * 2) – 2 dollars.
3. The delay at the menu board = (Dice * 10) seconds.
4. The delay at the kitchen = (Dice * 8) seconds.
5. The delay at the order pickup window = (Dice * 10) seconds.

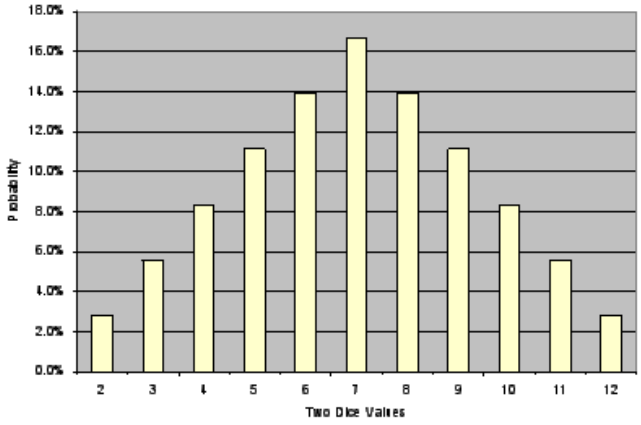

Figure 2: Distribution of Rolling Two Dice

3.6 The Calendar

The *calendar* for the simulation is a list of events that are scheduled to occur in the future. In every simulation, there is only one calendar of future events and it is ordered by the earliest scheduled time first. In a later example, it will become more clear how the calendar works and why it is important in the simulation. At this point, just remember that, at any given point in time, every event that has already been scheduled to occur in the future is held on the calendar.

3.7 System State Variables

Depending on the simulation package, there can be several system state variables, but the one system state variable that every simulation package has is the current time of the simulation. In order to keep from offending any simulation vendors, we will choose a different name for our simulation time variable. Our name of the current time of the simulation is *CurrentTime*. This variable is updated every time an entity is taken from the calendar.

3.8 Statistics Collectors

Statistics collectors are a part of the simulation that collects statistics on certain states (such as the state of a resources), or the value of global variables, or certain performance statistics based on attributes of the entity. There are three different types of statistics that are collected, *counts, time-persistent,* and *tallies.* Counts are very straightforward, they *count*. In our model, we count the number of *Lost Customers* because the line was too long. *Time-persistent* statistical collectors give the time-weighted values of different variables in the simulation. A common variable to track is the utilization of a resource. In our model, we collect 6 different time-persistent statistics, which are the number of busy resources of the three resources that we have in the model and, the number of entities in each of the three queues that serve the three resources. *Tally statistics* are collected one observation at a time without regard to the amount of time between observations. In our model, we collect a very common statistic, which is the amount of time that an entity stays in the system. Since we assign the value of *StartTime* to the attribute of the entity when it enters the restaurant parking lot, then the value *CurrentTime – StartTime* is the total amount of time in the system. If we are to improve this system, we would want to minimize this statistic without hurting the revenue too much.

4 A WALK THROUGH THESE CONCEPTS

What we are going to do now is walk through a couple of steps in our simulation model. As you will see, we will track the state of the system, the entities on the calendar, the values of attributes and state variables, and the statistics we are collecting.

4.1 The State of The System at Noon

Figure 3 shows the state of the system at noon during a lunch rush at our fast food restaurant. To help us design the line at the restaurant, we have set two of our global variables, *MenuBoardLineSize* and *OrderPickupLineSize*, both equal to 3. As you can see, that is the total capacity of the line at the restaurant. The state of our system at noon is as follows:

- Car 1 (the Corvette) is at the Order Pickup Window receiving its order.
- Cars 2 (the convertible VW Bug) and 3 (the minivan) are in line waiting for the Order Pickup Window.
- Car 4 (the 2-seat convertible) is ordering at the Menu Board.
- Cars 5 (the Police Car) and 6 (the Porsche) are waiting in line for the Menu Board.
- The Order for Car 2 is being cooked in the Kitchen.
- The Order for Car 3 is waiting in line for the Kitchen.
- Car 7 (of undetermined type) is schedule to arrive at the restaurant in the future.

