37

Using Queues to Model a Merchant's Inventory

John Manslow

37.1 Introduction

Queues frequently appear in game worlds. Sometimes they are obvious—like the queues formed by cars in a traffic jam or patients waiting to see a doctor. At other times, they are more difficult to identify—like the queues formed by items in the inventory of a merchant. M/M/1 queues arise when objects are added to, and removed from, a collection at random time intervals and can be used to model such processes even when the order of the objects is not important.

This chapter will describe how to use the statistical properties of M/M/1 queues to efficiently simulate how they change over time. To provide a practical example, the chapter will show how to represent the inventory of a merchant as a set of M/M/1 queues and demonstrate how to efficiently generate realistic and mutually consistent random realizations of the inventory that take account of the levels of supply and demand. The book's website (http://www.gameaipro.com) includes a full implementation of everything that is described in the article.

37.2 M/M/1 Queues

An M/M/1 queue is a process in which objects are added to, and removed from, a collection at random intervals. The name M/M/1 is actually a form of Kendall's notation [Zabinsky 13], which is used to classify queues and indicates that additions and removals are memoryless (the time of one addition or removal does not affect the time of another) and that the queue consists of only a single collection. The random timings of the additions and removals mean that the number of objects in the collection—the length of the queue—changes randomly with time. Although this makes it impossible to predict the length of the queue with any certainty, it is possible to model it using two probability distributions, the stationary distribution and the transient distribution.

The stationary distribution models the length of queue that one would expect to see if the queuing process had been running for a very long time before it was observed. It is therefore useful for describing the length of a queue when it is encountered by the player for the first time. If objects are added to a queue at an average rate of *add_rate* per unit time and removed from it at an average rate of *remove_rate* per unit time, then its stationary distribution is

$$p(n) = \left(1 - rate\_ratio\right) \cdot rate\_ratio^{n} \tag{37.1}$$

where

$$rate\_ratio = \frac{add\_rate}{remove\_rate} \tag{37.2}$$

and $p(n)$ is the probability of the length of the queue being n items.

Note that, for a queue to be of finite length, it is necessary for *rate_ratio* to be strictly less than one. In other words, if the merchant's inventory isn't going to grow without bound, things need to be leaving the store faster than they are coming in. In general, the average length of an M/M/1 queue is 1/(1 − *rate_ratio*), and the probability of observing an empty queue is 1 − *rate_ratio.*

Players naturally expect to encounter queues of specific lengths and not strange quantum queues in weird superpositions of states. It is therefore necessary to create a specific realization of a queue when the player first observes it by sampling from its stationary distribution. A simple way to do that is to generate a random number x that is greater than or equal to zero and less than one and, starting at $n = 0$, add the values of $p(n)$ until their sum exceeds x and then take n to be the length of the queue. This approach is implemented in `MM1Queue::GetSample`.

The transient distribution describes the length of a queue that has already been observed and is therefore useful when a player encounters a queue for anything other than the first time. If, t units of time ago, the player observed a queue to be of length m, its transient distribution is [Baccelli 89]

$$p\left(n \mid t, m\right) = e^{-(add\_rate + remove\_rate) \cdot t} \cdot \left[p_1 + p_2 + p_3\right] \tag{37.3}$$

where

$$p_1 = rate\_ratio^{\frac{n-m}{2}} \cdot I_{n-m}(at) \tag{37.4}$$

$$p_2 = rate\_ratio^{\frac{n-m-1}{2}} \cdot I_{n+m+1}(at) \tag{37.5}$$

$$p_3 = (1 - rate\_ratio) \cdot rate\_ratio^{n} \sum_{j=n+m+2}^{\infty} rate\_ratio^{-\frac{j}{2}} \cdot I_{n+m+1}(at) \tag{37.6}$$

and

$$a = 2 \times remove\_rate\sqrt{rate\_ratio}, \tag{37.7}$$

$I_n(\cdot)$ is the modified Bessel function of the first kind, which is computed by `MathUtilities::Iax`, and $p(n|t,m)$ is the probability that the length of the queue is n, which is computed by `MM1Queue::GetStateProbability`. Once again, it is necessary to create a specific realization of the queue, and that can be done using the procedure that has already been described, but substituting $p(n|t,m)$ for $p(n)$. This approach is implemented in the overload of `MM1Queue::GetSample` that takes time and count parameters.