The calendar of this system is made up of the entities that are scheduled to complete an activity with a specific time duration. These entities are Car 1, Car 4, the Order for Car 2, and Car 7. The calendar for these four entities is shown in Table 1.

Figure 3: State of the System at Noon

Table 1: The Calendar at Noon

Entity	Event	Event Time
Car(7)	Arrive at Restaurant	12:00:20 PM
Car(1)	Order Pickup Window Complete	12:00:40 PM
Order(2)	Kitchen Complete	12:00:56 PM
Car(4)	Menu Board Complete	12:01:10 PM

We also know the values of the attributes of the entities in the system. As you recall, we have two attributes, *StartTime* and *OrderValue*. The values for those attributes for each of the entities in the system is outlined in Table 2.

Table 2: Entity Attributes at Noon

Entity	StartTime	OrderValue
Car(1)	11:54:20 AM	$ 10
Car(2)	11:55:50 AM	$ 6
Car(3)	11:57:10 AM	$ 4
Car(4)	11:58:20 AM	$ 14
Car(5)	11:59:30 AM	$ 14
Car(6)	12:00:00 PM	$ 10
Car(7)	---	---

Other important information concerns our system state variable, *CurrentTime*, which is set to 12:00:00 PM.

The statistics that we are tracking in the simulation have the values listed in Table 3 as of noon. The "Time/Obs" column gives the amount of time that we have been collecting the statistic (which is since 11:00 AM) for time-persistent statistics or the number of observations for tally statistics.

Table 3: Statistics at Noon

Statistic	Value	Time/Obs
Revenue	$504	
Lost Revenue	$74	
MenuBoard Utilization	0.9984	1:00:00
Kitchen Utilization	0.7006	1:00:00
OrderWindow Utilization	0.9678	1:00:00
MenuBoard Waiting in Line	1.2306	1:00:00
Kitchen Waiting in Line	0.0822	1:00:00
OrderWindow Waiting in Line	1.0311	1:00:00
Time In System	5.5969	44

4.2 The State of The System at 12:00:20 PM

Here is where we get one of the key ideas about a *discrete-event simulation*. A discrete-event simulation moves the current time of the simulation through events and their timing instead of distinct time intervals. If we had distinct time intervals of 1 second, we would go through 20 time intervals before anything would happen. So instead of going through 20 time intervals with nothing happening, we go straight to the next scheduled event, which is the first event on the calendar, which is scheduled to occur at 12:00:20 PM. That event is the arrival of Car 7 to the restaurant.

When Car 7 arrives at the restaurant, the first activity in the simulation is to set its attributes. Obviously, *StartTime* is set to 12:00:20 PM, and the *OrderValue* is set using the formula (Dice * 2) – 2 after we roll the dice. The roll of the dice gives us a 9, so the value of the order is $16. After the attributes are assigned, Car 7 makes a decision whether or not to get in line to order. Since we have set the *MenuBoardLineSize* = 3 and there are 3 cars in the line, Car 7 leaves as a dissatisfied customer. When Car 7 leaves, we want to capture some important information, namely how much revenue have we lost because of dissatisfied customers, which is tracked in our global variable *RevenueLost*. *RevenueLost* is incremented from $74 to $90.

Now that Car 7 has arrived at (and subsequently, left) the restaurant, we also need to schedule the arrival of the next car (Car 8) to the restaurant. The time between arrivals to the restaurant is set using the formula (Dice * 10) in seconds. So, we roll the dice and get a 7. Car 8 will arrive 70 seconds in the future at time 12:01:30 PM. Car 8 is now put on the calendar with the event "Arrive at Restaurant" that will occur at the event time of 12:01:30 PM.

So what has changed? Car 7 has arrived and left and Car 8 is schedule to arrive in the future. We have also updated *RevenueLost*. We have a new calendar, which is shown in Table 4.

The statistics can be updated at this point as well. (However, most simulation packages only update statistics when the value that is being tracked changes). We have two types of statistics, the counting of *Revenue* and *RevenueLost*, the resource utilization statistics and the queue length statistics. The counting statistics are simply the values of the variables we are tracking, and *RevenueLost* has gone to $90. The other statistics are time-dependent statistics that are time-weighted averages of a given value. As an example, let us calculate the new value of the average number of cars waiting in line for the menu board. At noon, the simulation had been running for one hour and the average value was 1.2306. From noon to 12:00:20 PM, the number of cars waiting in line for the menu board has been 2. So the new time-weighted average is ((1.2306 * 1:00:00) + (2 * 0:00:20)) / 1:00:20 = 1.234851. If we were to convert this formula to seconds, it would be ((1.2306 * 3600) + (2 * 20)) / 3620 = 1.234851. All time-dependent statistics are calculated in this way. The value for all of the statistics at time 12:00:20 PM is in Table 5.

Table 4: The Calendar at 12:00:20 PM

Entity	Event	Event Time
Car(1)	Order Pickup Window Complete	12:00:40 PM
Order(2)	Kitchen Complete	12:00:56 PM
Car(4)	Menu Board Complete	12:01:10 PM
Car(8)	Arrive at Restaurant	12:01:30 PM

Table 5: Statistics at 12:00:20 PM

Statistic	Value	Time/Obs
Revenue	$ 504	
Lost Revenue	$ 90	
MenuBoard Utilization	0.998409	1:00:20
Kitchen Utilization	0.702254	1:00:20
OrderWindow Utilization	0.967978	1:00:20
MenuBoard Waiting in Line	1.234851	1:00:20
Kitchen Waiting in Line	0.087271	1:00:20
OrderWindow Waiting in Line	1.036453	1:00:20
Time In System	5.5969	44

4.3 The State of The System at 12:00:40 PM

We want to take one more step in the simulation because we actually see the line move! Looking at Table 4, we know that the next scheduled event is that Car 1 finishes its time at the Order Pickup Window at time 12:00:40 PM. The first thing to do is set the *CurrentTime* to 12:00:40 PM. When Car 1 finishes its time at the Order Pickup Window, we know that Car 1 has paid for its order, and so *Revenue* must be increase by the *OrderValue* attribute of Car 1. So the value *Revenue* is increased from $504 to $514. We also need to calculate a new average *Time in System* statistic. The time in the system for Car 1 is the *CurrentTime – StartTime*, which would be 12:00:40 PM – 11:54:20 AM, which is 6 minutes and 20 seconds. Since we are collecting *Time in System* in minutes, the new average *Time in System* is ((5.5969 * 44) + 6.3333) / 45 = 5.613265. The other thing that occurs is that the *Order Pickup Window* resource is no longer utilized. Since the *Order Pickup Window* resources is available for other cars, the simulation will allocate the *Order Pickup Window* resource to the car that is first in the line that is waiting to use the *Order Pickup Window*. Our Car 2 (the convertible VW Bug) is first in line for the *Order Pickup Window*, so Car 2 is allocated the *Order Pickup Window* resource. Can Car 2 start picking up its order? No, it cannot. The reason is that the order for Car 2 is still in the Kitchen. So even though the *Order Pickup Window* resource is occupied, no productive work is going on. At the next step in the simulation, the order for Car 2 will complete in the Kitchen and Car 2 will be able to start the process of picking up its order. But for now, it just has to wait!

Since Car 2 has moved forward, Car 3 moves to first place in the line waiting for the *Order Pickup Window* resource. Car 4 is still occupies the *Menu Board* resource and is still in the process of giving its order. This process is scheduled to end at time 12:01:10 PM (See Table 4). Cars 5 and 6 do no change their position in the line and Car 8 is still scheduled to arrive at the restaurant at time 12:01:30 PM. The new state of the system is shown in Figure 4.