Because both the stationary and transient distributions are derived from a statistical model of the queuing process, samples that are drawn from them naturally satisfy all common sense expectations as to how queues change over time. In particular,

- Queues with high rates of addition and low rates of removal will usually be long
- Queues with low rates of addition and high rates of removal will usually be short
- The effects of player interactions will disappear with time—slowly for queues with low rates of addition and removal and quickly for queues with high rates of addition and removal
- Queues will change little if the time between observations is short, especially for queues with low rates of addition and removal
- Queues will change a lot if the time between observations is long, especially for queues with high rates of addition and removal

The first row of Table 37.1 gives the stationary distribution for a queue with *add_rate* = 0.5 (one addition every other unit of time on average) and *remove_rate* = 1.0 (one removal per unit of time on average). Assuming that the player observes the queue to be of length two at time zero and adds four objects to it, the following rows show the transient distribution 0.01, 0.1, 1, 10, and 100 units of time later. It is important to note that these rows show how the transient distribution would evolve if the queue remained unobserved and hence no concrete realizations were produced. For example, 10 units of time after the player had observed the queue, there would be a 0.103 probability of the queue being 3 objects long.

Table 37.1 An Example of an Equilibrium Distribution and Transient Distributions

	Queue Length, n								
Time, t	0	1	2	3	4	5	6	7	8
0.00	0.500	0.250	0.125	0.063	0.031	0.016	0.008	0.004	0.002
0.01	0.000	0.000	0.000	0.000	0.000	0.010	0.985	0.005	0.000
0.10	0.000	0.000	0.000	0.000	0.004	0.087	0.865	0.043	0.001
1.00	0.000	0.002	0.010	0.042	0.131	0.284	0.349	0.142	0.033
10.00	0.315	0.186	0.131	0.103	0.082	0.063	0.046	0.031	0.020
100.00	0.500	0.250	0.125	0.063	0.031	0.016	0.008	0.004	0.002

The transient distribution makes it possible to model how the length of a queue changes after it has been observed but provides no way of determining how many of the individual objects that were originally in the queue are still there when it is observed again. That kind of information is useful when the objects are uniquely identifiable. For example, cars parked on a street will have different colors and be different makes and models, and people waiting in a hospital will have different clothing and facial features.

If the objects are processed on a strictly first-come, first-served basis, then the number that remain in the queue from the previous observation can roughly be estimated by taking the original length of the queue and subtracting a sample from a Poisson distribution that represents the number of objects processed since the queue was last observed.

If the objects are processed in a random order, the number that remain can be approximated by taking a sample from the same Poisson distribution and then using it in conjunction with a binomial distribution to estimate the number of objects in the original queue that were processed. Technically, this is equivalent to assuming that the length of the queue remained the same between the two observations, but it produces realistic-looking estimates even when that is not actually the case. Implementations of both techniques can be found in the `Inventory::GenerateRealization` overload that takes the time parameter.

Before finishing the discussion of M/M/1 queues, it is important to highlight the fact that the expressions for the stationary and transient distributions assume that the average rates at which objects are added to, and removed from, the queue are constant. This assumption holds for many natural processes but breaks down when additions or removals occur in bursts. Such bursts will occur in relation to the numbers of cars waiting at an intersection, for example, due to the presence of a neighboring intersection or the effects of traffic signals. In such cases, the theoretical deficiencies of M/M/1 queues will often not be apparent to the player and can be ignored, but, in some cases, it will be necessary to use an alternative model [Zabinsky 13].

This section has described the basic properties of M/M/1 queues, given expressions for the stationary and transient distributions of queue length, and shown how to sample from those distributions to generate consistent realizations when queues are encountered by the player. The following section will describe how M/M/1 queues can be used to model the inventory of a merchant to produce random inventories that are consistent with each other, with the levels of supply and demand of each type of item in the inventory and with the player's observations and interactions with the merchant.

37.3 Modeling a Merchant's Inventory

In a large, open world game with dozens of merchants, hundreds of NPCs, and tens of thousands of individual inventory items, it is impractical to explicitly model economic activity in real time in enough detail to track the inventory of each merchant. Fortunately, to make the world believable, it is sufficient to generate random inventories each time a merchant is encountered provided that they are consistent with players' common sense expectations as to how they should change with time.