Figure 4: The State of the System at 12:00:40 PM

We also update our statistics at this time. As we described in at time 12:00:20 PM, we look at what has happened since the last time we updated the statistics in order to determine the new values for the statistics. So, what number will we use to calculate the new average number of cars waiting for the *Order Window*, 2 (the number in line at 12:00:20) or 1 (the number in line now)? The answer is 2 because from time 12:00:20 PM to 12:00:40 PM there were 2 cars waiting in line. The new values for the statistics are shown in Table 6.

Table 6: Statistics at 12:00:40 PM

Statistic	Value	Time/Obs
Revenue	$514	
Lost Revenue	$90	
MenuBoard Utilization	0.998418	1:00:40
Kitchen Utilization	0.703890	1:00:40
OrderWindow Utilization	0.968154	1:00:40
MenuBoard Waiting in Line	1.239055	1:00:40
Kitchen Waiting in Line	0.092286	1:00:40
OrderWindow Waiting in Line	1.041747	1:00:40
Time in System	5.613265	45

5 INTERPRETING OUTPUT STATISTICS

Let us assume that our model has run from 11:00 AM to 2:00 PM. At 2:00 PM, we stop the simulation and we have all of our statistics calculated. The "answer" for the simulation is in Table 7.

Table 7: Statistics at End of Simulation

Statistic	Value	Time/Obs
Revenue	$ 1,638	
Lost Revenue	$ 170	
MenuBoard Utilization	0.9724	3:00:00

Kitchen Utilization	0.7250	3:00:00
OrderWindow Utilization	0.9846	3:00:00
MenuBoard Waiting in Line	0.8785	3:00:00
Kitchen Waiting in Line	0.133	3:00:00
OrderWindow Waiting in Line	1.1567	3:00:00
Time In System	5.0487	140

But what does this data really telling us? Is it saying that every day, from 11:00 AM to 2:00 PM, the revenue for the restaurant will be $1,638? Is it saying that every day, from 11:00 AM to 2:00 PM, the average amount of time that a car will be in line is 5.0487 minutes? What is this really saying?

The answer to that question is, "These numbers give us a random answer to the performance of the system." It is a random answer because there are random inputs to the system that give us this answer. So how can we be sure that the answer is any good?

The answer generated by only one run is not really an answer at all. To make the point, let us take the following 100 rolls of the dice. You would think that 100 rolls of the dice would tell us the average. But if you take the 100 rolls in Table 8, the average is only 6.72. Is that close enough? Would you be willing to bet that the next 100 rolls would come up with an average of 6.72. Probably not. (If you are, please contact me as soon as possible on any other wagers you may be considering.)

Table 8: 100 Rolls of the Dice

6	5	8	6	5	4	10	6	7	8
5	9	8	7	8	6	3	8	7	6
9	9	8	8	6	8	9	7	10	5
2	10	11	8	6	8	7	3	8	8
4	6	8	11	2	4	8	9	8	5
9	3	8	7	2	3	9	10	7	7
3	9	5	7	7	7	9	4	8	7
4	10	7	4	10	8	4	8	9	7
3	6	6	3	6	3	10	9	7	4
6	8	5	9	12	6	8	6	4	2
Average: 6.72									

So what is the answer to this problem. The answer is that if we really want to estimate the average value of the roll of the dice if we roll the dice 100 times, we need to run the simulation more times. Each time that we run a simulation is called an *iteration*. So, as an example, let us say that we have run our dice throwing simulation for 30 iterations and Table 9 has the average values for each of those iterations.

Table 9: 30 Iterations of Throwing Dice 100 Times

6.72	6.95	6.78	7.14	6.62	6.81
6.75	7.17	6.62	7.3	6.92	7.04
6.79	7.13	7.17	7.12	6.82	7.29
7.13	7.26	7.19	6.52	6.8	7.3
6.95	6.96	6.78	7.17	7.13	6.68
Average: 6.967 **Standard Deviation: 0.22992** **95% Confidence Interval: [6.8847,7.0493]**					

If we are simply trying to estimate the average of 100 rolls of the dice, we do not need to worry about the standard deviation for each iteration. However, we do need to worry about the standard deviation of the averages from each iteration. As is shown in Table 9, the average of the averages (so to speak) is 6.967. The standard deviation of those 30 averages is 0.22992. With that information, we can calculate a *confidence interval*.

A *confidence interval* is a statistical measure where we want to bound some statistic. The level of confidence (95% in our example) is the statistical probability that the statistic that we are considering lies in the interval. So, the interpretation of the 95% confidence interval of [6.8847,7.0493] would be "There is a 95% chance that the true average of rolling the dice 100 times lies between 6.8847 and 7.0493." Most statistical and simulation packages automatically calculate confidence intervals for you. Even Microsoft Excel has a function to calculate confidence intervals.

So in our example, we want to run 30 iterations so that we can have good confidence intervals for each of our statistics. (Although we will not get into the topic in this paper, one should run 30 iterations (or more) if you can in order to get good confidence intervals.) Table 10 shows the confidence intervals for each of our statistics.

Table 10: Confidence Intervals after 30 Iterations

Statistic	Value	CI Lower	CI Upper
Revenue	$1,649	$1,627	$1,670
Lost Revenue	$155	$135	$176
Lost Customers	12.83	10.99	14.67
MenuBoard Utilization	0.97	0.96	0.98
Kitchen Utilization	0.73	0.72	0.74
OrderWindow Utilization	0.99	0.99	0.99
MenuBoard Waiting in Line	0.95	0.89	1.01
Kitchen Waiting in Line	0.12	0.11	0.13
OrderWindow Waiting in Line	1.23	1.18	1.28
Time In System	5.31	5.18	5.44

These statistics are very interesting on several fronts. First, could improve our revenue every day at lunch 9.4% if we could capture the lost revenue. We need to find a way to lower the number of lost customers from the 12.83 that we have seen in the current system. We also know that both the *Menu Board* and *Order Window* are highly utilized, which means that lines are forming in front of those two resources. We would also want to look at reducing the car's time in the system. The minimum average theoretical time for a car would be 70 seconds at the *Menu Board* and 70 seconds at the *Order Window*, which is 140 seconds, or 2.33 minutes. So, about 3 minutes of the car's time is simply waiting. A key customer satisfaction metric is to minimize the amount of time they have to wait in line.

6 FINDING WAYS TO IMPROVE A SYSTEM

What we have accomplished to this point is the analysis of a system that exists. We have learned that we can improve revenue and that we have very busy resources. So let us run some alternative scenarios to determine what (if any) improvements we should make.

6.1 Scenario 1: No Lines

Scenario 1 is based on the "eliminating inventory" thought. If we want to get the customer out of the system quicker, we would simply need to eliminate some (or all) of the line. We'll take drastic action and eliminate all waiting in the system. If a car cannot pull right up to the *Menu Board,* then it will leave. If the *Order Window* is not available after the order is placed, then the car waits at the *Menu Board* until the

Order Window is free. This should greatly lower the *Time in System* statistic. Some would say that this would be the best way to run the drive-through line since the system is *balanced*. The arrival rate is 1 every 70 seconds, the *Menu Board* rate is 1 every 70 seconds, and the *Order Window* rate is 1 every 70 seconds. Some would say that this is a perfect production line.

In order to accomplish this in the model, we simply change 2 variables, *MenuBoardLineSize* and *OrderPickupLineSize*, and set them both to a value of 1. Then we run the model for 30 iterations and get the statistics in Table 11.