In terms of the numbers of each type of item in an inventory, those expectations are essentially the same as those for the lengths of queues that were given earlier. This strongly suggests that a merchant's inventory can be modeled in the following way:

1. Create one M/M/1 queue for each type of item the merchant can have in his or her inventory.
2. Set the average rate at which each type of item is added to its queue to be equal to the average rate at which the merchant will buy it.
3. Set the average rate at which each type of item is removed from its queue to be equal to the average rate at which the merchant will sell it when he or she has it.
4. When the player first encounters the merchant, create his or her inventory by sampling from the stationary distribution for each type of item.
5. On all subsequent encounters, create his or her inventory by sampling from the transient distribution for each type of item.

Even though real merchants do not buy and sell at random, steps 2 and 3 ensure that the merchant's inventory is generally consistent with the levels of supply and demand for each type of item, and step 5 ensures that players see an inventory that is consistent from one visit to the next.

Table 37.2 gives an example of how the numbers of three inventory items—truffles, arrows, and swords—vary with time, which, in this example, is measured in game world hours. Truffles are assumed to have limited supply (only one is added to the inventory every 1000 h on average) but high demand (one is sold every hour on average), arrows are assumed to have high supply (one is added every hour on average) and high demand (one is sold every 0.98 h on average), and swords are assumed to have low supply (one is added every 200 h on average) and low demand (one is sold every 100 h on average).

Table 37.2 An Example of How a Simple Three-Item Inventory Changes with Time

	Item		
Time (h)	Truffles	Arrows	Swords
0	0	83	1
0 after player interaction	5	33	1
1	3	33	1
48	0	37	1
100	0	26	0
200	0	47	0

The numbers of truffles, arrows, and swords at time zero—when the player encounters the merchant for the first time—are obtained by sampling from each type of item's stationary distribution. The player sells the merchant five truffles and buys 50 arrows and then explores the environment for 1 h. He or she then returns to the merchant, and a new inventory is generated by sampling from each type of item's transient distribution. This reveals that the merchant still has three truffles left and the number of arrows and swords hasn't changed. Returning to the merchant after 48 h reveals that all truffles have been sold and the merchant has 37 arrows. The code that was used to generate the numbers in this example is included on the book's website.

The basic inventory model that has been described so far can easily be enhanced to simulate more complex behavior. For example, a merchant might buy 50 arrows every Monday but only if he has fewer than 50 arrows in stock. This kind of behavior can be closely approximated by sampling from the distribution for the number of arrows in the merchant's inventory from the previous Monday and adding 50 if the sample is less than 50. The resulting number can then be used as a "virtual observation" when sampling from the current transient distribution to obtain the current number of arrows—the game simply behaves as though the player had been present the previous Monday and seen how many arrows the merchant had.

Similar logic can be used if the merchant always buys enough arrows to bring his or her stock up to 50, if the arrow vendor comes randomly rather than every Monday, or if the supply of arrows is not entirely dependable. The merchant buying additional stock is only one type of special event that affects the numbers of items in his inventory. Another might be the death of a nobleman, causing the merchant to suddenly acquire a large number of luxury items at the resulting estate sale or the commander of a local garrison buying up all of the armor. Such events can be modeled in a similar way to the merchant's buying behavior; a virtual observation of the affected inventory item can be created for the time of the event and used in the transient distribution when the player encounters the merchant.

Other events might cause permanent changes in the levels of supply and demand, and they can be simulated by changing *add_rate* and *remove_rate*. For example, a new mine might open up, leading to an increase in the supply of iron. This effect can be simulated by making a virtual observation of the amount of iron that the merchant had in stock when the mine opened by sampling from a distribution with the old values of *add_rate* and *remove_rate*. That observation would then be used in the transient distribution with the new values of *add_rate* and *remove_rate* when the player encountered the merchant. If the levels of supply and demand change multiple times between encounters, the effects of the changes can be simulated by multiple virtual observations that are obtained using the previous observation, the previous values of *add_rate* and *remove_rate*, and the sampling from the transient distribution. The game would thus behave as though the player had observed the level of stock of the affected type of item each time its supply and demand changed.

In some cases, it is desirable to ensure that a merchant always has a certain minimum number of items of a particular type in stock. If a game directs the player to travel a long way to buy special items, for example, it would be very frustrating to arrive at the destination only to discover that the items were not available. This problem can easily be solved by adding a constant to the number of items generated by the stationary distribution on the player's first encounter. If the merchant should generally maintain a certain minimum stock level, then adding a constant is unsatisfactory because it does not adequately model the dynamics of how stock changes in response to interactions with the player—if the player buys all the stock, for example, the amount of stock needs to recover in a convincing way.

The simplest solution to this problem is to model the merchant regularly buying new stock, as was described earlier. Alternately, it is possible to create a reserve of items that can only be purchased by the player and model how it recovers over time if the player makes a purchase that depletes it. This is done by estimating the number of items that could have been added to the reserve since it was depleted if the merchant repopulated it by buying items at a rate of *add_rate* and selling nothing. If the reserve could only have been partially repopulated, the reserve is the full extent of the inventory, and no sample from the transient distribution is required. If the reserve could have been fully repopulated, however, the time when the process of repopulation would've been completed is calculated, and the number of nonreserve items is obtained by sampling from the transient distribution using a virtual observation of zero nonreserve items backdated to when the reserve would've reached full strength. This technique is implemented in the overload of `Inventory::GenerateRealization` that takes the time parameter.

Finally, some types of items, such as arrows, are naturally traded in batches, and it is unlikely that a merchant would buy or sell only a single instance of such types. This effect can be approximated by using stationary and transient distributions to represent the numbers of batches held by the merchant rather than the numbers of individual items. When the number of batches changes from the player's last observation, the number of items can be generated randomly by assuming that a particular number of batches would correspond to a particular range of numbers of items. For example, if each batch of arrows is of size 25, then one batch would correspond to between 1 and 25 arrows, two batches, 26 and 50 arrows, etc. If a sample from the distribution specified a stock level of two batches, the actual number of items would be chosen randomly from the range 26 to 50.

In general, the properties of M/M/1 queues that were described earlier make it possible to guess values for parameters like *add_rate* and *remove_rate* to simulate specific behaviors. It is, however, important to validate those behaviors using a simple test application like the one included on the book's website that allows the behaviors to be quickly and efficiently evaluated over a wide range of timescales.

37.3.1 Computational Considerations

A game might contain many thousands of different types of items that could potentially be found in an inventory, so the question naturally arises as to whether it's computationally practical to sample from such a large number of queues. Fortunately, for types where *rate_ratio* is small (i.e., for types that are unlikely to appear in the inventory or to only be present in small numbers—such as truffles and swords), samples can be obtained at a rate of hundreds of thousands per second per core on a typical desktop PC. Where *rate_ratio* is close to one—as was the case with the arrows—samples can only be obtained at a rate of thousands per second, so the approach described in this chapter might not be suitable for inventories where hundreds of different types of items are likely to be present in hundreds or thousands. Such inventories are likely to be the exception, however, and it is important to remember that samples are only required when a player encounters a merchant—there's no ongoing computation—and that samples for each type of item are independent, and hence the process of sampling can, if necessary, easily be distributed across multiple cores and multiple frames.

37.4 Conclusion

This article has described M/M/1 queues and showed how they can be simulated in a consistent and computationally efficient way by sampling from their stationary and transient distributions. It has shown how they can be used to represent the inventory of a merchant in such a way that it remains consistent with each item's supply and demand, the player's observations of the inventory, and the player's interactions with the merchant. This provides a simple and efficient way to simulate how the inventory of a merchant changes with time.

References

[Baccelli 89] Baccelli, F., Massey, W. A. 1989. A sample path analysis of the M/M/1 queue. *Journal of Applied Probability*, 26(2): 418–422. https://www.princeton.edu/~wmassey/20th%20Century/sample%20path%20MM1.pdf (accessed July 20, 2014).

[Zabinsky 13] Zabinsky, Z. 2013. Stochastic models and decision analysis, University of Washington, Seattle, WA. http://courses.washington.edu/inde411/QueueingTheory.pdf (accessed July 20, 2014).