Table 11: Scenario 1 Statistics

Statistic	Value	CI Lower	CI Upper
Revenue	$989	$968	$1,010
Lost Revenue	$851	$827	$875
Lost Customers	70.43	68.87	71.99
MenuBoard Utilization	0.68	0.67	0.69
Kitchen Utilization	0.43	0.42	0.44
OrderWindow Utilization	0.84	0.83	0.85
MenuBoard Waiting in Line	0.00	0.00	0.00
Kitchen Waiting in Line	0.00	0.00	0.00
OrderWindow Waiting in Line	0.00	0.00	0.00
Time In System	3.31	3.28	3.34

Well, the strategy worked the way we expected it to work. We reduced the *Time in System* statistic by 37.7%. Now the cars are whipping through the drive-through. But there is a catch. We're losing 5.5 times the number of customers that we were losing before! We've lowered our revenues by 40%! This would hardly be considered a viable alternative!

What was it about eliminating the lines that caused such a problem? Without the lines, we caused excessive *blocking* in the system. Blocking occurs when something in the system will not allow a resource to become available even though its service is finished. Throughout this scenario, the car at the *MenuBoard* was finished ordering, but the car could not move forward because the *Order Window* was still occupied. In this scenario (or in any balanced system with random variation and no queuing space), each car at the *MenuBoard* has a 50% chance of being blocked by the *OrderWindow*. This can be reduced by adding queues between the resources.

In industrial problems, it is often difficult to determine where and why blocking occurs. That is why there is so much emphasis on eliminating and/or controlling the *bottleneck* of the system. It is likely that the bottleneck is in large part responsible for blocking in other parts of the system.

6.2 Scenario 2: Longer Lines at the Menu Board

Well, since Scenario 1 did not work very well, let us try another. Let us rearrange the parking lot so that we can get room for 6 cars waiting in line for the Menu Board (including the car whose order is being taken). We will keep the line for 3 cars for the Order Pickup Window. The results for Scenario 2 is shown in Table 12.

Table 12: Scenario 2 Statistics

Statistic	Value	CI Lower	CI Upper
Revenue	$1,690	$1,665	$1,715
Lost Revenue	$99	$77	$122
Lost Customers	8.50	6.73	10.27

MenuBoard Utilization	0.99	0.99	0.99
Kitchen Utilization	0.73	0.72	0.74
OrderWindow Utilization	0.99	0.99	0.99
MenuBoard Waiting in Line	3.07	2.90	3.24
Kitchen Waiting in Line	0.13	0.12	0.14
OrderWindow Waiting in Line	1.28	1.23	1.33
Time In System	6.33	6.14	6.52

Well, we did add revenue ($51 per day) and decrease the number of lost customers, that is good, but now the average customer waits an additional minute to get the order, which is bad. You can see why this occurs. With the Menu Board line being longer, more cars can wait for the menu board and they do not become lost customers in this scenario, where they would have become lost customers in the original model. The price they pay for having room to get in line is that the line is longer, and so the time through the system increases. This is another typical trade-off. Queues are good because they allow more throughput in a system, but they add cycle time for the system.

6.3 Scenario 3: Improved Service Times

Let us try one more scenario. This scenario has found new technology that will cut the average service time at the Menu Board and the Order Window by 20% from 70 seconds to 56 seconds. To implement this technology, it would take $30,000 at each store and must be paid for by increased revenue at the store. This is the only thing that changes from the original model. The results are shown in Table 13.

Table 13: Scenario 3 Statistics

Statistic	Value	CI Lower	CI Upper
Revenue	$1,822	$1,794	$1,850
Lost Revenue	$10	$5	$15
Lost Customers	0.70	0.33	1.07
MenuBoard Utilization	0.81	0.80	0.82
Kitchen Utilization	0.79	0.78	0.80
OrderWindow Utilization	0.97	0.97	0.97
MenuBoard Waiting in Line	0.26	0.23	0.29
Kitchen Waiting in Line	0.23	0.21	0.25
OrderWindow Waiting in Line	0.85	0.82	0.88
Time In System	3.42	3.36	3.48

Well, now we see some improvement! The *Time in System* has dropped from 5.31 minutes to 3.42 minutes. We have virtually eliminated any lost customers, and have increased revenue by $174 per lunch shift. We would pay back the $30,000 for the implementation of the new technology in 172 days or less than 6 months! It is well worth the investment and you are a hero for figuring it out!

Why did this scenario show such improvement? It was because our resources were not so highly loaded. The *Menu Board* utilization dropped from 97% to 81%, and so the line for the Menu Board would be much smaller (it dropped by 72%). We would have seen a similar drop in the *Order Window* utilization, but now the Order Window is being often being blocked by the Kitchen.

6.4 Generating Other Scenarios

This simulation and any other simulation can be used to evaluate many different scenarios if the person creating the model allows some flexibility in the model structure. One also has to consider the amount of

time it takes to run a new scenario. In a large scale simulation, to run and evaluate a new scenario could take several days.

7 WHAT HAVE WE LEARNED?

This exercise points out several things about simulation in general. First, simulation can mimic the *dynamic* behavior of a system. That is what it is built to do. Regardless of how complex a system may be, it is likely that a simulation *expert* will be able to create a model that will evaluate it. However, the more complex a system is, the longer it takes to model, run and evaluate. But do not be discouraged; there are very good simulation people available to model large systems.

Second, you (or the person analyzing the system) must have a good understanding of simulation statistics. It is important during the creation of the model so that input distributions are used properly. It is important during the analysis of the output statistics so that the output is not misinterpreted. Mistakes with either the inputs or the outputs will cause the simulation analysis to be invalid.

Third, to analyze a system, simulation is used to evaluate different scenarios. It does not choose the best scenario for you. This may seem to be a problem, but most managers have no shortage of scenarios to evaluate. The trade off for this is that you can analyze the *dynamics* of the system and not just the average behavior.

Fourth, the scenarios that you do choose are generated by you and not the system. This is where familiarity with the system under study and a familiarity with system dynamics concepts are very valuable.

This is, of course, simply an introduction. Through this conference and interaction with simulation professionals, you can get a deeper understanding of simulation and what it can do for you.

REFERENCES

Banks, J., J.S. Carson II, B.L. Nelson, and D.M. Nicol. 2000. *Discrete Event System Simulation*, 3rd Ed., Prentice-Hall.

Law, A.M. and W.D. Kelton. 2000. *Simulation Modeling and Analysis*, 3rd Ed., McGraw-Hill.

Kelton, W.D., R. Sadowski, D. Sadowski. 2001. *Simulation with Arena, 2nd Edition*, McGraw-Hill.

Ingalls, R.G, C. Kasales, 1999. "CSCAT: Compaq Supply Chain Analysis Tool." In *Proceedings of the 1999 Winter Simulation Conference*, edited by P.A. Farrington, H.B. Nembhard, D.T. Sturrock and G.W. Evans. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Ingalls, R.G., 1998. "The Value of Simulation in Modeling Supply Chains." In *Proceedings of the 1998 Winter Simulation Conference*, edited by D.J. Medeiros, E.F. Watson, J.S. Carson, and M.S. Manivannan. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Ingalls, R.G., C. Eckersley, 1992. Simulation Issues in Electronics Manufacturing. *Proceedings of the 1992 Winter Simulation Conference*. ed. J.J. Swain, D. Goldsman, R.C. Crain and J.R. Wilson. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Shannon, Robert E., 1975. *Systems Simulation – The Art and Science*. Prentice-Hall.

AUTHOR BIOGRAPHY

RICKI G. INGALLS is Associate Professor and Site Director of the Center for Engineering Logistics and Distribution (CELDi) in the School of Industrial Engineering and Management at Oklahoma State University. He is also a Founding Principal with Diamond Head Associates, Inc. He joined OSU in the Fall of 2000 after 16 years in industry with Compaq, SEMATECH, General Electric and Motorola. He has a B.S. in Mathematics from East Texas Baptist College (1982), a M.S. in Industrial Engineering from Texas A&M University (1984) and a Ph.D. in Management Science from the University of Texas at Austin (1999). His research interests include the supply chain design issues and the development and application of qualitative discrete-event simulation. He is a member of IIE. His email address is <ricki.ingalls@okstate.edu>.